# NebulaGraph Database Manual

v3.6.0

NebulaGraph

2023 Vesoft Inc.

# Table of contents

1. W	elcome to NebulaGraph 3.6.0 Documentation	6
1.1	Getting started	6
1.2	Release notes	6
1.3	Other Sources	6
1.4	Symbols used in this manual	6
1.5	Modify errors	7
2. In	troduction	8
2.1	What is NebulaGraph	8
2.2	Data modeling	12
2.3	Path types	14
2.4	VID	16
2.5	NebulaGraph architecture	18
3. Qı	uick start	37
3.1	Quickly deploy NebulaGraph using Docker	38
3.2	Deploy NebulaGraph on-premise	41
3.3	nGQL cheatsheet	59
4. nC	GQL guide	81
4.1	nGQL overview	81
4.2	Data types	98
4.3	Operators	117
4.4	Functions and expressions	131
4.5	General queries statements	174
4.6	Clauses and options	227
4.7	Variables and composite queries	259
4.8	Space statements	266
4.9	Tag statements	276
4.10	0 Edge type statements	286
4.11	1 Vertex statements	292
4.12	2 Edge statements	300
4.13	3 Native index statements	307
4.14	4 Full-text index statements	319
4.15	5 Query tuning and terminating statements	329
4.16	6 Job manager and the JOB statements	335
5. De	eploy and install	339
5.1	Prepare resources for compiling, installing, and running NebulaGraph	339

5.2 Compile and install	345
5.3 Local single-node installation	351
5.4 Deploy a NebulaGraph cluster with RPM/DEB package on multiple servers	358
5.5 Deploy NebulaGraph with Docker Compose	364
5.6 Deploy NebulaGraph with NebulaGraph Lite	370
5.7 Install NebulaGraph with ecosystem tools	371
5.8 Manage NebulaGraph Service	372
5.9 Connect to NebulaGraph	374
5.10 Manage Storage hosts	376
5.11 Upgrade NebulaGraph to 3.6.0	378
5.12 Uninstall NebulaGraph	383
6. Configure and log	385
6.1 Configurations	385
6.2 Log management	416
7. Monitor	422
7.1 Query NebulaGraph metrics	422
7.2 RocksDB statistics	430
8. Data security	432
8.1 Authentication and authorization	432
8.2 SSL encryption	438
9. Backup and restore	440
9.1 NebulaGraph BR Community	440
9.2 Backup and restore data with snapshots	452
10. Synchronize and migrate	455
10.1 BALANCE syntax	455
11. Import and export	456
11.1 Import tools	456
11.2 NebulaGraph Importer	458
11.3 NebulaGraph Exchange	468
12. Connectors	574
12.1 NebulaGraph Spark Connector	574
12.2 NebulaGraph Flink Connector	581
13. Best practices	587
13.1 Compaction	587
13.2 Storage load balance	589
13.3 Graph data modeling suggestions	590
13.4 System design suggestions	594
13.5 Execution plan	595

13.7 Enable AutoFDO for NebulaGraph	EOO				
	Enable AutoFDO for NebulaGraph 598				
13.8 Best practices	604				
14. Clients	605				
14.1 Clients overview	605				
14.2 NebulaGraph Console	606				
14.3 NebulaGraph CPP	611				
14.4 NebulaGraph Java	613				
14.5 NebulaGraph Python	615				
14.6 NebulaGraph Go	617				
14.7 Community contributed clients	618				
15. Studio	619				
15.1 About NebulaGraph Studio	619				
15.2 Deploy and connect	622				
15.3 Quick start	633				
15.4 Troubleshooting	658				
16. Dashboard (Community)	662				
16.1 What is NebulaGraph Dashboard Community Edition	662				
16.2 Deploy Dashboard Community Edition	664				
16.3 Connect Dashboard	667				
16.4 Dashboard	668				
16.5 Metrics	674				
17. NebulaGraph Operator	682				
17.1 What is NebulaGraph Operator	682				
17.2 Getting started	684				
17.3 NebulaGraph Operator management	700				
17.4 Cluster administration	709				
17.5 FAQ	756				
18. Graph computing	758				
18.1 NebulaGraph Algorithm	758				
19. NebulaGraph Bench	765				
19.1 Scenario	765				
19.2 Release note	765				
19.3 Test process	765				
20. FAQ	766				
20.1 About manual updates	766				
20.2 About legacy version compatibility	766				
	766				

20.4 About design and functions	769
20.5 About operation and maintenance	772
20.6 About connections	776
21. Appendix	778
21.1 Release Note	778
21.2 Ecosystem tools overview	781
21.3 Port guide for company products	785
21.4 How to Contribute	788
21.5 History timeline for NebulaGraph	792
21.6 Error code	798

# 1. Welcome to NebulaGraph 3.6.0 Documentation

#### Q Note

This manual is revised on 2024-5-9, with GitHub commit e0affdbb41.

NebulaGraph is a distributed, scalable, and lightning-fast graph database. It is the optimal solution in the world capable of hosting graphs with dozens of billions of vertices (nodes) and trillions of edges (relationships) with millisecond latency.

# 1.1 Getting started

- Quick start
- Preparations before deployment
- nGQL cheatsheet
- FAQ
- Ecosystem Tools
- Live Demo

## 1.2 Release notes

- NebulaGraph Community Edition 3.6.0
- NebulaGraph Dashboard Community
- NebulaGraph Studio

# 1.3 Other Sources

- To cite NebulaGraph
- Forum
- NebulaGraph Homepage
- Blogs
- Videos
- Chinese Docs

# 1.4 Symbols used in this manual

# Note

Additional information or operation-related notes.

# Caution

May have adverse effects, such as causing performance degradation or triggering known minor problems.

## 

May lead to serious issues, such as data loss or system crash.

# **B**anger

May lead to extremely serious issues, such as system damage or information leakage.

# Empatibility

The compatibility notes between nGQL and openCypher, or between the current version of nGQL and its prior ones.

# Sterpriseonly

Differences between the NebulaGraph Community and Enterprise editions.

# 1.5 Modify errors

This NebulaGraph manual is written in the Markdown language. Users can click the pencil sign on the upper right side of each document title and modify errors.

Last update: November 29, 2023

# 2. Introduction

# 2.1 What is NebulaGraph

NebulaGraph is an open-source, distributed, easily scalable, and native graph database. It is capable of hosting graphs with hundreds of billions of vertices and trillions of edges, and serving queries with millisecond-latency.



### 2.1.1 What is a graph database

A graph database, such as NebulaGraph, is a database that specializes in storing vast graph networks and retrieving information from them. It efficiently stores data as vertices (nodes) and edges (relationships) in labeled property graphs. Properties can be attached to both vertices and edges. Each vertex can have one or multiple tags (labels).



Graph databases are well suited for storing most kinds of data models abstracted from reality. Things are connected in almost all fields in the world. Modeling systems like relational databases extract the relationships between entities and squeeze them into table columns alone, with their types and properties stored in other columns or even other tables. This makes data management time-consuming and cost-ineffective.

NebulaGraph, as a typical native graph database, allows you to store the rich relationships as edges with edge types and properties directly attached to them.

#### 2.1.2 Advantages of NebulaGraph

#### Open source

NebulaGraph is open under the Apache 2.0 License. More and more people such as database developers, data scientists, security experts, and algorithm engineers are participating in the designing and development of NebulaGraph. To join the opening of source code and ideas, surf the NebulaGraph GitHub page.

#### Outstanding performance

Written in C++ and born for graphs, NebulaGraph handles graph queries in milliseconds. Among most databases, NebulaGraph shows superior performance in providing graph data services. The larger the data size, the greater the superiority of NebulaGraph.For more information, see NebulaGraph benchmarking.

#### High scalability

NebulaGraph is designed in a shared-nothing architecture and supports scaling in and out without interrupting the database service.

#### **Developer friendly**

NebulaGraph supports clients in popular programming languages like Java, Python, C++, and Go, and more are under development. For more information, see NebulaGraph clients.

#### **Reliable access control**

NebulaGraph supports strict role-based access control and external authentication servers such as LDAP (Lightweight Directory Access Protocol) servers to enhance data security. For more information, see Authentication and authorization.

#### **Diversified ecosystem**

More and more native tools of NebulaGraph have been released, such as NebulaGraph Studio, NebulaGraph Console, and NebulaGraph Exchange. For more ecosystem tools, see Ecosystem tools overview.

Besides, NebulaGraph has the ability to be integrated with many cutting-edge technologies, such as Spark, Flink, and HBase, for the purpose of mutual strengthening in a world of increasing challenges and chances.

#### OpenCypher-compatible query language

The native NebulaGraph Query Language, also known as nGQL, is a declarative, openCypher-compatible textual query language. It is easy to understand and easy to use. For more information, see nGQL guide.

#### Future-oriented hardware with balanced reading and writing

Solid-state drives have extremely high performance and they are getting cheaper. NebulaGraph is a product based on SSD. Compared with products based on HDD and large memory, it is more suitable for future hardware trends and easier to achieve balanced reading and writing.

#### Easy data modeling and high flexibility

You can easily model the connected data into NebulaGraph for your business without forcing them into a structure such as a relational table, and properties can be added, updated, and deleted freely. For more information, see Data modeling.

#### **High popularity**

NebulaGraph is being used by tech leaders such as Tencent, Vivo, Meituan, and JD Digits. For more information, visit the NebulaGraph official website.

#### 2.1.3 Use cases

NebulaGraph can be used to support various graph-based scenarios. To spare the time spent on pushing the kinds of data mentioned in this section into relational databases and on bothering with join queries, use NebulaGraph.

#### Fraud detection

Financial institutions have to traverse countless transactions to piece together potential crimes and understand how combinations of transactions and devices might be related to a single fraud scheme. This kind of scenario can be modeled in graphs, and with the help of NebulaGraph, fraud rings and other sophisticated scams can be easily detected.

#### **Real-time recommendation**

NebulaGraph offers the ability to instantly process the real-time information produced by a visitor and make accurate recommendations on articles, videos, products, and services.

#### Intelligent question-answer system

Natural languages can be transformed into knowledge graphs and stored in NebulaGraph. A question organized in a natural language can be resolved by a semantic parser in an intelligent question-answer system and re-organized. Then, possible answers to the question can be retrieved from the knowledge graph and provided to the one who asked the question.

#### Social networking

Information on people and their relationships is typical graph data. NebulaGraph can easily handle the social networking information of billions of people and trillions of relationships, and provide lightning-fast queries for friend recommendations and job promotions in the case of massive concurrency.

#### 2.1.4 Related links

- Official website
- Docs
- Blogs
- Forum
- GitHub

Last update: October 25, 2023

# 2.2 Data modeling

A data model is a model that organizes data and specifies how they are related to one another. This topic describes the Nebula Graph data model and provides suggestions for data modeling with NebulaGraph.

#### 2.2.1 Data structures

NebulaGraph data model uses six data structures to store data. They are graph spaces, vertices, edges, tags, edge types and properties.

- **Graph spaces**: Graph spaces are used to isolate data from different teams or programs. Data stored in different graph spaces are securely isolated. Storage replications, privileges, and partitions can be assigned.
- Vertices: Vertices are used to store entities.
- In NebulaGraph, vertices are identified with vertex identifiers (i.e. VID). The VID must be unique in the same graph space. VID should be int64, or fixed string(N).
- A vertex has zero to multiple tags.

# 

In NebulaGraph 2.x a vertex must have at least one tag. And in NebulaGraph 3.6.0, a tag is not required for a vertex.

- Edges: Edges are used to connect vertices. An edge is a connection or behavior between two vertices.
- There can be multiple edges between two vertices.
- Edges are directed. -> identifies the directions of edges. Edges can be traversed in either direction.
- An edge is identified uniquely with <a source vertex, an edge type, a rank value, and a destination vertex>. Edges have no EID.
- An edge must have one and only one edge type.
- The rank value is an immutable user-assigned 64-bit signed integer. It identifies the edges with the same edge type between two vertices. Edges are sorted by their rank values. The edge with the greatest rank value is listed first. The default rank value is zero.
- Tags: Tags are used to categorize vertices. Vertices that have the same tag share the same definition of properties.
- Edge types: Edge types are used to categorize edges. Edges that have the same edge type share the same definition of properties.
- Properties: Properties are key-value pairs. Both vertices and edges are containers for properties.

## Note

Tags and Edge types are similar to "vertex tables" and "edge tables" in the relational databases.

## 2.2.2 Directed property graph

NebulaGraph stores data in directed property graphs. A directed property graph has a set of vertices connected by directed edges. Both vertices and edges can have properties. A directed property graph is represented as:

# $G = \langle V, E, P_V, P_E \rangle$

- V is a set of vertices.
- E is a set of directed edges.
- $\boldsymbol{P}_{\boldsymbol{V}}$  is the property of vertices.
- $\mathbf{P}_{\mathbf{E}}$  is the property of edges.

The following table is an example of the structure of the basketball player dataset. We have two types of vertices, that is **player** and **team**, and two types of edges, that is **serve** and **follow**.

Element	Name	Property name (Data type)	Description
Tag	player	name (string) age (int)	Represents players in the team.
Tag	team	name (string)	Represents the teams.
Edge type	serve	start_year (int) end_year (int)	Represents actions taken by players in the team. An action links a player with a team, and the direction is from a player to a team.
Edge type	follow	degree (int)	Represents actions taken by players in the team. An action links a player with another player, and the direction is from one player to the other player.

# Note

NebulaGraph supports only directed edges.

# *H*mpatibility

NebulaGraph 3.6.0 allows dangling edges. Therefore, when adding or deleting, you need to ensure the corresponding source vertex and destination vertex of an edge exist. For details, see INSERT VERTEX, DELETE VERTEX, INSERT EDGE, and DELETE EDGE.

The MERGE statement in openCypher is not supported.

```
Last update: November 3, 2023
```

# 2.3 Path types

In graph theory, a path in a graph is a finite or infinite sequence of edges which joins a sequence of vertices. Paths are fundamental concepts of graph theory.

Paths can be categorized into 3 types: walk, trail, and path. For more information, see Wikipedia.

The following figure is an example for a brief introduction.



#### 2.3.1 Walk

A walk is a finite or infinite sequence of edges. Both vertices and edges can be repeatedly visited in graph traversal.

## Note

GO statements use walk.

#### 2.3.2 Trail

A trait is a finite sequence of edges. Only vertices can be repeatedly visited in graph traversal. The Seven Bridges of Königsberg is a typical trait.

In the above figure, edges cannot be repeatedly visited. So, this figure contains finite paths. The longest path in this figure consists of 5 edges: A - B - C - D - E - C.

# Note

MATCH, FIND PATH, and GET SUBGRAPH statements use trail.

There are two special cases of trail, cycle and circuit. The following figure is an example for a brief introduction.



• cycle

A cycle refers to a closed trail. Only the terminal vertices can be repeatedly visited. The longest path in this figure consists of 3 edges: A - B - C - A or C - D - E - C.

#### • circuit

A circuit refers to a closed trait. Edges cannot be repeatedly visited in graph traversal. Apart from the terminal vertices, other vertices can also be repeatedly visited. The longest path in this figure: A - B - C - D - E - C - A.

#### 2.3.3 Path

A path is a finite sequence of edges. Neither vertices nor edges can be repeatedly visited in graph traversal.

So, the above figure contains finite paths. The longest path in this figure consists of 4 edges: A - B - C - D - E.

Last update: October 25, 2023

# 2.4 VID

In a graph space, a vertex is uniquely identified by its ID, which is called a VID or a Vertex ID.

#### 2.4.1 Features

- The data types of VIDs are restricted to  $FIXED_STRING(<N>)$  or INT64. One graph space can only select one VID type.
- A VID in a graph space is unique. It functions just as a primary key in a relational database. VIDs in different graph spaces are independent.
- The VID generation method must be set by users, because NebulaGraph does not provide auto increasing ID, or UUID.
- Vertices with the same VID will be identified as the same one. For example:
- A VID is the unique identifier of an entity, like a person's ID card number. A tag means the type of an entity, such as driver, and boss. Different tags define two groups of different properties, such as driving license number, driving age, order amount, order taking alt, and job number, payroll, debt ceiling, business phone number.
- When two INSERT statements (neither uses a parameter of IF NOT EXISTS) with the same VID and tag are operated at the same time, the latter INSERT will overwrite the former.
- When two INSERT statements with the same VID but different tags, like TAG A and TAG B, are operated at the same time, the operation of Tag A will not affect Tag B.
- VIDs will usually be indexed and stored into memory (in the way of LSM-tree). Thus, direct access to VIDs enjoys peak performance.

#### 2.4.2 VID Operation

- NebulaGraph 1.x only supports INT64 while NebulaGraph 2.x supports INT64 and FIXED\_STRING(<N>). In CREATE SPACE, VID types can be set via vid\_type.
- id() function can be used to specify or locate a VID.
- LOOKUP or MATCH statements can be used to find a VID via property index.
- Direct access to vertices statements via VIDs enjoys peak performance, such as DELETE xxx WHERE id(xxx) = "player100" or GO FROM "player100". Finding VIDs via properties and then operating the graph will cause poor performance, such as LOOKUP | GO FROM \$-.ids, which will run both LOOKUP and | one more time.

#### 2.4.3 VID Generation

VIDs can be generated via applications. Here are some tips:

- (Optimal) Directly take a unique primary key or property as a VID. Property access depends on the VID.
- Generate a VID via a unique combination of properties. Property access depends on property index.
- Generate a VID via algorithms like snowflake. Property access depends on property index.
- If short primary keys greatly outnumber long primary keys, do not enlarge the N of FIXED\_STRING(<N>) too much. Otherwise, it will occupy a lot of memory and hard disks, and slow down performance. Generate VIDs via BASE64, MD5, hash by encoding and splicing.
- If you generate int64 VID via hash, the probability of collision is about 1/10 when there are 1 billion vertices. The number of edges has no concern with the probability of collision.

#### 2.4.4 Define and modify a VID and its data type

The data type of a VID must be defined when you create the graph space. Once defined, it cannot be modified.

A VID is set when you insert a vertex and cannot be modified.

#### 2.4.5 Query start vid and global scan

In most cases, the execution plan of query statements in NebulaGraph (  $\tt MATCH$  ,  $\tt GO$  , and  $\tt LOOKUP$  ) must query the start vid in a certain way.

There are only two ways to locate start vid :

- 1. For example, 60 FROM "player100" OVER explicitly indicates in the statement that start vid is "player100".
- 2. For example, LOOKUP ON player WHERE player.name == "Tony Parker" or MATCH (v:player {name:"Tony Parker"}) locates start vid by the index of the property player.name.

Last update: October 25, 2023

# 2.5 NebulaGraph architecture

#### 2.5.1 Architecture overview

NebulaGraph consists of three services: the Graph Service, the Storage Service, and the Meta Service. It applies the separation of storage and computing architecture.

Each service has its executable binaries and processes launched from the binaries. Users can deploy a NebulaGraph cluster on a single machine or multiple machines using these binaries.

The following figure shows the architecture of a typical NebulaGraph cluster.



#### The Meta Service

The Meta Service in the NebulaGraph architecture is run by the nebula-metad processes. It is responsible for metadata management, such as schema operations, cluster administration, and user privilege management.

For details on the Meta Service, see Meta Service.

#### The Graph Service and the Storage Service

NebulaGraph applies the separation of storage and computing architecture. The Graph Service is responsible for querying. The Storage Service is responsible for storage. They are run by different processes, i.e., nebula-graphd and nebula-storaged. The benefits of the separation of storage and computing architecture are as follows:

• Great scalability

The separated structure makes both the Graph Service and the Storage Service flexible and easy to scale in or out.

• High availability

If part of the Graph Service fails, the data stored by the Storage Service suffers no loss. And if the rest part of the Graph Service is still able to serve the clients, service recovery can be performed quickly, even unfelt by the users.

• Cost-effective

The separation of storage and computing architecture provides a higher resource utilization rate, and it enables clients to manage the cost flexibly according to business demands.

• Open to more possibilities

With the ability to run separately, the Graph Service may work with multiple types of storage engines, and the Storage Service may also serve more types of computing engines.

For details on the Graph Service and the Storage Service, see Graph Service and Storage Service.

```
Last update: October 25, 2023
```

#### 2.5.2 Meta Service

This topic introduces the architecture and functions of the Meta Service.

#### The architecture of the Meta Service

The architecture of the Meta Service is as follows:



The Meta Service is run by nebula-metad processes. Users can deploy nebula-metad processes according to the scenario:

- In a test environment, users can deploy one or three nebula-metad processes on different machines or a single machine.
- In a production environment, we recommend that users deploy three nebula-metad processes on different machines for high availability.

All the nebula-metad processes form a Raft-based cluster, with one process as the leader and the others as the followers.

The leader is elected by the majorities and only the leader can provide service to the clients or other components of NebulaGraph. The followers will be run in a standby way and each has a data replication of the leader. Once the leader fails, one of the followers will be elected as the new leader.

#### Q Note

The data of the leader and the followers will keep consistent through Raft. Thus the breakdown and election of the leader will not cause data inconsistency. For more information on Raft, see Storage service architecture.

#### Functions of the Meta Service

#### MANAGES USER ACCOUNTS

The Meta Service stores the information of user accounts and the privileges granted to the accounts. When the clients send queries to the Meta Service through an account, the Meta Service checks the account information and whether the account has the right privileges to execute the queries or not.

For more information on NebulaGraph access control, see Authentication.

#### MANAGES PARTITIONS

The Meta Service stores and manages the locations of the storage partitions and helps balance the partitions.

#### MANAGES GRAPH SPACES

NebulaGraph supports multiple graph spaces. Data stored in different graph spaces are securely isolated. The Meta Service stores the metadata of all graph spaces and tracks the changes of them, such as adding or dropping a graph space.

#### MANAGES SCHEMA INFORMATION

NebulaGraph is a strong-typed graph database. Its schema contains tags (i.e., the vertex types), edge types, tag properties, and edge type properties.

The Meta Service stores the schema information. Besides, it performs the addition, modification, and deletion of the schema, and logs the versions of them.

For more information on NebulaGraph schema, see Data model.

#### MANAGES TTL INFORMATION

The Meta Service stores the definition of TTL (Time to Live) options which are used to control data expiration. The Storage Service takes care of the expiring and evicting processes. For more information, see TTL.

#### MANAGES JOBS

The Job Management module in the Meta Service is responsible for the creation, queuing, querying, and deletion of jobs.

Last update: November 3, 2023

#### 2.5.3 Graph Service

The Graph Service is used to process the query. It has four submodules: Parser, Validator, Planner, and Executor. This topic will describe the Graph Service accordingly.

#### The architecture of the Graph Service



After a query is sent to the Graph Service, it will be processed by the following four submodules:

- 1. Parser: Performs lexical analysis and syntax analysis.
- 2. Validator: Validates the statements.
- 3. **Planner**: Generates and optimizes the execution plans.
- 4. Executor: Executes the plans with operators.

#### Parser

After receiving a request, the statements will be parsed by Parser composed of Flex (lexical analysis tool) and Bison (syntax analysis tool), and its corresponding AST will be generated. Statements will be directly intercepted in this stage because of their invalid syntax.

For example, the structure of the AST of GO FROM "Tim" OVER Like WHERE properties(edge).Likeness > 8.0 YIELD dst(edge) is shown in the following figure.



#### Validator

Validator performs a series of validations on the AST. It mainly works on these tasks:

• Validating metadata

Validator will validate whether the metadata is correct or not.

When parsing the OVER, where, and VIELD clauses, Validator looks up the Schema and verifies whether the edge type and tag data exist or not. For an INSERT statement, Validator verifies whether the types of the inserted data are the same as the ones defined in the Schema.

• Validating contextual reference

Validator will verify whether the cited variable exists or not, or whether the cited property is variable or not.

For composite statements, like \$var = GO FROM "Tim" OVER Like YIELD dst(edge) AS ID; GO FROM \$var.ID OVER serve YIELD dst(edge), Validator verifies first to see if var is defined, and then to check if the ID property is attached to the var variable.

• Validating type inference

Validator infers what type the result of an expression is and verifies the type against the specified clause.

For example, the where clause requires the result to be a bool value, a NULL value, or  ${\tt empty}$  .

• Validating the information of \*

Validator needs to verify all the Schema that involves \* when verifying the clause if there is a \* in the statement.

Take a statement like GO FROM "Tim" OVER \* YIELD dst(edge), properties(edge).likeness, dst(edge) as an example. When verifying the OVER clause, Validator needs to verify all the edge types. If the edge type includes like and serve, the statement would be GO FROM "Tim" OVER like, serve YIELD dst(edge), properties(edge).likeness, dst(edge).

• Validating input and output

Validator will check the consistency of the clauses before and after the **II**.

In the statement GO FROM "Tim" OVER Like YIELD dst(edge) AS ID | GO FROM \$-.ID OVER serve YIELD dst(edge), Validator will verify whether \$-.ID is defined in the clause before the ||.

When the validation succeeds, an execution plan will be generated. Its data structure will be stored in the src/planner directory.

#### Planner

In the nebula-graphd.conf file, when enable\_optimizer is set to be false, Planner will not optimize the execution plans generated by Validator. It will be executed by Executor directly.

In the nebula-graphd.conf file, when enable\_optimizer is set to be true, Planner will optimize the execution plans generated by Validator. The structure is as follows.



#### • Before optimization

In the execution plan on the right side of the preceding figure, each node directly depends on other nodes. For example, the root node Project depends on the Filter node, the Filter node depends on the GetNeighbor node, and so on, up to the leaf node Start. Then the execution plan is (not truly) executed.

During this stage, every node has its input and output variables, which are stored in a hash table. The execution plan is not truly executed, so the value of each key in the associated hash table is empty (except for the Start node, where the input variables hold the starting data), and the hash table is defined in src/context/ExecutionContext.cpp under the nebula-graph repository.

For example, if the hash table is named as ResultMap when creating the Filter node, users can determine that the node takes data from ResultMap["GN1"], then puts the result into ResultMap["Filter2"], and so on. All these work as the input and output of each node.

#### • Process of optimization

The optimization rules that Planner has implemented so far are considered RBO (Rule-Based Optimization), namely the predefined optimization rules. The CBO (Cost-Based Optimization) feature is under development. The optimized code is in the src/ optimizer/ directory under the nebula-graph repository.

RBO is a "bottom-up" exploration process. For each rule, the root node of the execution plan (in this case, the Project node) is the entry point, and step by step along with the node dependencies, it reaches the node at the bottom to see if it matches the rule.

As shown in the preceding figure, when the Filter node is explored, it is found that its children node is GetNeighbors, which matches successfully with the pre-defined rules, so a transformation is initiated to integrate the Filter node into the GetNeighbors node, the Filter node is removed, and then the process continues to the next rule. Therefore, when the GetNeighbor operator calls interfaces of the Storage layer to get the neighboring edges of a vertex during the execution stage, the Storage layer will directly filter out the unqualified edges internally. Such optimization greatly reduces the amount of data transfer, which is commonly known as filter pushdown.

#### Executor

The Executor module consists of Scheduler and Executor. The Scheduler generates the corresponding execution operators against the execution plan, starting from the leaf nodes and ending at the root node. The structure is as follows.



Each node of the execution plan has one execution operator node, whose input and output have been determined in the execution plan. Each operator only needs to get the values for the input variables, compute them, and finally put the results into the corresponding output variables. Therefore, it is only necessary to execute step by step from Start, and the result of the last operator is returned to the user as the final result.

#### Source code hierarchy

The source code hierarchy under the nebula-graph repository is as follows.

Src	
graph	
context	//contexts for validation and execution
executor	//execution operators
gc	//garbage collector
optimizer	//optimization rules
planner	<pre>//structure of the execution plans</pre>
scheduler	//scheduler
service	<pre>//external service management</pre>
session	//session management
stats	//monitoring metrics
util	//basic components
validator	<pre>//validation of the statements</pre>
visitor	//visitor expression

Last update: October 25, 2023

#### 2.5.4 Storage Service

The persistent data of NebulaGraph have two parts. One is the Meta Service that stores the meta-related data.

The other is the Storage Service that stores the data, which is run by the nebula-storaged process. This topic will describe the architecture of the Storage Service.

#### Advantages

- High performance (Customized built-in KVStore)
- Great scalability (Shared-nothing architecture, not rely on NAS/SAN-like devices)
- Strong consistency (Raft)
- High availability (Raft)
- Supports synchronizing with the third party systems, such as Elasticsearch.

#### The architecture of the Storage Service



The Storage Service is run by the nebula-storaged process. Users can deploy nebula-storaged processes on different occasions. For example, users can deploy 1 nebula-storaged process in a test environment and deploy 3 nebula-storaged processes in a production environment.

All the nebula-storaged processes consist of a Raft-based cluster. There are three layers in the Storage Service:

• Storage interface

The top layer is the storage interface. It defines a set of APIs that are related to the graph concepts. These API requests will be translated into a set of KV operations targeting the corresponding Partition. For example:

- getNeighbors : queries the in-edge or out-edge of a set of vertices, returns the edges and the corresponding properties, and supports conditional filtering.
- insert vertex/edge : inserts a vertex or edge and its properties.
- getProps : gets the properties of a vertex or an edge.

It is this layer that makes the Storage Service a real graph storage. Otherwise, it is just a KV storage.

• Consensus

Below the storage interface is the consensus layer that implements Multi Group Raft, which ensures the strong consistency and high availability of the Storage Service.

• Store engine

The bottom layer is the local storage engine library, providing operations like get, put, and scan on local disks. The related interfaces are stored in KVStore.h and KVEngine.h files. You can develop your own local store plugins based on your needs.

The following will describe some features of the Storage Service based on the above architecture.

#### Storage writing process



#### KVStore

NebulaGraph develops and customizes its built-in KVStore for the following reasons.

- It is a high-performance KVStore.
- It is provided as a (kv) library and can be easily developed for the filter pushdown purpose. As a strong-typed database, how to provide Schema during pushdown is the key to efficiency for NebulaGraph.
- It has strong data consistency.

Therefore, NebulaGraph develops its own KVStore with RocksDB as the local storage engine. The advantages are as follows.

- For multiple local hard disks, NebulaGraph can make full use of its concurrent capacities through deploying multiple data directories.
- The Meta Service manages all the Storage servers. All the partition distribution data and current machine status can be found in the meta service. Accordingly, users can execute a manual load balancing plan in meta service.

#### Q Note

NebulaGraph does not support auto load balancing because auto data transfer will affect online business.

- NebulaGraph provides its own WAL mode so one can customize the WAL. Each partition owns its WAL.
- One NebulaGraph KVStore cluster supports multiple graph spaces, and each graph space has its own partition number and replica copies. Different graph spaces are isolated physically from each other in the same cluster.

#### Data storage structure

Graphs consist of vertices and edges. NebulaGraph uses key-value pairs to store vertices, edges, and their properties. Vertices and edges are stored in keys and their properties are stored in values. Such structure enables efficient property filtering.

#### • The storage structure of vertices

Different from NebulaGraph version 2.x, version 3.x added a new key for each vertex. Compared to the old key that still exists, the new key has no TagID field and no value. Vertices in NebulaGraph can now live without tags owing to the new key.

Field	Description
Туре	One byte, used to indicate the key type.
PartID	Three bytes, used to indicate the sharding partition and to scan the partition data based on the prefix when re-balancing the partition.
VertexID	The vertex ID. For an integer VertexID, it occupies eight bytes. However, for a string VertexID, it is changed to <code>fixed_string</code> of a fixed length which needs to be specified by users when they create the space.
TagID	Four bytes, used to indicate the tags that vertex relate with.
SerializedValue	The serialized value of the key. It stores the property information of the vertex.

#### • The storage structure of edges

Type (1 byte)	PartID (3 bytes)	VertexID (n bytes)	EdgeType (4 bytes)	Rank (8 bytes)	VertexiD (n bytes)	PlaceHolder (1 byte)	SerializedValue
<b>Field</b>	<b>Descrip</b> One byt	<b>Description</b> One byte, used to indicate the key type.					
PartID	Three b the pref	Three bytes, used to indicate the partition ID. This field can be used to scan the partition data based on the prefix when re-balancing the partition.					
VertexID	Used to VID in t VID in t	Used to indicate vertex ID. The former VID refers to the source VID in the outgoing edge and the dest VID in the incoming edge, while the latter VID refers to the dest VID in the outgoing edge and the source VID in the incoming edge.					
Edge Type	Four by edge.	Four bytes, used to indicate the edge type. Greater than zero indicates out-edge, less than zero means in- edge.					
Rank	Eight by store we	Eight bytes, used to indicate multiple edges in one edge type. Users can set the field based on needs and store weight, such as transaction time and transaction number.					
PlaceHolder	One byt	One byte. Reserved.					
SerializedValue	The serialized value of the key. It stores the property information of the edge.						

PROPERTY DESCRIPTIONS

NebulaGraph uses strong-typed Schema.

NebulaGraph will store the properties of vertex and edges in order after encoding them. Since the length of fixed-length properties is fixed, queries can be made in no time according to offset. Before decoding, NebulaGraph needs to get (and cache) the schema information in the Meta Service. In addition, when encoding properties, NebulaGraph will add the corresponding schema version to support online schema change.

#### Data partitioning

Since in an ultra-large-scale relational network, vertices can be as many as tens to hundreds of billions, and edges are even more than trillions. Even if only vertices and edges are stored, the storage capacity of both exceeds that of ordinary servers. Therefore, NebulaGraph uses hash to shard the graph elements and store them in different partitions.



EDGE PARTITIONING AND STORAGE AMPLIFICATION

In NebulaGraph, an edge corresponds to two key-value pairs on the hard disk. When there are lots of edges and each has many properties, storage amplification will be obvious. The storage format of edges is shown in the figure below.



In this example, SrcVertex connects DstVertex via EdgeA, forming the path of (SrcVertex)-[EdgeA]->(DstVertex). SrcVertex, DstVertex, and EdgeA will all be stored in Partition x and Partition y as four key-value pairs in the storage layer. Details are as follows:

- The key value of SrcVertex is stored in Partition x. Key fields include Type, PartID(x), VID(Src), and TagID. SerializedValue, namely Value, refers to serialized vertex properties.
- The first key value of EdgeA, namely EdgeA\_Out, is stored in the same partition as the SrcVertex. Key fields include Type, PartID(x), VID(Src), EdgeType(+ means out-edge), Rank(0), VID(Dst), and PlaceHolder. SerializedValue, namely Value, refers to serialized edge properties.
- The key value of DstVertex is stored in Partition y. Key fields include Type, PartID(y), VID(Dst), and TagID. SerializedValue, namely Value, refers to serialized vertex properties.
- The second key value of EdgeA, namely EdgeA\_In, is stored in the same partition as the DstVertex. Key fields include Type, PartID(y), VID(Dst), EdgeType(- means in-edge), Rank(0), VID(Src), and PlaceHolder. SerializedValue, namely Value, refers to serialized edge properties, which is exactly the same as that in EdgeA\_Out.

EdgeA\_Out and EdgeA\_In are stored in storage layer with opposite directions, constituting EdgeA logically. EdgeA\_Out is used for traversal requests starting from SrcVertex, such as (a)-[]->(); EdgeA\_In is used for traversal requests starting from DstVertex, such as ()-[]->(a).

Like EdgeA\_Out and EdgeA\_In, NebulaGraph redundantly stores the information of each edge, which doubles the actual capacities needed for edge storage. The key corresponding to the edge occupies a small hard disk space, but the space occupied by Value is proportional to the length and amount of the property value. Therefore, it will occupy a relatively large hard disk space if the property value of the edge is large or there are many edge property values.

PARTITION ALGORITHM

NebulaGraph uses a **static Hash** strategy to shard data through a modulo operation on vertex ID. All the out-keys, in-keys, and tag data will be placed in the same partition. In this way, query efficiency is increased dramatically.

#### O Note

The number of partitions needs to be determined when users are creating a graph space since it cannot be changed afterward. Users are supposed to take into consideration the demands of future business when setting it.

When inserting into NebulaGraph, vertices and edges are distributed across different partitions. And the partitions are located on different machines. The number of partitions is set in the CREATE SPACE statement and cannot be changed afterward.

If certain vertices need to be placed on the same partition (i.e., on the same machine), see Formula/code.

The following code will briefly describe the relationship between VID and partition.

```
// If VertexID occupies 8 bytes, it will be stored in int64 to be compatible with the version 1.0.
uint64_t vid = 0;
if (id.size() == 8) {
    memcpy(static_cast<void*>(&vid), id.data(), 8);
} else {
    MurmurHash2 hash;
    vid = hash(id.data());
}
PartitionID pId = vid % numParts + 1;
```

Roughly speaking, after hashing a fixed string to int64, (the hashing of int64 is the number itself), do modulo, and then plus one, namely:

pId = vid % numParts + 1;

Parameters and descriptions of the preceding formula are as follows:

Parameter	Description
%	The modulo operation.
numParts	The number of partitions for the graph space where the VID is located, namely the value of partition_num in the CREATE SPACE statement.
pId	The ID for the partition where the VID is located.

Suppose there are 100 partitions, the vertices with VID 1, 101, and 1001 will be stored on the same partition. But, the mapping between the partition ID and the machine address is random. Therefore, we cannot assume that any two partitions are located on the same machine.

#### Raft

#### RAFT IMPLEMENTATION

In a distributed system, one data usually has multiple replicas so that the system can still run normally even if a few copies fail. It requires certain technical means to ensure consistency between replicas.

Basic principle: Raft is designed to ensure consistency between replicas. Raft uses election between replicas, and the (candidate) replica that wins more than half of the votes will become the Leader, providing external services on behalf of all replicas. The rest Followers will play backups. When the Leader fails (due to communication failure, operation and maintenance commands, etc.), the rest Followers will conduct a new round of elections and vote for a new Leader. The Leader and Followers will detect each other's survival through heartbeats and write them to the hard disk in Raft-wal mode. Replicas that do not respond to more than multiple heartbeats will be considered faulty.

# Note

Raft-wal needs to be written into the hard disk periodically. If hard disk bottlenecks to write, Raft will fail to send a heartbeat and conduct a new round of elections. If the hard disk IO is severely blocked, there will be no Leader for a long time.

Read and write: For every writing request of the clients, the Leader will initiate a Raft-wal and synchronize it with the Followers. Only after over half replicas have received the Raft-wal will it return to the clients successfully. For every reading request of the clients, it will get to the Leader directly, while Followers will not be involved.

Failure: Scenario 1: Take a (space) cluster of a single replica as an example. If the system has only one replica, the Leader will be itself. If failure happens, the system will be completely unavailable. Scenario 2: Take a (space) cluster of three replicas as an example. If the system has three replicas, one of them will be the Leader and the rest will be the Followers. If the Leader fails, the rest two can still vote for a new Leader (and a Follower), and the system is still available. But if any of the two Followers fails again, the system will be completely unavailable due to inadequate voters.

#### O Note

Raft and HDFS have different modes of duplication. Raft is based on a quorum vote, so the number of replicas cannot be even.

#### MULTI GROUP RAFT

The Storage Service supports a distributed cluster architecture, so NebulaGraph implements Multi Group Raft according to Raft protocol. Each Raft group stores all the replicas of each partition. One replica is the leader, while others are followers. In this way, NebulaGraph achieves strong consistency and high availability. The functions of Raft are as follows.

NebulaGraph uses Multi Group Raft to improve performance when there are many partitions because Raft-wal cannot be NULL. When there are too many partitions, costs will increase, such as storing information in Raft group, WAL files, or batch operation in low load. There are two key points to implement the Multi Raft Group:

• To share transport layer

Each Raft Group sends messages to its corresponding peers. So if the transport layer cannot be shared, the connection costs will be very high.

• To share thread pool

Raft Groups share the same thread pool to prevent starting too many threads and a high context switch cost.

BATCH

For each partition, it is necessary to do a batch to improve throughput when writing the WAL serially. As NebulaGraph uses WAL to implement some special functions, batches need to be grouped, which is a feature of NebulaGraph.

For example, lock-free CAS operations will execute after all the previous WALs are committed. So for a batch, if there are several WALs in CAS type, we need to divide this batch into several smaller groups and make sure they are committed serially.

#### TRANSFER LEADERSHIP

Transfer leadership is extremely important for balance. When moving a partition from one machine to another, NebulaGraph first checks if the source is a leader. If so, it should be moved to another peer. After data migration is completed, it is important to balance leader distribution again.

When a transfer leadership command is committed, the leader will abandon its leadership and the followers will start a leader election.

#### PEER CHANGES

To avoid split-brain, when members in a Raft Group change, an intermediate state is required. In such a state, the quorum of the old group and new group always have an overlap. Thus it prevents the old or new group from making decisions unilaterally. To make it even simpler, in his doctoral thesis Diego Ongaro suggests adding or removing a peer once to ensure the overlap between the quorum of the new group and the old group. NebulaGraph also uses this approach, except that the way to add or remove a member is different. For details, please refer to addPeer/removePeer in the Raft Part class.

#### **Differences with HDFS**

The Storage Service is a Raft-based distributed architecture, which has certain differences with that of HDFS. For example:

- The Storage Service ensures consistency through Raft. Usually, the number of its replicas is odd to elect a leader. However, DataNode used by HDFS ensures consistency through NameNode, which has no limit on the number of replicas.
- In the Storage Service, only the replicas of the leader can read and write, while in HDFS all the replicas can do so.
- In the Storage Service, the number of replicas needs to be determined when creating a space, since it cannot be changed afterward. But in HDFS, the number of replicas can be changed freely.
- The Storage Service can access the file system directly. While the applications of HDFS (such as HBase) have to access HDFS before the file system, which requires more RPC times.

In a word, the Storage Service is more lightweight with some functions simplified and its architecture is simpler than HDFS, which can effectively improve the read and write performance of a smaller block of data.

Last update: November 3, 2023

2.5.4 Storage Service
# 3. Quick start

# 3.1 Quickly deploy NebulaGraph using Docker

 $\label{eq:cond} \ensuremath{\mathsf{You}}\ \ensuremath{\mathsf{can}}\ \ensuremath{\mathsf{vith}}\ \ensuremath{\mathsf{Docker}}\ \ensuremath\ensurema$ 

Using Docker Desktop Using Docker Compose

NebulaGraph is available as a Docker Extension that you can easily install and run on your Docker Desktop. You can quickly deploy NebulaGraph using Docker Desktop with just one click.

1. Install Docker Desktop.

## Caution

We do not recommend you deploy NebulaGraph on Docker Desktop for Windows due to its subpar performance. For details, see #12401. If you must use Docker Desktop for Windows, install WSL 2 first.

- 2. In the left sidebar of Docker Desktop, click Extensions or Add Extensions.
- 3. On the Extensions Marketplace, search for NebulaGraph and click Install.

🔵 🔵 🌑 Docker Desktop 🛛 Upgra		Q. Search	жк	¥ \$	Sign in 😫
Containers Limages Volumes	Q nebula Containers (15)	Images (61) Extensions (1)	× Le	am more	:
Dev Environments BETA	WebulaGraph	Easily deploy and test NebulaGraph, the Open-Source Distributed Graph Databa	se. 👱 200	Actions	
Extensions			FU.	Actions	
Add Extensions			_		Î
0				▶ =	ĩ
	NebulaG Vesoft Inc	aph Updated 3 d	Install ays ago • v0.4.4		
	About				
	Easily deploy and t	st NebulaGraph, the Open-Source Distributed Graph Database.			
	NebulaGraph is a p	opular Open-Source, distributed Cloud Native Graph Database for trillion edges grap	oh data volume.		
	Use ↑ ↓ or up and do	n arrow keys to navigate between results Next tip	Give feedback 🖳		
				Showing	2 items
<b>*</b>	RAM 0.44 GB CPU 31.6	% Disk 51.89 GB avail. of 58.37 GB 🛛 💥 Not connected to Hub			v4.16.2 Q*

Click Update to update NebulaGraph to the latest version when a new version is available.

Docker Desktop Upgrade plan	Q Search	Ctrl+K 🛛 😆 🌣 Sign in 😌 —	
Containers  Images	Extensions Market	Diace <u>Give feedback</u> and a strain of the st	
Volumes Volumes Dev Environments BETA	Browse Manage	Build NEW Request an ext	ension 🛛
	All	Q Search Recently add	ded 👻
Extensions :	CI/CD		
Add Extensions	Cloud Deployment	Vesofi Inc.	all
0	Cloud Development	Database.	
	Container Orchestration	Gefyra Blueshoe GmbH	<u>♣</u> 300
	Database	Gefyra's Docker extension to bridge running containers into Kubernetes clusters.	1
	Kubernetes		
	Networking	Kubescape- 39/805 -	2023 Vesoft Ir
	Image Registry	Secure your Kubernetes cluster and gain insight into your cluster's security posture via an Instal	1

Last update: March 26, 2024

## 3.2 Deploy NebulaGraph on-premise

## 3.2.1 Step 1: Install NebulaGraph

RPM and DEB are common package formats on Linux systems. This topic shows how to quickly install NebulaGraph with the RPM or DEB package.

# Note

The console is not complied or packaged with NebulaGraph server binaries. You can install nebula-console by yourself.

## Prerequisites

• The tool wget is installed.

#### Step 1: Download the package from cloud service

# Note NebulaGraph is currently only supported for installation on Linux systems, and only CentOS 7.x, CentOS 8.x, Ubuntu 16.04, Ubuntu 18.04, and Ubuntu 20.04 operating systems are supported.

#### • Download the released version.

#### URL:

//Centos 7
https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.el7.x86\_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.el8.x86\_64.rpm

#### //Ubuntu 1604

https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1604.amd64.deb

#### //Ubuntu 1804

https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1804.amd64.deb

#### //Ubuntu 2004

https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.ubuntu2004.amd64.deb

For example, download the release package 3.6.0 for Centos 7.5:

wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.el7.x86\_64.rpm
wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.el7.x86\_64.rpm.sha256sum.txt

#### Download the release package 3.6.0 for Ubuntu 1804:

wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.ubuntu1804.amd64.deb wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.ubuntu1804.amd64.deb.sha256sum.txt

#### · Download the nightly version.

## Danger

- Nightly versions are usually used to test new features. Do not use it in a production environment.
- Nightly versions may not be built successfully every night. And the names may change from day to day.

#### URL:



#### For example, download the Centos 7.5 package developed and built in 2021.11.28:

wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.el7.x86\_64.rpm
wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.el7.x86\_64.rpm.sha256sum.txt

#### For example, download the Ubuntu 1804 package developed and built in 2021.11.28:

wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.ubuntu1804.amd64.deb wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.ubuntu1804.amd64.deb.sha256sum.txt

## Step 2: Install NebulaGraph

• Use the following syntax to install with an RPM package.

\$ sudo rpm -ivh --prefix=<installation\_path> <package\_name>

The option --prefix indicates the installation path. The default path is /usr/local/nebula/.

For example, to install an RPM package in the default path for the 3.6.0 version, run the following command.

sudo rpm -ivh nebula-graph-3.6.0.el7.x86\_64.rpm

• Use the following syntax to install with a DEB package.

\$ sudo dpkg -i <package\_name>

## Note

 $Customizing \ the \ installation \ path \ is \ not \ supported \ when \ installing \ NebulaGraph \ with \ a \ DEB \ package. \ The \ default \ installation \ path \ is \ /usr/local/nebula/ \ .$ 

For example, to install a DEB package for the 3.6.0 version, run the following command.

sudo dpkg -i nebula-graph-3.6.0.ubuntu1804.amd64.deb

Q Note

The default installation path is  $\mbox{/usr/local/nebula/}$  .

#### Next to do

- Start NebulaGraph
- Connect to NebulaGraph

Last update: October 25, 2023

#### 3.2.2 Step 2: Manage NebulaGraph Service

NebulaGraph supports managing services with scripts.

#### Manage services with script

You can use the nebula.service script to start, stop, restart, terminate, and check the NebulaGraph services.

Note	
nebula.service is stor environment.	red in the /usr/local/nebula/scripts directory by default. If you have customized the path, use the actual path in your
NTAX	
<pre>\$ sudo /usr/local/nebula/scr [-v] [-c <config_file_path>] <start ki<br="" restart="" stop=""  =""><metad graphd="" pre="" storaged=""  =""  <=""></metad></start></config_file_path></pre>	ripts/nebula.service 
Parameter	Description
- V	Display detailed debugging information.
- C	Specify the configuration file path. The default path is /usr/local/nebula/etc/.
start	Start the target services.
stop	Stop the target services.
restart	Restart the target services.
kill	Terminate the target services.
status	Check the status of the target services.
metad	Set the Meta Service as the target service.
graphd	Set the Graph Service as the target service.
storaged	Set the Storage Service as the target service.
all	Set all the NebulaGraph services as the target services.

#### Start NebulaGraph

Run the following command to start NebulaGraph.

\$ sudo /usr/local/nebula/scripts/nebula.service start all [INF0] Starting nebula-metad... [INF0] Done [INF0] Starting nebula-graphd... [INF0] Done [INF0] Starting nebula-storaged... [INF0] Done

#### Stop NebulaGraph

Banger

Do not run kill -9 to forcibly terminate the processes. Otherwise, there is a low probability of data loss.

Run the following command to stop NebulaGraph.

```
$ sudo /usr/local/nebula/scripts/nebula.service stop all
[INF0] Stopping nebula-metad...
[INF0] Done
[INF0] Done
[INF0] Stopping nebula-graphd...
[INF0] Stopping nebula-storaged...
[INF0] Done
```

#### Check the service status

Run the following command to check the service status of NebulaGraph.

\$ sudo /usr/local/nebula/scripts/nebula.service status all

• NebulaGraph is running normally if the following information is returned.

```
INFO] nebula-metad(33fd35e): Running as 29020, Listening on 9559
[INFO] nebula-graphd(33fd35e): Running as 29095, Listening on 9669
[WARN] nebula-storaged after v3.0.0 will not start service until it is added to cluster.
[WARN] See Manage Storage hosts:ADD HOSTS in https://docs.nebula-graph.io/
[INFO] nebula-storaged(33fd35e): Running as 29147, Listening on 9779
```

Note

After starting NebulaGraph, the port of the nebula-storaged process is shown in red. Because the nebula-storaged process waits for the nebula-metad to add the current Storage service during the startup process. The Storage works after it receives the ready signal. Starting from NebulaGraph 3.0.0, the Meta service cannot directly read or write data in the Storage service that you add in the configuration file. The configuration file only registers the Storage service to the Meta service. You must run the ADD HOSTS command to enable the Meta to read and write data in the Storage service. For more information, see Manage Storage hosts.

• If the returned result is similar to the following one, there is a problem. You may also go to the NebulaGraph community for help.

[INF0] nebula-metad: Running as 25600, Listening on 9559 [INF0] nebula-graphd: Exited [INF0] nebula-storaged: Running as 25646, Listening on 9779

The NebulaGraph services consist of the Meta Service, Graph Service, and Storage Service. The configuration files for all three services are stored in the /usr/local/nebula/etc/ directory by default. You can check the configuration files according to the returned result to troubleshoot problems.

#### Next to do

Connect to NebulaGraph

Last update: October 25, 2023

#### 3.2.3 Step 3: Connect to NebulaGraph

This topic provides basic instruction on how to use the native CLI client NebulaGraph Console to connect to NebulaGraph.

## Caution

When connecting to NebulaGraph for the first time, you must register the Storage Service before querying data.

NebulaGraph supports multiple types of clients, including a CLI client, a GUI client, and clients developed in popular programming languages. For more information, see the client list.

#### Prerequisites

- You have started NebulaGraph services.
- The machine on which you plan to run NebulaGraph Console has network access to the Graph Service of NebulaGraph.
- The NebulaGraph Console version is compatible with the NebulaGraph version.

#### Q Note

NebulaGraph Console and NebulaGraph of the same version number are the most compatible. There may be compatibility issues when connecting to NebulaGraph with a different version of NebulaGraph Console. The error message incompatible version between client and server is displayed when there is such an issue.

STEPS

1. On the NebulaGraph Console releases page, select a NebulaGraph Console version and click Assets.

Note	]
t is recommended to select the <b>latest</b> version.	

- 2. In the **Assets** area, find the correct binary file for the machine where you want to run NebulaGraph Console and download the file to the machine.
- 3. (Optional) Rename the binary file to nebula-console for convenience.

## Note

For Windows, rename the file to nebula-console.exe.

4. On the machine to run NebulaGraph Console, grant the execute permission of the nebula-console binary file to the user.

Note
For Windows, skip this step.
chmod 111 nebula-console

5. In the command line interface, change the working directory to the one where the nebula-console binary file is stored.

- $_{6.}$  Run the following command to connect to NebulaGraph.
- For Linux or macOS:

```
$ ./nebula-console -addr <ip> -port <port> -u <username> -p <password>
[-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

#### • For Windows:

> nebula-console.exe -addr <ip> -port <port> -u <username> -p <password>
[-t 120] [-e "nGQL\_statement" | -f filename.nGQL]

#### Parameter descriptions are as follows:

Parameter	Description
-h/-help	Shows the help menu.
-addr/-address	Sets the IP (or hostname) of the Graph service. The default address is 127.0.0.1.
-P/-port	Sets the port number of the graphd service. The default port number is 9669.
-u/-user	Sets the username of your NebulaGraph account. Before enabling authentication, you can use any existing username. The default username is root.
-p/-password	Sets the password of your NebulaGraph account. Before enabling authentication, you can use any characters as the password.
-t/-timeout	Sets an integer-type timeout threshold of the connection. The unit is millisecond. The default value is 120.
-e/-eval	Sets a string-type nGQL statement. The nGQL statement is executed once the connection succeeds. The connection stops after the result is returned.
-f/-file	Sets the path of an nGQL file. The nGQL statements in the file are executed once the connection succeeds. The result will be returned and the connection stops then.
-enable_ssl	Enables SSL encryption when connecting to NebulaGraph.
-ssl_root_ca_path	Sets the storage path of the certification authority file.
-ssl_cert_path	Sets the storage path of the certificate file.
- ssl_private_key_path	Sets the storage path of the private key file.

For information on more parameters, see the project repository.

Last update: October 25, 2023

### 3.2.4 Register the Storage Service

When connecting to NebulaGraph for the first time, you have to add the Storage hosts, and confirm that all the hosts are online.

# Empatibility

• Starting from NebulaGraph 3.0.0, you have to run ADD HOSTS before reading or writing data into the Storage Service.

• For NebulaGraph of versions earlier than 3.0.0 and NebulaGraph Cloud clusters, ADD HOSTS is not needed.

#### Prerequisites

You have connected to NebulaGraph.

#### Steps

#### 1. Add the Storage hosts.

Run the following command to add hosts:

ADD HOSTS <ip>:<port> [,<ip>:<port> ...];

#### Example:

nebula> ADD HOSTS 192.168.10.100:9779, 192.168.10.101:9779, 192.168.10.102:9779;

## Caution

Make sure that the IP you added is the same as the IP configured for local\_ip in the nebula-storaged.conf file. Otherwise, the Storage service will fail to start. For information about configurations, see Configurations.

#### 2. Check the status of the hosts to make sure that they are all online.

nebula> SHOW HOSTS;

++	++	+	+	+		++
Host	Port   Statu	s   Leader count	t   Leader distri	bution   Par	tition distributio	n   Version
"192.168.10.100" "192.168.10.101" "192.168.10.102"	9779   "ONL] 9779   "ONL] 9779   "ONL]	NE"   O NE"   O NE"   O NE"   O	"No valid par   "No valid par   "No valid par	tition"   "No tition"   "No tition"   "No	valid partition" valid partition" valid partition"	"3.6.0"     "3.6.0"    "3.6.0"
+	+++	+	+	+		++

The Status column of the result above shows that all Storage hosts are online.

Last update: October 25, 2023

## 3.2.5 Step 4: Use nGQL (CRUD)

This topic will describe the basic CRUD operations in NebulaGraph.

For more information, see nGQL guide.

#### Graph space and NebulaGraph schema

A NebulaGraph instance consists of one or more graph spaces. Graph spaces are physically isolated from each other. You can use different graph spaces in the same instance to store different datasets.



To insert data into a graph space, define a schema for the graph database. NebulaGraph schema is based on the following components.

Schema component	Description
Vertex	Represents an entity in the real world. A vertex can have zero to multiple tags.
Tag	The type of the same group of vertices. It defines a set of properties that describes the types of vertices.
Edge	Represents a <b>directed</b> relationship between two vertices.
Edge type	The type of an edge. It defines a group of properties that describes the types of edges.

For more information, see Data modeling.

In this topic, we will use the following dataset to demonstrate basic CRUD operations.



ASYNC IMPLEMENTATION OF CREATE AND ALTER

## Caution

In NebulaGraph, the following CREATE or ALTER commands are implemented in an async way and take effect in the **next** heartbeat cycle. Otherwise, an error will be returned. To make sure the follow-up operations work as expected, Wait for two heartbeat cycles, i.e., 20 seconds.

- CREATE SPACE
- CREATE TAG
- CREATE EDGE
- ALTER TAG
- ALTER EDGE
- CREATE TAG INDEX
- CREATE EDGE INDEX

#### O Note

The default heartbeat interval is 10 seconds. To change the heartbeat interval, modify the heartbeat\_interval\_secs parameter in the configuration files for all services.

#### Create and use a graph space

NGQL SYNTAX

• Create a graph space:

[COMMENT = '<comment>'];

For more information on parameters, see CREATE SPACE.

• List graph spaces and check if the creation is successful:

nebula> SHOW SPACES;

• Use a graph space:

USE <graph\_space\_name>;

EXAMPLES

1. Use the following statement to create a graph space named basketballplayer.

nebula> CREATE SPACE basketballplayer(partition\_num=15, replica\_factor=1, vid\_type=fixed\_string(30));



If the system returns the error [ERROR (-1005)]: Host not enough!, check whether registered the Storage Service.

2. Check the partition distribution with SHOW HOSTS to make sure that the partitions are distributed in a balanced way.

nebula> SHOW HOSTS;						
Host   Por	t   Status	Leader count   Leader distribution	Partition distribution   Version			
"storaged0"   977   "storaged1"   977   "storaged1"   977	9   "ONLINE" 9   "ONLINE"	5   "basketballplayer:5" 5   "basketballplayer:5"	"basketballplayer:5"   "3.6.0"    "basketballplayer:5"   "3.6.0"    "basketballplayer:5"   "3.6.0"			

If the **Leader distribution** is uneven, use BALANCE LEADER to redistribute the partitions. For more information, see BALANCE.

## 3. Use the basketballplayer graph space.

nebula[(none)]> USE basketballplayer;

You can use SHOW SPACES to check the graph space you created.

nebula> SHOW SPACES; +-----+ | Name | +----+ | "basketballplayer" | +----++

#### Create tags and edge types

#### NGQL SYNTAX

For more information on parameters, see CREATE TAG and CREATE EDGE.

#### EXAMPLES

Create tags player and team, and edge types follow and serve. Descriptions are as follows.

Component name	Туре	Property
player	Tag	name (string), age (int)
team	Tag	name (string)
follow	Edge type	degree (int)
serve	Edge type	start_year (int), end_year (int)

nebula> CREATE TAG player(name string, age int); nebula> CREATE TAG team(name string); nebula> CREATE EDGE follow(degree int);

nebula> CREATE EDGE serve(start\_year int, end\_year int);

#### Insert vertices and edges

You can use the INSERT statement to insert vertices or edges based on existing tags or edge types.

NGQL SYNTAX

• Insert vertices:

```
INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...]
VALUES <vid>>: ([prop_value_list])
tag_props:
    tag_name ([prop_name_list])
```

```
prop_name_list:
   [prop_name [, prop_name] ...]
prop_value_list:
   [prop_value [, prop_value] ...]
```

vid is short for Vertex ID. A vid must be a unique string value in a graph space. For details, see INSERT VERTEX.

#### • Insert edges:

```
INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) VALUES
<src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> )
[, <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ), ...];
<prop_name_list> ::=
[ <prop_name> [, <prop_name> ] ...]
<prop_value_list> ::=
[ <prop_value_[, <prop_value> ] ...]
```

For more information on parameters, see INSERT EDGE.

#### EXAMPLES

#### • Insert vertices representing basketball players and teams:

nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42); nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36); nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33); nebula> INSERT VERTEX team(name) VALUES "team203":("Trail Blazers"), "team204":("Spurs");

• Insert edges representing the relations between basketball players and teams:

nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player100":(95); nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player102":(90); nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player100":(75); nebula> INSERT EDGE serve(start\_year, end\_year) VALUES "player101" -> "team204":(1999, 2018),"player102" -> "team203":(2006, 2015);

#### Read data

- The GO statement can traverse the database based on specific conditions. A 60 traversal starts from one or more vertices, along one or more edges, and returns information in a form specified in the YIELD clause.
- The FETCH statement is used to get properties from vertices or edges.
- The LOOKUP statement is based on indexes. It is used together with the WHERE clause to search for the data that meet the specific conditions.
- The MATCH statement is the most commonly used statement for graph data querying. It can describe all kinds of graph patterns, but it relies on indexes to match data patterns in NebulaGraph. Therefore, its performance still needs optimization.

NGQL SYNTAX

```
• GO
```

```
G0 [[4]> T0] <N> {STEP|STEPS} ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
YIELD [DISTINCT] <return_list>
[ { SAMPLE <sample_list> {<limit_by_list_clause> }]
[ | GROUP BY {<col_name> | expression> | <position>} YIELD <col_name>]
```

```
[| ORDER BY <expression> [{ASC | DESC}]]
[| LIMIT [<offset>,] <number_rows>];
```

#### • FETCH

#### • Fetch properties on tags:

```
FETCH PROP ON {<tag_name>[, tag_name ...] | *}
<vid> [, vid ...]
YIELD <return_list> [AS <alias>];
```

#### • Fetch properties on edges:

```
FETCH PROP ON <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid> ...]
YIELD <output>;
```

• LOOKUP

```
LOOKUP ON {<vertex_tag> | <edge_type>}
[WHERE <expression> [AND <expression> ...]]
YIELD <return_list> [AS <alias>];
<return_list>
<prop_name> [AS <col_alias>] [, <prop_name> [AS <prop_alias>] ...];
```

• MATCH

MATCH <pattern> [<clause\_1>] RETURN <output> [<clause\_2>];

EXAMPLES OF 60 STATEMENT

• Search for the players that the player with VID player101 follows.

```
nebula> 60 FROM "player101" OVER follow YIELD id($$);
+-----+
| id($$) |
+-----+
| "player100" |
| "player102" |
```

| "player125" |

• Filter the players that the player with VID player101 follows whose age is equal to or greater than 35. Rename the corresponding columns in the results with Teammate and Age.

- Search for the players that the player with VID player101 follows. Then retrieve the teams of the players that the player with VID player100 follows. To combine the two queries, use a pipe or a temporary variable.
- With a pipe:

```
nebula> GO FROM "player101" OVER follow YIELD dst(edge) AS id | \
GO FROM $-.id OVER serve YIELD properties($$).name AS Team, \
properties($^).name AS Player;
+-----+
Team Player |
Team Player |
"Spurs" | "Tim Duncan" |
"Trail Blazers" | "LaMarcus Aldridge" |
"Spurs" | "Amarcus Aldridge" |
"Spurs" | "Manu Ginobili" |
```

Clause/Sign	Description
\$^	Represents the source vertex of the edge.
Ф	A pipe symbol can combine multiple queries.
\$-	Represents the outputs of the query before the pipe symbol.

• With a temporary variable:

## Note

Once a composite statement is submitted to the server as a whole, the life cycle of the temporary variables in the statement ends.

nebula>	\$var = GO F GO FROM \$va properties(	ROM "player101" r.id OVER serve \$^).name AS Play	OVER follow YIELD proper /er;	YIELD ties(\$	dst(edge) \$).name AS	AS id;∖ S Team,∖
+   Team +	++	Player	+   +			
"Spurs   "Trail   "Spurs   "Spurs +	"   Blazers"       	"Tim Duncan" "LaMarcus Aldri "LaMarcus Aldri "Manu Ginobili'	  dge"    dge"    +			

EXAMPLE OF FETCH STATEMENT

Use FETCH : Fetch the properties of the player with VID player100 .

```
nebula> FETCH PROP ON player "player100" YIELD properties(vertex);
+-----+
| properties(VERTEX) |
+-----+
| {age: 42, name: "Tim Duncan"} |
+-----+
```

Q Note

The examples of LOOKUP and MATCH statements are in indexes.

#### Update vertices and edges

Users can use the UPDATE or the UPSERT statements to update existing data.

UPSERT is the combination of UPDATE and INSERT. If you update a vertex or an edge with UPSERT, the database will insert a new vertex or edge if it does not exist.

Note

UPSERT operates serially in a partition-based order. Therefore, it is slower than INSERT OR UPDATE. And UPSERT has concurrency only between multiple partitions.

NGQL SYNTAX

• UPDATE vertices:

```
UPDATE VERTEX <vid> SET <properties to be updated>
[WHEN <condition>] [YIELD <columns>];
```

• UPDATE edges:

UPDATE EDGE ON <edge\_type> <source vid> -> <destination vid> [@rank] SET <properties to be updated> [WHEN <condition>] [YIELD <columns to be output>];

• UPSERT vertices or edges:

```
UPSERT {VERTEX <vid> | EDGE <edge_type>} SET <update_columns>
[WHEN <condition>] [YIELD <columns>];
```

EXAMPLES

• UPDATE the name property of the vertex with VID player100 and check the result with the FETCH statement.

nebula> UPDATE VERTEX "player100" SET player.name = "Tim"; nebula> FETCH PROP ON player "player100" YIELD properties(vertex); +-----+ | properties(VERTEX) | | {age: 42, name: "Tim"} |

• UPDATE the degree property of an edge and check the result with the FETCH statement.

nebula> UPDATE EDGE ON follow "player101" -> "player100" SET degree = 96; nebula> FETCH PROP ON follow "player101" -> "player100" YIELD properties(edge); +-----+ | properties(EDGE) | +-----+ | {degree: 96} |

• Insert a vertex with VID player111 and UPSERT it.

nebula> INSERT VERTEX player(name,age) VALUES "player111":("David West", 38);

```
nebula> UPSERT VERTEX "player111" SET player.name = "David", player.age = $^.player.age + 11 \
    WHEN $^.player.name == "David West" AND $^.player.age > 20 \
    YIELD $^.player.name AS Name, $^.player.age AS Age;
+-----+
| Name | Age |
+------+
| "David" | 49 |
+------+
```

#### Delete vertices and edges

NGQL SYNTAX

• Delete vertices:

DELETE VERTEX <vid1>[, <vid2>...]

• Delete edges:

DELETE EDGE <edge\_type> <src\_vid> -> <dst\_vid>[@<rank>]
[, <src\_vid> -> <dst\_vid>...]

EXAMPLES

• Delete vertices:

nebula> DELETE VERTEX "player111", "team203";

• Delete edges:

nebula> DELETE EDGE follow "player101" -> "team204";

#### About indexes

Users can add indexes to tags and edge types with the CREATE INDEX statement.

## Nust-read for using indexes

Both MATCH and LOOKUP statements depend on the indexes. But indexes can dramatically reduce the write performance. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.

Users **MUST** rebuild indexes for pre-existing data. Otherwise, the pre-existing data cannot be indexed and therefore cannot be returned in MATCH or LOOKUP statements. For more information, see REBUILD INDEX.

NGQL SYNTAX

#### • Create an index:

```
CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index_name>
ON {<tag_name> | <edge_name>} ([<prop_name_list>]) [COMMENT = '<comment>'];
```

#### • Rebuild an index:

REBUILD {TAG | EDGE} INDEX <index\_name>;

#### Q Note

Define the index length when creating an index for a variable-length property. In UTF-8 encoding, a non-ascii character occupies 3 bytes. You should set an appropriate index length according to the variable-length property. For example, the index should be 30 bytes for 10 non-ascii characters. For more information, see CREATE INDEX

EXAMPLES OF LOOKUP AND MATCH (INDEX-BASED)

Make sure there is an index for LOOKUP or MATCH to use. If there is not, create an index first.

Find the information of the vertex with the tag player and its value of the name property is Tony Parker.

This example creates the index player\_index\_1 on the name property.

nebula> CREATE TAG INDEX IF NOT EXISTS player\_index\_1 ON player(name(20));

This example rebuilds the index to make sure it takes effect on pre-existing data.

nebula> REBUILD TAG INDEX player\_index\_1 +-----+ | New Job Id | +-----+ | 31 | +-----+

This example uses the LOOKUP statement to retrieve the vertex property.

```
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
    YIELD properties(vertex).name AS name, properties(vertex).age AS age;
+-----+
| name | age |
+-----+
| "Tony Parker" | 36 |
+-----+
```

This example uses the MATCH statement to retrieve the vertex property.

Last update: November 3, 2023

# 3.3 nGQL cheatsheet

## 3.3.1 Functions

• Math functions

Function	Description
double abs(double x)	Returns the absolute value of the argument.
double floor(double x)	Returns the largest integer value smaller than or equal to the argument. (Rounds down)
double ceil(double x)	Returns the smallest integer greater than or equal to the argument. (Rounds up)
double round(double x)	Returns the integer value nearest to the argument. Returns a number farther away from 0 if the argument is in the middle.
double sqrt(double x)	Returns the square root of the argument.
double cbrt(double x)	Returns the cubic root of the argument.
double hypot(double x, double y)	Returns the hypotenuse of a right-angled triangle.
double pow(double x, double y)	Returns the result of $x^y$ .
double exp(double x)	Returns the result of $e^{X}$ .
double exp2(double x)	Returns the result of $2^{X}$ .
double log(double x)	Returns the base-e logarithm of the argument.
double log2(double x)	Returns the base-2 logarithm of the argument.
double log10(double x)	Returns the base-10 logarithm of the argument.
double sin(double x)	Returns the sine of the argument.
double asin(double x)	Returns the inverse sine of the argument.
double cos(double x)	Returns the cosine of the argument.
double acos(double x)	Returns the inverse cosine of the argument.
double tan(double x)	Returns the tangent of the argument.
double atan(double x)	Returns the inverse tangent of the argument.
double rand()	Returns a random floating point number in the range from 0 (inclusive) to 1 (exclusive); i.e. $[0,1)$ .
int rand32(int min, int max)	Returns a random 32-bit integer in [min, max). If you set only one argument, it is parsed as max and min is 0 by default. If you set no argument, the system returns a random signed 32-bit integer.
int rand64(int min, int max)	Returns a random 64-bit integer in [min, max). If you set only one argument, it is parsed as max and min is 0 by default. If you set no argument, the system returns a random signed 64-bit integer.
bit_and()	Bitwise AND.
bit_or()	Bitwise OR.
bit_xor()	Bitwise XOR.
int size()	Returns the number of elements in a list or a map or the length of a string.
int range(int start, int end, int step)	Returns a list of integers from [start,end] in the specified steps. step is 1 by default.
int sign(double x)	Returns the signum of the given number. If the number is 0, the system returns 0. If the number is negative, the system returns -1. If the number is positive, the system returns 1.

Function	Description	
double e()	Returns the base of the natural logarithm, e (2.718281828459045).	
double pi()	Returns the mathematical constant pi (3.141592653589793).	
double radians()	Converts degrees to radians. radians(180) returns 3.141592653589793.	

## • Aggregating functions

Function	Description
avg()	Returns the average value of the argument.
count()	<pre>Syntax: count({expr   *}) . count() returns the number of rows (including NULL). count(expr) returns the number of non-NULL values that meet the expression. count() and size() are different.</pre>
max()	Returns the maximum value.
min()	Returns the minimum value.
collect()	The collect() function returns a list containing the values returned by an expression. Using this function aggregates data by merging multiple records or values into a single list.
std()	Returns the population standard deviation.
sum()	Returns the sum value.

## • String functions

int strcasecmp(string a, string b)	Compares string a and b without case sensitivity. When $a = b$ , the return
string lower(string a)	Returns the argument in lowercase.
string toLower(string a)	The same as lower().
string upper(string a)	Returns the argument in uppercase.
string toUpper(string a)	The same as upper().
int length(a)	Returns the length of the given string in bytes or the length of a path in hops.
string trim(string a)	Removes leading and trailing spaces.
string ltrim(string a)	Removes leading spaces.
string rtrim(string a)	Removes trailing spaces.
string left(string a, int count)	Returns a substring consisting of count characters from the left side of
string right(string a, int count)	Returns a substring consisting of count characters from the right side of
string lpad(string a, int size, string letters)	Left-pads string a with string letters and returns a
string rpad(string a, int size, string letters)	Right-pads string a with string letters and returns a
string substr(string a, int pos, int count)	Returns a substring extracting count characters starting from
string substring(string a, int pos, int count)	The same as substr().
string reverse(string)	Returns a string in reverse order.
string replace(string a, string b, string c)	Replaces string b in string a with string c.
list split(string a, string b)	Splits string a at string b and returns a list of strings.
concat()	The concat() function requires at least two or more strings. All the parameters are concatenated into one string. Syntax: concat(string1,string2,)
concat_ws()	The concat_ws() function connects two or more strings with a predefined separator.
extract()	extract() uses regular expression matching to retrieve a single substring or all substrings from a string.
json_extract()	The json_extract() function converts the specified JSON string to map.

#### • Data and time functions

Function	Description
int now()	Returns the current timestamp of the system.
timestamp timestamp()	Returns the current timestamp of the system.
date date()	Returns the current UTC date based on the current system.
time time()	Returns the current UTC time based on the current system.
datetime datetime()	Returns the current UTC date and time based on the current system.

## • Schema-related functions

## • For nGQL statements

Function	Description
id(vertex)	Returns the ID of a vertex. The data type of the result is the same as the vertex ID.
map properties(vertex)	Returns the properties of a vertex.
map properties(edge)	Returns the properties of an edge.
string type(edge)	Returns the edge type of an edge.
src(edge)	Returns the source vertex ID of an edge. The data type of the result is the same as the vertex ID.
dst(edge)	Returns the destination vertex ID of an edge. The data type of the result is the same as the vertex ID.
int rank(edge)	Returns the rank value of an edge.
vertex	Returns the information of vertices, including VIDs, tags, properties, and values.
edge	Returns the information of edges, including edge types, source vertices, destination vertices, ranks, properties, and values.
vertices	Returns the information of vertices in a subgraph. For more information, see GET SUBGRAPH.
edges	Returns the information of edges in a subgraph. For more information, see GET SUBGRAPH.
path	Returns the information of a path. For more information, see FIND PATH.

## • For statements compatible with openCypher

Function	Description
id( <vertex>)</vertex>	Returns the ID of a vertex. The data type of the result is the same as the vertex ID.
list tags( <vertex>)</vertex>	Returns the Tag of a vertex, which serves the same purpose as labels().
list labels( <vertex>)</vertex>	Returns the Tag of a vertex, which serves the same purpose as tags(). This function is used for compatibility with openCypher syntax.
map properties( <vertex_or_edge>)</vertex_or_edge>	Returns the properties of a vertex or an edge.
string type( <edge>)</edge>	Returns the edge type of an edge.
<pre>src(<edge>)</edge></pre>	Returns the source vertex ID of an edge. The data type of the result is the same as the vertex ID.
dst( <edge>)</edge>	Returns the destination vertex ID of an edge. The data type of the result is the same as the vertex ID.
vertex startNode( <path>)</path>	Visits an edge or a path and returns its source vertex ID.
string endNode( <path>)</path>	Visits an edge or a path and returns its destination vertex ID.
int rank( <edge>)</edge>	Returns the rank value of an edge.

## • List functions

Function	Description	
keys(expr)	Returns a list containing the string representations for all the property names of vertices, edges, or maps.	
labels(vertex)	Returns the list containing all the tags of a vertex.	
nodes(path)	Returns the list containing all the vertices in a path.	
range(start, end [, step])	Returns the list containing all the fixed-length steps in $\cite{start,end}\c$	
relationships(path)	Returns the list containing all the relationships in a path.	
reverse(list)	Returns the list reversing the order of all elements in the original list.	
tail(list)	Returns all the elements of the original list, excluding the first one.	
head(list)	Returns the first element of a list.	
last(list)	Returns the last element of a list.	
reduce()	The reduce() function applies an expression to each element in a list one by one, chains the result to the next iteration by taking it as the initial value, and returns the final result.	

#### • Type conversion functions

Function	Description
bool toBoolean()	Converts a string value to a boolean value.
float toFloat()	Converts an integer or string value to a floating point number.
string toString()	Converts non-compound types of data, such as numbers, booleans, and so on, to strings.
int toInteger()	Converts a floating point or string value to an integer value.
set toSet()	Converts a list or set value to a set value.
int hash()	The hash() function returns the hash value of the argument. The argument can be a number, a string, a list, a boolean, null, or an expression that evaluates to a value of the preceding data types.

## • Predicate functions

 $\label{eq:predicate functions return true or false. They are most commonly used in {\tt WHERE } clauses.$ 

<predicate>(<variable> IN <list> WHERE <condition>)</condition></list></variable></predicate>		
Function	Description	
exists()	Returns true if the specified property exists in the vertex, edge or map. Otherwise, returns false.	
any()	Returns true if the specified predicate holds for at least one element in the given list. Otherwise, returns false .	
all()	Returns true if the specified predicate holds for all elements in the given list. Otherwise, returns false.	
none()	Returns true if the specified predicate holds for no element in the given list. Otherwise, returns false.	
single()	Returns true if the specified predicate holds for exactly one of the elements in the given list. Otherwise, returns false.	

## $\hfill \bullet$ Conditional expressions functions

Function	Description
CASE	The CASE expression uses conditions to filter the result of an nGQL query statement. It is usually used in the YIELD and RETURN clauses. The CASE expression will traverse all the conditions. When the first condition is met, the CASE expression stops reading the conditions and returns the result. If no conditions are met, it returns the result in the ELSE clause. If there is no ELSE clause and no conditions are met, it returns NULL.
coalesce()	Returns the first not null value in all expressions.

## 3.3.2 General queries statements

• MATCH

MATCH <pattern> [<clause\_1>] RETURN <output> [<clause\_2>];

Pattern	Example	Description
Match vertices	(v)	You can use a user-defined variable in a pair of parentheses to represent a vertex in a pattern. For example: $\left(v\right)$ .
Match tags	MATCH (v:player) RETURN v	You can specify a tag with : <tag_name> after the vertex in a pattern.</tag_name>
Match multiple tags	MATCH (v:player:team) RETURN v	To match vertices with multiple tags, use colons (:).
Match vertex properties	MATCH (v:player{name:"Tim Duncan"}) RETURN v	You can specify a vertex property with { <prop_name>: <prop_value>} after the tag in a pattern; or use a vertex property value to get vertices directly.</prop_value></prop_name>
	<pre>MATCH (v) WITH v, properties(v) as props, keys(properties(v)) as kk WHERE [i in kk where props[i] == "Tim Duncan"] RETURN v</pre>	
Match a VID.	MATCH (v) WHERE id(v) == 'player101' RETURN v	You can use the VID to match a vertex. The $\mbox{id}(\mbox{)}$ function can retrieve the VID of a vertex.
Match multiple VIDs.	MATCH (v:player { name: 'Tim Duncan' })(v2) WHERE id(v2) IN ["player101", "player102"] RETURN v2	To match multiple VIDs, use $\ensuremath{\tt WHERE}\ id(v)\ IN\ [vid_list]$ .
Match connected vertices	MATCH (v:player{name:"Tim Duncan"}) (v2) RETURN v2.player.name AS Name	You can use the symbol to represent edges of both directions and match vertices connected by these edges. You can add a > or < to the symbol to specify the direction of an edge.
Match paths	MATCH p=(v:player{name:"Tim Duncan"})>(v2) RETURN p	Connected vertices and edges form a path. You can use a user-defined variable to name a path as follows.
Match edges	MATCH (v:player{name:"Tim Duncan"})- [e]-(v2) RETURN e MATCH ()<-[e]-() RETURN e	Besides using ,> , or < to indicate a nameless edge, you can use a user-defined variable in a pair of square brackets to represent a named edge. For example: -[e]
Match an edge type	MATCH ()-[e:follow]-() RETURN e	Just like vertices, you can specify an edge type with : <edge_type> in a pattern. For example: -[e:follow]</edge_type>
Match edge type properties	<pre>MATCH (v:player{name:"Tim Duncan"})- [e:follow{degree:95}]-&gt;(v2) RETURN e MATCH ()-[e]-&gt;() WITH e, properties(e) as props, keys(properties(e)) as kk WHERE [i in kk where props[i] == 90] RETURN e</pre>	You can specify edge type properties with { <prop_name>: <prop_value>} in a pattern. For example: [e:follow{Likeness:95}]; or use an edge type property value to get edges directly.</prop_value></prop_name>
Match multiple edge types	MATCH (v:player{name:"Tim Duncan"})- [e:follow   :serve]->(v2) RETURN e	The   symbol can help matching multiple edge types. For example: [e:follow :serve]. The English colon (:) before the first edge type cannot be omitted, but the English colon before the subsequent edge type can be omitted, such as [e:follow serve].
Match multiple edges	<pre>MATCH (v:player{name:"Tim Duncan"})- []-&gt;(v2)&lt;-[e:serve]-(v3) RETURN v2, v3</pre>	You can extend a pattern to match multiple edges in a path.
Match fixed- length paths	MATCH p=(v:player{name:"Tim Duncan"})- [e:follow*2]->(v2) RETURN DISTINCT v2 AS Friends	You can use the : <edge_type>*<hop> pattern to match a fixed- length path. hop must be a non-negative integer. The data type of e is the list.</hop></edge_type>
Match variable- length paths	MATCH p=(v:player{name:"Tim Duncan"})- [e:follow*13]->(v2) RETURN v2 AS Friends	minHop : Optional. It represents the minimum length of the path. minHop : must be a non-negative integer. The default value is 1.

Pattern	Example	<b>Description</b> minHop and maxHop are optional and the default value is 1 and infinity respectively. The data type of e is the list.	
Match variable- length paths with multiple edge types	MATCH p=(v:player{name:"Tim Duncan"})- [e:follow   serve*2]->(v2) RETURN DISTINCT v2	You can specify multiple edge types in a fixed-length or variable-length pattern. In this case, hop, minHop, and maxHop take effect on all edge types. The data type of e is the list.	
Retrieve vertex or edge information	MATCH (v:player{name:"Tim Duncan"}) RETURN v MATCH (v:player{name:"Tim Duncan"})- [e]->(v2) RETURN e	<pre>ncan"}) Use RETURN {<vertex_name>   <edge_name>} to retrieve all the information of a vertex or an edge. ncan"})-</edge_name></vertex_name></pre>	
Retrieve VIDs	<pre>MATCH (v:player{name:"Tim Duncan"}) RETURN id(v)</pre>	Use the id() function to retrieve VIDs.	
Retrieve tags	MATCH (v:player{name:"Tim Duncan"}) RETURN labels(v)	Use the labels() function to retrieve the list of tags on a vertex. To retrieve the nth element in the labels(v) list, use labels(v) [n-1].	
Retrieve a single property on a vertex or an edge	MATCH (v:player{name:"Tim Duncan"}) RETURN v.player.age	Use RETURN { <vertex_name>   <edge_name>}.<property> to retrieve a single property. Use AS to specify an alias for a property.</property></edge_name></vertex_name>	
Retrieve all properties on a vertex or an edge	<pre>MATCH p=(v:player{name:"Tim Duncan"})- []-&gt;(v2) RETURN properties(v2)</pre>	Use the properties() function to retrieve all properties on a vertex or an edge.	
Retrieve edge types	<pre>MATCH p=(v:player{name:"Tim Duncan"})- [e]-&gt;() RETURN DISTINCT type(e)</pre>	Use the type() function to retrieve the matched edge types.	
Retrieve paths	<pre>MATCH p=(v:player{name:"Tim Duncan"})- [*3]-&gt;() RETURN p</pre>	Use RETURN <path_name> to retrieve all the information of the matched paths.</path_name>	
Retrieve vertices in a path	<pre>MATCH p=(v:player{name:"Tim Duncan"})- []-&gt;(v2) RETURN nodes(p)</pre>	Use the $\ensuremath{\operatorname{nodes}}()$ function to retrieve all vertices in a path.	
Retrieve edges in a path	<pre>MATCH p=(v:player{name:"Tim Duncan"})- []-&gt;(v2) RETURN relationships(p)</pre>	Use the relationships() function to retrieve all edges in a path.	
Retrieve path length	<pre>MATCH p=(v:player{name:"Tim Duncan"})- [*2]-&gt;(v2) RETURN p AS Paths, length(p) AS Length</pre>	Use the length() function to retrieve the length of a path.	

## • OPTIONAL MATCH

Pattern	Example	Description
Matches patterns against your graph database, just like MATCH does.	MATCH (m)-[]->(n) WHERE id(m)=="player100" OPTIONAL MATCH (n)-[]->(l) RETURN id(m),id(n),id(l)	If no matches are found, OPTIONAL MATCH will use a null for missing parts of the pattern.

## • LOOKUP

LOOKUP ON { <vertex_tag>   <edge_type>} [WHERE <expression> [AND <expression>]] YIELD <return_list> [AS <alias>]</alias></return_list></expression></expression></edge_type></vertex_tag>			
Pattern	Example	Description	
Retrieve vertices	LOOKUP ON player WHERE player.name == "Tony Parker" YIELD player.name AS name, player.age AS age	The following example returns vertices whose name is Tony Parker and the tag is player.	
Retrieve edges	LOOKUP ON follow WHERE follow.degree == 90 YIELD follow.degree	Returns edges whose degree is 90 and the edge type is follow.	
List vertices with a tag	LOOKUP ON player YIELD properties(vertex),id(vertex)	Shows how to retrieve the VID of all vertices tagged with $\ensuremath{\operatorname{player}}$ .	
List edges with an edge types	LOOKUP ON follow YIELD edge AS e	Shows how to retrieve the source Vertex IDs, destination vertex IDs, and ranks of all edges of the follow edge type.	
Count the numbers of vertices or edges	LOOKUP ON player YIELD id(vertex)  YIELD COUNT(*) AS Player_Count	Shows how to count the number of vertices tagged with $\ensuremath{ \text{player}}$ .	
Count the numbers of edges	LOOKUP ON follow YIELD edge as e  YIELD COUNT(*) AS Like_Count	Shows how to count the number of edges of the follow edge type.	

#### • GO

GO [[<M> TO] <N> {STEP|STEPS} ] FROM <vertex\_List> OVER <edge\_type\_List> [{REVERSELY | BIDIRECT}] [ WHERE <conditions> ] YIELD [DISTINCT] <return\_List> [{SAMPLE <sample\_List> | LIMIT <limit\_List>}] [ | GROUP BY {col\_name | expr | position} YIELD <col\_name>] [ | ORDER BY <expression> [{ASC | DESC}]] [ | LIMIT [<offset\_value>,] <number\_rows>]

Example	Description
GO FROM "player102" OVER serve YIELD dst(edge)	Returns the teams that player 102 serves.
GO 2 STEPS FROM "player102" OVER follow YIELD dst(edge)	Returns the friends of player 102 with 2 hops.
GO FROM "player100", "player102" OVER serve WHERE properties(edge).start_year > 1995 YIELD DISTINCT properties(\$\$).name AS team_name, properties(edge).start_year AS start_year, properties(\$^).name AS player_name	Adds a filter for the traversal.
GO FROM "player100" OVER follow, serve YIELD properties(edge).degree, properties(edge).start_year	The following example traverses along with multiple edge types. If there is no value for a property, the output is NULL .
GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS destination	The following example returns the neighbor vertices in the incoming direction of player 100.
GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS id   GO FROM \$id OVER serve WHERE properties(\$^).age > 20 YIELD properties(\$^).name AS FriendOf, properties( \$).name AS Team	The following example retrieves the friends of player 100 and the teams that they serve.
GO FROM "player102" OVER follow YIELD dst(edge) AS both	The following example returns all the neighbor vertices of player 102.
GO 2 STEPS FROM "player100" OVER follow YIELD src(edge) AS src, dst(edge) AS dst, properties(\$\$).age AS age   GROUP BY \$dst YIELD \$dst AS dst, collect_set(\$src) A src, collect(\$age) AS age	The following example the outputs according to age.

## • FETCH

#### • Fetch vertex properties

<pre>FETCH PROP ON {<tag_name>[, tag_name</tag_name></pre>	]	L	*}
<vid> [, vid]</vid>			
YIELD <return list=""> [AS <alias>]</alias></return>			

Example	Description
FETCH PROP ON player "player100" YIELD properties(vertex)	Specify a tag in the FETCH statement to fetch the vertex properties by that tag.
FETCH PROP ON player "player100" YIELD player.name AS name	Use a YIELD clause to specify the properties to be returned.
FETCH PROP ON player "player101", "player102", "player103" YIELD properties(vertex)	Specify multiple VIDs (vertex IDs) to fetch properties of multiple vertices. Separate the VIDs with commas.
FETCH PROP ON player, t1 "player100", "player103" YIELD properties(vertex)	Specify multiple tags in the FETCH statement to fetch the vertex properties by the tags. Separate the tags with commas.
FETCH PROP ON * "player100", "player106", "team200" YIELD properties(vertex)	Set an asterisk symbol $\star$ to fetch properties by all tags in the current graph space.

## • Fetch edge properties

FETCH PROP ON <edge\_type> <src\_vid> -> <dst\_vid>[@<rank>] [, <src\_vid> -> <dst\_vid> ...]
YIELD <output>;

Example	Description
<pre>FETCH PROP ON serve "player100" -&gt; "team204" YIELD properties(edge)</pre>	The following statement fetches all the properties of the serve edge that connects vertex "player100" and vertex "team204".
FETCH PROP ON serve "player100" -> "team204" YIELD serve.start_year	Use a <code>YIELD</code> clause to fetch specific properties of an edge.
<pre>FETCH PROP ON serve "player100" -&gt; "team204", "player133" -&gt; "team202" YIELD properties(edge)</pre>	Specify multiple edge patterns ( $ -> [@]$ ) to fetch properties of multiple edges. Separate the edge patterns with commas.
<pre>FETCH PROP ON serve "player100" -&gt; "team204"@1 YIELD properties(edge)</pre>	To fetch on an edge whose rank is not 0, set its rank in the FETCH statement.
GO FROM "player101" OVER follow YIELD followsrc AS s, followdst AS d   FETCH PROP ON follow \$s -> \$d YIELD follow.degree	The following statement returns the degree values of the follow edges that start from vertex "player101" .
<pre>\$var = G0 FROM "player101" OVER follow YIELD followsrc AS s, followdst AS d; FETCH PROP ON follow \$var.s -&gt; \$var.d YIELD follow.degree</pre>	You can use user-defined variables to construct similar queries.

## • SHOW

Statement	Syntax	Example	Description
SHOW CHARSET	SHOW CHARSET	SHOW CHARSET	Shows the available character sets.
SHOW COLLATION	SHOW COLLATION	SHOW COLLATION	Shows the collations supported by NebulaGraph.
SHOW CREATE SPACE	SHOW CREATE SPACE <space_name></space_name>	SHOW CREATE SPACE basketballplayer	Shows the creating statement of the specified graph space.
SHOW CREATE TAG/EDGE	SHOW CREATE {TAG <tag_name>   EDGE <edge_name>}</edge_name></tag_name>	SHOW CREATE TAG player	Shows the basic information of the specified tag.
SHOW HOSTS	SHOW HOSTS [GRAPH   STORAGE   META]	SHOW HOSTS SHOW HOSTS GRAPH	Shows the host and version information of Graph Service, Storage Service, and Meta Service.
SHOW INDEX STATUS	SHOW {TAG   EDGE} INDEX STATUS	SHOW TAG INDEX STATUS	Shows the status of jobs that rebuild native indexes, which helps check whether a native index is successfully rebuilt or not.
SHOW INDEXES	SHOW {TAG   EDGE} INDEXES	SHOW TAG INDEXES	Shows the names of existing native indexes.
SHOW PARTS	SHOW PARTS [ <part_id>]</part_id>	SHOW PARTS	Shows the information of a specified partition or all partitions in a graph space.
SHOW ROLES	SHOW ROLES IN <space_name></space_name>	SHOW ROLES in basketballplayer	Shows the roles that are assigned to a user account.
SHOW SNAPSHOTS	SHOW SNAPSHOTS	SHOW SNAPSHOTS	Shows the information of all the snapshots.
SHOW SPACES	SHOW SPACES	SHOW SPACES	Shows existing graph spaces in NebulaGraph.
SHOW STATS	SHOW STATS	SHOW STATS	Shows the statistics of the graph space collected by the latest STATS job.
SHOW TAGS/ EDGES	SHOW TAGS   EDGES	SHOW TAGS , SHOW EDGES	Shows all the tags in the current graph space.
SHOW USERS	SHOW USERS	SHOW USERS	Shows the user information.
SHOW SESSIONS	SHOW SESSIONS	SHOW SESSIONS	Shows the information of all the sessions.
SHOW SESSIONS	SHOW SESSION <session_id></session_id>	SHOW SESSION 1623304491050858	Shows a specified session with its ID.
SHOW QUERIES	SHOW [ALL] QUERIES	SHOW QUERIES	Shows the information of working queries in the current session.
SHOW META LEADER	SHOW META LEADER	SHOW META LEADER	Shows the information of the leader in the current Meta cluster.
## 3.3.3 Clauses and options

Clause	Syntax	Example	Description
GROUP BY	GROUP BY <var> YIELD <var>, <aggregation_function(var)></aggregation_function(var)></var></var>	GO FROM "player100" OVER follow BIDIRECT YIELD \$\$.player.name as Name   GROUP BY \$Name YIELD \$Name as Player, count(*) AS Name_Count	Finds all the vertices connected directly to vertex "player100", groups the result set by player names, and counts how many times the name shows up in the result set.
LIMIT	YIELD <var> [  LIMIT [<offset_value>,] <number_rows>]</number_rows></offset_value></var>	GO FROM "player100" OVER follow REVERSELY YIELD \$\$.player.name AS Friend, \$\$.player.age AS Age   ORDER BY \$Age, \$Friend   LIMIT 1, 3	Returns the 3 rows of data starting from the second row of the sorted output.
SKIP	RETURN <var> [SKIP <offset>] [LIMIT <number_rows>]</number_rows></offset></var>	MATCH (v:player{name:"Tim Duncan"})> (v2) RETURN v2.player.name AS Name, v2.player.age AS Age ORDER BY Age DESC SKIP 1	SKIP can be used alone to set the offset and return the data after the specified position.
SAMPLE	<go_statement> SAMPLE <sample_list>;</sample_list></go_statement>	<pre>G0 3 STEPS FROM "player100" OVER * YIELD properties(\$\$).name AS NAME, properties(\$\$).age AS Age SAMPLE [1,2,3];</pre>	Takes samples evenly in the result set and returns the specified amount of data.
ORDER BY	<yield clause=""> ORDER BY <expression> [ASC   DESC] [, <expression> [ASC   DESC]]</expression></expression></yield>	FETCH PROP ON player "player100", "player101", "player102", "player103" YIELD player.age AS age, player.name AS name   ORDER BY \$age ASC, \$name DESC	The ORDER BY clause specifies the order of the rows in the output.
RETURN	<pre>RETURN {<vertex_name> <edge_name>  <vertex_name>.<property>  <edge_name>.<property> }</property></edge_name></property></vertex_name></edge_name></vertex_name></pre>	MATCH (v:player) RETURN v.player.name, v.player.age LIMIT 3	Returns the first three rows with values of the vertex properties name and age.
TTL	<property_value_1>, <property_name_1> <property_value_1>, <property_name_2> <property_value_2>,) ttl_duration= <value_int>, ttl_col = <property_name></property_name></value_int></property_value_2></property_name_2></property_value_1></property_name_1></property_value_1>	CREATE TAG t2(a int, b int, c string) ttl_duration= 100, ttl_col = "a"	Create a tag and set the TTL options.
WHERE	WHERE { <vertex  edge_alias&gt;.<property_name> {&gt; == &lt; } <value>}</value></property_name></vertex  	MATCH (v:player) WHERE v.player.name == "Tim Duncan" XOR (v.player.age < 30 AND v.player.name == "Yao Ming") OR NOT (v.player.name == "Yao Ming" OR v.player.name == "Tim Duncan") RETURN v.player.name, v.player.age	The WHERE clause filters the output by conditions. The WHERE clause usually works in Native nGQL G0 and LOOKUP statements, and OpenCypher MATCH and WITH statements.
YIELD	<pre>YIELD [DISTINCT] <col/> [AS <alias>] [, <col/> [AS <alias>]] [WHERE <conditions>];</conditions></alias></alias></pre>	GO FROM "player100" OVER follow YIELD dst(edge) AS ID   FETCH PROP ON player \$ID YIELD player.age AS Age   YIELD AVG(\$Age) as Avg_age, count(*)as Num_friends	Finds the players that "player100" follows and calculates their average age.
WITH	<pre>MATCH \$expressions WITH {nodes()  Labels() }</pre>	MATCH p=(v:player{name:"Tim Duncan"}) () WITH nodes(p) AS n UNWIND n AS n1 RETURN DISTINCT n1	The WITH clause can retrieve the output from a query part, process it, and pass it to the next query part as the input.
UNWIND	UNWIND <list> AS <alias> <return clause&gt;</return </alias></list>	UNWIND [1,2,3] AS n RETURN n	Splits a list into rows.

## 3.3.4 Space statements

Statement	Syntax	Example	Description
CREATE SPACE	<pre>CREATE SPACE [IF NOT EXISTS] <graph_space_name> ( [partition_num =</graph_space_name></pre>	CREATE SPACE my_space_1 (vid_type=FIXED_STRING(30))	Creates a graph space with
CREATE SPACE	CREATE SPACE <new_graph_space_name> AS <ol> <li>graph_space_name&gt;</li> </ol></new_graph_space_name>	CREATE SPACE my_space_4 as my_space_3	Clone a graph. space.
USE	USE <graph_space_name></graph_space_name>	USE space1	Specifies a graph space as the current working graph space for subsequent queries.
SHOW SPACES	SHOW SPACES	SHOW SPACES	Lists all the graph spaces in the NebulaGraph examples.
DESCRIBE SPACE	<pre>DESC[RIBE] SPACE <graph_space_name></graph_space_name></pre>	DESCRIBE SPACE basketballplayer	Returns the information about the specified graph space.
CLEAR SPACE	CLEAR SPACE [IF EXISTS] <graph_space_name></graph_space_name>	Deletes the vertices and edges in a graph space, but does not delete the graph space itself and the schema information.	
DROP SPACE	<pre>DROP SPACE [IF EXISTS] <graph_space_name></graph_space_name></pre>	DROP SPACE basketballplayer	Deletes everything in the specified graph space.

## 3.3.5 TAG statements

Statement	Syntax	Example	Description
CREATE TAG	<pre>CREATE TAG [IF NOT EXISTS] <tag_name> ( <prop_name></prop_name></tag_name></pre>	CREATE TAG woman(name string, age int, married bool, salary double, create_time timestamp) TTL_DURATION = 100, TTL_COL = "create_time"	Creates a tag with the given name in a graph space.
DROP TAG	DROP TAG [IF EXISTS] <tag_name></tag_name>	DROP TAG test;	Drops a tag with the given name in the current working graph space.
ALTER TAG	<pre>ALTER TAG <tag_name> <alter_definition> [, alter_definition]] [ttl_definition [, ttl_definition]] [COMMENT = '<comment>']</comment></alter_definition></tag_name></pre>	ALTER TAG t1 ADD (p3 int, p4 string)	Alters the structure of a tag with the given name in a graph space. You can add or drop properties, and change the data type of an existing property. You can also set a TTL (Time- To-Live) on a property, or change its TTL duration.
SHOW TAGS	SHOW TAGS	SHOW TAGS	Shows the name of all tags in the current graph space.
DESCRIBE TAG	DESC[RIBE] TAG <tag_name></tag_name>	DESCRIBE TAG player	Returns the information about a tag with the given name in a graph space, such as field names, data type, and so on.
DELETE TAG	DELETE TAG <tag_name_list> FROM <vid></vid></tag_name_list>	DELETE TAG test1 FROM "test"	Deletes a tag with the given name on a specified vertex.

## 3.3.6 Edge type statements

Statement	Syntax	Example	Description
CREATE EDGE	<pre>CREATE EDGE [IF NOT EXISTS] <edge_type_name> ( <prop_name> <data_type> [NULL   NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'] [{, <prop_name> <data_type> [NULL   NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>']}] ) [TTL_DURATION = <ttl_duration>] [TTL_COL = <prop_name>] [COMMENT = '<comment>']</comment></prop_name></ttl_duration></comment></default_value></data_type></prop_name></comment></default_value></data_type></prop_name></edge_type_name></pre>	CREATE EDGE e1(p1 string, p2 int, p3 timestamp) TTL_DURATION = 100, TTL_COL = "p2"	Creates an edge type with the given name in a graph space.
DROP EDGE	DROP EDGE [IF EXISTS] <edge_type_name></edge_type_name>	DROP EDGE el	Drops an edge type with the given name in a graph space.
ALTER EDGE	ALTER EDGE <edge_type_name> <alter_definition> [, alter_definition]] [ttl_definition [, ttl_definition]] [COMMENT = '<comment>']</comment></alter_definition></edge_type_name>	ALTER EDGE e1 ADD (p3 int, p4 string)	Alters the structure of an edge type with the given name in a graph space.
SHOW EDGES	SHOW EDGES	SHOW EDGES	Shows all edge types in the current graph space.
DESCRIBE EDGE	<pre>DESC[RIBE] EDGE <edge_type_name></edge_type_name></pre>	DESCRIBE EDGE follow	Returns the information about an edge type with the given name in a graph space, such as field names, data type, and so on.

## 3.3.7 Vertex statements

Statement	Syntax	Example	Description
INSERT VERTEX	INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props]] VALUES <vid>: ([prop_value_list])</vid>	INSERT VERTEX t2 (name, age) VALUES "13":("n3", 12), "14":("n4", 8)	Inserts one or more vertices into a graph space in NebulaGraph.
DELETE VERTEX	<pre>DELETE VERTEX <vid> [, <vid>]</vid></vid></pre>	DELETE VERTEX "team1"	Deletes vertices and the related incoming and outgoing edges of the vertices.
UPDATE VERTEX	UPDATE VERTEX ON <tag_name> <vid> SET <update_prop> [WHEN <condition>] [YIELD <output>]</output></condition></update_prop></vid></tag_name>	UPDATE VERTEX ON player "player101" SET age = age + 2	Updates properties on tags of a vertex.
UPSERT VERTEX	UPSERT VERTEX ON <tag> <vid> SET <update_prop> [WHEN <condition>] [YIELD <output>]</output></condition></update_prop></vid></tag>	UPSERT VERTEX ON player "player667" SET age = 31	The UPSERT statement is a combination of UPDATE and INSERT. You can use UPSERT VERTEX to update the properties of a vertex if it exists or insert a new vertex if it does not exist.

# 3.3.8 Edge statements

Statement	Syntax	Example	Description
INSERT EDGE	<pre>INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) VALUES <src_vid> -&gt; <dst_vid>[@<rank>] : ( <prop_value_list> ) [, <src_vid> -&gt; <dst_vid>[@<rank>] : ( <prop_value_list> ),]</prop_value_list></rank></dst_vid></src_vid></prop_value_list></rank></dst_vid></src_vid></prop_name_list></edge_type></pre>	<pre>INSERT EDGE e2 (name, age) VALUES "11"- &gt;"13":("n1", 1)</pre>	Inserts an edge or multiple edges into a graph space from a source vertex (given by src_vid) to a destination vertex (given by dst_vid) with a specific rank in NebulaGraph.
DELETE EDGE	<pre>DELETE EDGE <edge_type> <src_vid> -&gt; <dst_vid>[@<rank>] [, <src_vid> -&gt; <dst_vid>[@<rank>]]</rank></dst_vid></src_vid></rank></dst_vid></src_vid></edge_type></pre>	DELETE EDGE serve "player100" -> "team204"@0	Deletes one edge or multiple edges at a time.
UPDATE EDGE	UPDATE EDGE ON <edge_type> <src_vid> -&gt; <dst_vid> [@<rank>] SET <update_prop> [WHEN <condition>] [YIELD <output>]</output></condition></update_prop></rank></dst_vid></src_vid></edge_type>	UPDATE EDGE ON serve "player100" -> "team204"@O SET start_year = start_year + 1	Updates properties on an edge.
UPSERT EDGE	UPSERT EDGE ON <edge_type> <src_vid> -&gt; <dst_vid> [@rank] SET <update_prop> [WHEN <condition>] [YIELD <properties>]</properties></condition></update_prop></dst_vid></src_vid></edge_type>	UPSERT EDGE on serve "player666" -> "team200"@0 SET end_year = 2021	The UPSERT statement is a combination of UPDATE and INSERT. You can use UPSERT EDGE to update the properties of an edge if it exists or insert a new edge if it does not exist.

## 3.3.9 Index

## • Native index

You can use native indexes together with  $\ensuremath{\texttt{LOOKUP}}$  and  $\ensuremath{\texttt{MATCH}}$  statements.

Statement	Syntax	Example	Description
CREATE INDEX	<pre>CREATE {TAG   EDGE} INDEX [IF NOT EXISTS] <index_name> ON {<tag_name>   <edge_name>} ([<prop_name_list>]) [COMMENT = '<comment>']</comment></prop_name_list></edge_name></tag_name></index_name></pre>	CREATE TAG INDEX player_index on player()	Add native indexes for the existing tags, edge types, or properties.
SHOW CREATE INDEX	SHOW CREATE {TAG   EDGE} INDEX <index_name></index_name>	show create tag index index_2	Shows the statement used when creating a tag or an edge type. It contains detailed information about the index, such as its associated properties.
SHOW INDEXES	SHOW {TAG   EDGE} INDEXES	SHOW TAG INDEXES	Shows the defined tag or edge type indexes names in the current graph space.
DESCRIBE INDEX	DESCRIBE {TAG   EDGE} INDEX <index_name></index_name>	DESCRIBE TAG INDEX player_index_0	Gets the information about the index with a given name, including the property name (Field) and the property type (Type) of the index.
REBUILD INDEX	REBUILD {TAG   EDGE} INDEX [ <index_name_list>]</index_name_list>	REBUILD TAG INDEX single_person_index	Rebuilds the created tag or edge type index. If data is updated or inserted before the creation of the index, you must rebuild the indexes <b>manually</b> to make sure that the indexes contain the previously added data.
SHOW INDEX STATUS	SHOW {TAG   EDGE} INDEX STATUS	SHOW TAG INDEX STATUS	Returns the name of the created tag or edge type index and its status.
DROP INDEX	DROP {TAG   EDGE} INDEX [IF EXISTS] <index_name></index_name>	DROP TAG INDEX player_index_0	Removes an existing index from the current graph space.

## • Full-text index

Syntax	Example	Description
<pre>SIGN IN TEXT SERVICE [(<elastic_ip:port>   [,<username>, <password>]),   (<elastic_ip:port>),]</elastic_ip:port></password></username></elastic_ip:port></pre>	SIGN IN TEXT SERVICE (127.0.0.1:9200)	The full-text indexes is implemented based on Elasticsearch. After deploying an Elasticsearch cluster, you can use the SIGN IN statement to log in to the Elasticsearch client.
SHOW TEXT SEARCH CLIENTS	SHOW TEXT SEARCH CLIENTS	Shows text search clients.
SIGN OUT TEXT SERVICE	SIGN OUT TEXT SERVICE	Signs out to the text search clients.
CREATE FULLTEXT {TAG   EDGE} INDEX <index_name> ON {<tag_name>   <edge_name>} (<prop_name> [,<prop_name>]) [ANALYZER="<analyzer_name>"]</analyzer_name></prop_name></prop_name></edge_name></tag_name></index_name>	CREATE FULLTEXT TAG INDEX nebula_index_1 ON player(name)	Creates full-text indexes.
SHOW FULLTEXT INDEXES	SHOW FULLTEXT INDEXES	Show full-text indexes.
REBUILD FULLTEXT INDEX	REBUILD FULLTEXT INDEX	Rebuild full-text indexes.
DROP FULLTEXT INDEX <index_name></index_name>	DROP FULLTEXT INDEX nebula_index_1	Drop full-text indexes.
LOOKUP ON { <tag>   <edge_type>} WHERE ES_QUERY(<index_name>, "<text>") YIELD <return_list> [  LIMIT [<offset>,] <number_rows>]</number_rows></offset></return_list></text></index_name></edge_type></tag>	LOOKUP ON player WHERE ES_QUERY(fulltext_index_1,"Chris") YIELD id(vertex)	Use query options.

## 3.3.10 Subgraph and path statements

Туре	Syntax	Example	Description
GET SUBGRAPH	GET SUBGRAPH [WITH PROP] [ <step_count> {STEP STEPS}] FROM {<vid>, <vid>} [{IN   OUT   BOTH} <edge_type>, <edge_type>] YIELD [VERTICES AS <vertex_alias>] [,EDGES AS <edge_alias>]</edge_alias></vertex_alias></edge_type></edge_type></vid></vid></step_count>	GET SUBGRAPH 1 STEPS FROM "player100" YIELD VERTICES AS nodes, EDGES AS relationships	Retrieves information of vertices and edges reachable from the source vertices of the specified edge types and returns information of the subgraph.
FIND PATH	<pre>FIND { SHORTEST   ALL   NOLOOP } PATH [WITH PROP] FROM <vertex_id_list> TO <vertex_id_list> OVER <edge_type_list> [REVERSELY   BIDIRECT] [<where clause="">] [UPTO <n> {STEP STEPS}] YIELD path as <alias> [  ORDER BY \$path] [  LIMIT &lt;</alias></n></where></edge_type_list></vertex_id_list></vertex_id_list></pre>	FIND SHORTEST PATH FROM "player102" TO "team204" OVER * YIELD path as p	<pre>Finds the paths between the selected source vertices and destination vertices. A returned path is like (<vertex_id>)-[:<edge_type_name>@<rank>]- &gt;(<vertex_id).< pre=""></vertex_id).<></rank></edge_type_name></vertex_id></pre>

## 3.3.11 Query tuning statements

Туре	Syntax	Example	Description
EXPLAIN	EXPLAIN [format="row"   "dot"] <your_ngql_statement></your_ngql_statement>	EXPLAIN format="row" SHOW TAGS EXPLAIN format="dot" SHOW TAGS	Helps output the execution plan of an nGQL statement without executing the statement.
PROFILE	PROFILE [format="row"   "dot"] <your_ngql_statement></your_ngql_statement>	PROFILE format="row" SHOW TAGS EXPLAIN format="dot" SHOW TAGS	Executes the statement, then outputs the execution plan as well as the execution profile.

## 3.3.12 Operation and maintenance statements

## • SUBMIT JOB BALANCE

Syntax	Description
BALANCE LEADER	Starts a job to balance the distribution of all the storage leaders in graph spaces. It returns the job ID.

## • Job statements

Syntax	Description
SUBMIT JOB COMPACT	Triggers the long-term RocksDB compact operation.
SUBMIT JOB FLUSH	Writes the RocksDB memfile in the memory to the hard disk.
SUBMIT JOB STATS	Starts a job that makes the statistics of the current graph space. Once this job succeeds, you can use the SHOW STATS statement to list the statistics.
SHOW JOB <job_id></job_id>	Shows the information about a specific job and all its tasks in the current graph space. The Meta Service parses a SUBMIT JOB request into multiple tasks and assigns them to the nebula-storaged processes.
SHOW JOBS	Lists all the unexpired jobs in the current graph space.
STOP JOB	Stops jobs that are not finished in the current graph space.
RECOVER JOB	Re-executes the failed jobs in the current graph space and returns the number of recovered jobs.

## • Kill queries

Syntax	Example	Description
<pre>KILL QUERY (session=<session_id>,</session_id></pre>	KILL	Terminates the query being executed, and is
<pre>plan=<plan_id>)</plan_id></pre>	QUERY(SESSION=1625553545984255,PLAN=163)	often used to terminate slow queries.

### • Kill sessions

Syntax	Example	Description
<pre>KILL {SESSION SESSIONS} <sessionid></sessionid></pre>	KILL SESSION 1672887983842984	Terminates a single session.
SHOW SESSIONS   YIELD \$SessionId AS sid [WHERE <filter_clause>]   KILL {SESSION  SESSIONS} \$sid</filter_clause>	SHOW SESSIONS   YIELD \$SessionId AS sid, \$CreateTime as CreateTime   ORDER BY \$CreateTime ASC   LIMIT 2   KILL SESSIONS \$sid	Terminates multiple sessions based on specified criteria.
SHOW SESSIONS   KILL SESSIONS \$SessionId	SHOW SESSIONS   KILL SESSIONS \$SessionId	Terminates all sessions.

Last update: December 29, 2023

# 4. nGQL guide

## 4.1 nGQL overview

## 4.1.1 NebulaGraph Query Language (nGQL)

This topic gives an introduction to the query language of NebulaGraph, nGQL.

## What is nGQL

nGQL is a declarative graph query language for NebulaGraph. It allows expressive and efficient graph patterns. nGQL is designed for both developers and operations professionals. nGQL is an SQL-like query language, so it's easy to learn.

nGQL is a project in progress. New features and optimizations are done steadily. There can be differences between syntax and implementation. Submit an issue to inform the NebulaGraph team if you find a new issue of this type. NebulaGraph 3.0 or later releases will support openCypher 9.

#### What can nGQL do

- Supports graph traversals
- Supports pattern match
- Supports aggregation
- Supports graph mutation
- Supports access control
- Supports composite queries
- Supports index
- Supports most openCypher 9 graph query syntax (but mutations and controls syntax are not supported)

### Example data Basketballplayer

Users can download the example data Basketballplayer in NebulaGraph. After downloading the example data, you can import it to NebulaGraph by using the -f option in NebulaGraph Console.

## Note

Ensure that you have executed the ADD HOSTS command to add the Storage service to your NebulaGraph cluster before importing the example data. For more information, see Manage Storage hosts.

#### Placeholder identifiers and values

Refer to the following standards in nGQL:

- (Draft) ISO/IEC JTC1 N14279 SC 32 Database\_Languages GQL
- (Draft) ISO/IEC JTC1 SC32 N3228 SQL\_Property\_Graph\_Queries SQLPGQ
- OpenCypher 9

In template code, any token that is not a keyword, a literal value, or punctuation is a placeholder identifier or a placeholder value.

For details of the symbols in nGQL syntax, see the following table:

Token	Meaning	
< >	name of a syntactic element	
:	formula that defines an element	
[]	optional elements	
{ }	explicitly specified elements	
I	complete alternative elements	
	may be repeated any number of times	

For example, create vertices in nGQL syntax:

```
INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...]
VALUES <vid>: ([prop_value_list])
tag_props:
tag_name ([prop_name_list])
prop_name_list:
    [prop_name [, prop_name] ...]
prop_value_list:
    [prop_value [, prop_value] ...]
```

#### Example statement:

nebula> CREATE TAG IF NOT EXISTS player(name string, age int);

#### About openCypher compatibility

NATIVE NGQL AND OPENCYPHER

Native nGQL is the part of a graph query language designed and implemented by NebulaGraph. OpenCypher is a graph query language maintained by openCypher Implementers Group.

The latest release is openCypher 9. The compatible parts of openCypher in nGQL are called openCypher compatible sentences (short as openCypher).

Note		
nGQL = native nGQL + openCypher compatible sentences		

IS NGQL COMPATIBLE WITH OPENCYPHER 9 COMPLETELY?

NO.

# Empatibility with openCypher

nGQL is designed to be compatible with part of DQL (match, optional match, with, etc.).

• It is not planned to be compatible with any DDL, DML, or DCL.

• It is not planned to be compatible with the Bolt Protocol.

• It is not planned to be compatible with APOC and GDS.

Users can search in this manual with the keyword compatibility to find major compatibility issues.

Multiple known incompatible items are listed in NebulaGraph Issues. Submit an issue with the incompatible tag if you find a new issue of this type.

WHAT ARE THE MAJOR DIFFERENCES BETWEEN NGQL AND OPENCYPHER 9?

The following are some major differences (by design incompatible) between nGQL and openCypher.

Category	openCypher 9	nGQL
Schema	Optional Schema	Strong Schema
Equality operator	=	=
Math exponentiation	٨	$\wedge$ is not supported. Use pow(x, y) instead.
Edge rank	No such concept.	edge rank (reference by @)
Statement		All DMLs ( CREATE , MERGE , etc) of openCypher 9.
Label and tag	A label is used for searching a vertex, namely an index of vertex.	A tag defines the type of a vertex and its corresponding properties. It cannot be used as an index.
Pre-compiling and parameterized queries	Support	Parameterized queries are supported, but precompiling is not.

# Empatibility

OpenCypher 9 and Cypher have some differences in grammar and licence. For example,

Cypher requires that **All Cypher statements are explicitly run within a transaction**. While openCypher has no such requirement. And nGQL does not support transactions.

- 2. Cypher has a variety of constraints, including Unique node property constraints, Node property existence constraints, Relationship property existence constraints, and Node key constraints. While OpenCypher has no such constraints. As a strong schema system, most of the constraints mentioned above can be solved through schema definitions (including NOT NULL) in nGQL. The only function that cannot be supported is the UNIQUE constraint.
- Cypher has APoC, while openCypher 9 does not have APoC. Cypher has Blot protocol support requirements, while openCypher 9 does not.

WHERE CAN I FIND MORE NGQL EXAMPLES?

Users can find more than 2500 nGQL examples in the features directory on the NebulaGraph GitHub page.

The features directory consists of .feature files. Each file records scenarios that you can use as nGQL examples. Here is an example:

```
Feature: Basic match
 Background:
   Given a graph with space named "basketballplayer"
 Scenario: Single node
   When executing query:
     MATCH (v:player {name: "Yao Ming"}) RETURN v;
   Then the result should be, in any order, with relax comparison:
     ("player133" :player{age: 38, name: "Yao Ming"})
 Scenario: One step
   When executing query:
     MATCH (v1:player{name: "LeBron James"}) -[r]-> (v2)
     RETURN type(r) AS Type, v2.player.name AS Name
   Then the result should be, in any order:
       Туре
                  Name
                  "Ray Allen"
        "follow"
       "serve"
                 Lakers
                  "Heat"
        "serve"
       "serve"
                 Cavaliers"
```

The keywords in the preceding example are described as follows.

Keyword	Description
Feature	Describes the topic of the current .feature file.
Background	Describes the background information of the current .feature file.
Given	Describes the prerequisites of running the test statements in the current .feature file.
Scenario	Describes the scenarios. If there is the @skip before one Scenario, this scenario may not work and do not use it as a working example in a production environment.
When	Describes the nGQL statement to be executed. It can be a executing query or profiling query.
Then	Describes the expected return results of running the statement in the When clause. If the return results in your environment do not match the results described in the .feature file, submit an issue to inform the NebulaGraph team.
And	Describes the side effects of running the statement in the When clause.
@skip	This test case will be skipped. Commonly, the to-be-tested code is not ready.

Welcome to add more tck case and return automatically to the using statements in CI/CD.

DOES IT SUPPORT TINKERPOP GREMLIN?

No. And no plan to support that.

DOES NEBULAGRAPH SUPPORT W3C RDF (SPARQL) OR GRAPHQL?

No. And no plan to support that.

The data model of NebulaGraph is the property graph. And as a strong schema system, NebulaGraph does not support RDF.

 $Nebula Graph \ Query \ Language \ does \ not \ support \ \ SPARQL \ nor \ \ Graph QL \ .$ 

Last update: November 3, 2023

### 4.1.2 Patterns

Patterns and graph pattern matching are the very heart of a graph query language. This topic will describe the patterns in NebulaGraph, some of which have not yet been implemented.

#### Patterns for vertices

A vertex is described using a pair of parentheses and is typically given a name. For example:

(a)

This simple pattern describes a single vertex and names that vertex using the variable a.

#### Patterns for related vertices

A more powerful construct is a pattern that describes multiple vertices and edges between them. Patterns describe an edge by employing an arrow between two vertices. For example:

(a)-[]->(b)

This pattern describes a very simple data structure: two vertices and a single edge from one to the other. In this example, the two vertices are named as a and b respectively and the edge is directed: it goes from a to b.

This manner of describing vertices and edges can be extended to cover an arbitrary number of vertices and the edges between them, for example:

#### (a)-[]->(b)<-[]-(c)

Such a series of connected vertices and edges is called a path.

Note that the naming of the vertices in these patterns is only necessary when one needs to refer to the same vertex again, either later in the pattern or elsewhere in the query. If not, the name may be omitted as follows:

#### (a)-[]->()<-[]-(c)

#### Patterns for tags

## Note

The concept of tag in nGQL has a few differences from that of tabet in openCypher. For example, users must create a tag before using it. And a tag also defines the type of properties.

In addition to simply describing the vertices in the graphs, patterns can also describe the tags of the vertices. For example:

(a:User)-[]->(b)

Patterns can also describe a vertex that has multiple tags. For example:

(a:User:Admin)-[]->(b)

#### Patterns for properties

Vertices and edges are the fundamental elements in a graph. In nGQL, properties are added to them for richer models.

In the patterns, the properties can be expressed as follows: some key-value pairs are enclosed in curly brackets and separated by commas, and the tag or edge type to which a property belongs must be specified.

For example, a vertex with two properties will be like:

(a:player{name: "Tim Duncan", age: 42})

One of the edges that connect to this vertex can be like:

(a)-[e:follow{degree: 95}]->(b)

#### Patterns for edges

The simplest way to describe an edge is by using the arrow between two vertices, as in the previous examples.

Users can describe an edge and its direction using the following statement. If users do not care about its direction, the arrowhead can be omitted. For example:

#### (a)-[]-(b)

Like vertices, edges can also be named. A pair of square brackets will be used to separate the arrow and the variable will be placed between them. For example:

#### (a)-[r]->(b)

Like the tags on vertices, edges can also have types. To describe an edge with a specific type, use the pattern as follows:

(a)-[r:REL\_TYPE]->(b)

An edge can only have one edge type. But if we'd like to describe some data such that the edge could have a set of types, then they can all be listed in the pattern, separating them with the pipe symbol | like this:

(a)-[r:TYPE1|TYPE2]->(b)

Like vertices, the name of an edge can be omitted. For example:

(a)-[:REL\_TYPE]->(b)

#### Variable-length pattern

Rather than describing a long path using a sequence of many vertex and edge descriptions in a pattern, many edges (and the intermediate vertices) can be described by specifying a length in the edge description of a pattern. For example:

(a)-[\*2]->(b)

The following pattern describes a graph of three vertices and two edges, all in one path (a path of length 2). It is equivalent to:

(a)-[]->()-[]->(b)

The range of lengths can also be specified. Such edge patterns are called variable-length edges. For example:

(a)-[\*3..5]->(b)

The preceding example defines a path with a minimum length of 3 and a maximum length of 5.

It describes a graph of either 4 vertices and 3 edges, 5 vertices and 4 edges, or 6 vertices and 5 edges, all connected in a single path.

You may specify either the upper limit or lower limit of the length range, or neither of them, for example:

```
    (a)-[*..5]->(b) // The minimum length is 1 and the maximum length is 5.
    (a)-[*3..]->(b) // The minimum length is 3 and the maximum length is infinity.
    (a)-[*]->(b) // The minimum length is 1 and the maximum length is infinity.
```

## Assigning to path variables

As described above, a series of connected vertices and edges is called a <code>path</code>. nGQL allows paths to be named using variables. For example:

p = (a)-[\*3..5]->(b)

Users can do this in the  $\ensuremath{\operatorname{\mathsf{MATCH}}}$  statement.

Last update: December 5, 2023

### 4.1.3 Comments

This topic will describe the comments in nGQL.

L jacy version compatibility

```
• In NebulaGraph 1.x, there are four comment styles: #, --, //, /* */.
```

• Since NebulaGraph 2.x, -- cannot be used as comments.

#### Examples

#### Q Note

• In nGQL statements, the backslash  $\backslash$  in a line indicates a line break.

• If a statement starts with # or //, the statement is not executed and the error StatementEmpty is returned.

#### OpenCypher compatibility

- In nGQL, you must add a  $\$  at the end of every line, even in multi-line comments /\* \*/.
- In openCypher, there is no need to use a  $\setminus$  as a line break.

```
/* openCypher style:
The following comment
spans more than
one line */
MATCH (n:label)
RETURN n;
```

/\* nGQL style: \ The following comment \ spans more than \ one line \*/ \ MATCH (n:tag) \ RETURN n;

Last update: November 14, 2023

## 4.1.4 Identifier case sensitivity

#### Identifiers are Case-Sensitive

The following statements will not work because they refer to two different spaces, i.e. my\_space and MY\_SPACE.

```
nebula> CREATE SPACE IF NOT EXISTS my_space (vid_type=FIXED_STRING(30));
nebula> use MY_SPACE;
[ERROR (-1005)]: SpaceNotFound:
```

#### Keywords and Reserved Words are Case-Insensitive

The following statements are equivalent since show and spaces are keywords.

nebula> show spaces; nebula> SHOW SPACES; nebula> SHOW spaces; nebula> show SPACES;

#### Functions are Case-Insensitive

 $\label{eq:count} Functions are case-insensitive. \ For example, \ count() \ , \ COUNT() \ , \ and \ \ couNT() \ are \ equivalent.$ 

```
nebula> WITH [NULL, 1, 1, 2, 2] As a \
    UNWIND a AS b \
    RETURN count(b), COUNT(*), couNT(DISTINCT b);
+-----+
| count(b) | COUNT(*) | couNT(distinct b) |
+----++
| 4 | 5 | 2 |
+----+++---++
```

Last update: October 25, 2023

### 4.1.5 Keywords

Keywords in nGQL are words with particular meanings, such as CREATE and TAG in the CREATE TAG statement. Keywords that require special processing to be used as identifiers are referred to as reserved keywords, while the part of keywords that can be used directly as identifiers are called non-reserved keywords.

It is not recommended to use keywords to identify schemas. If you must use keywords as identifiers, pay attention to the following restrictions:

- To use reserved keywords or special characters as identifiers, you must enclose them with backticks (`), such as `AND`. Otherwise, a syntax error is thrown.
- To use non-reserved keywords as identifiers:
- If the identifier contains any uppercase letter, you must enclose them with backticks (`), such as `Comment`. Otherwise, the execution succeeds but the system automatically converts the identifier to all lowercase.
- If the identifier contains all lowercase letters, you do not need to enclose them with backticks (`).

Note	
Keywords are case-insensitive.	
nebula> CREATE TAG TAG(name string); [ERROR (-1004)]: SyntaxError: syntax error near `TAG'	
nebula> CREATE TAG `TAG` (name string); Execution succeeded	

nebula> CREATE TAG SPACE(name string); Execution succeeded

nebula> CREATE TAG 中文(简体 string); Execution succeeded

nebula> CREATE TAG`\$ special characters&\*+-\*/` (`q~ ! () = wer` string); Execution succeeded

#### **Reserved keywords**

ACROSS		
ADD		
ALTER		
AND		
45		
ASC		
ASCENDING		
PALANCE		
POOL		
BUCL		
DI		
CHANCE		
CONDACT		
CDEATE		
DATE		
DATETINE		
DELETE		
DESC		
DESCENDING		
DESCRIBE		
DISTINCT		
DOUBLE		
DOWNLOAD		
DRUP		
DURATION		
EDGE		
EDGES		
EXISIS		
EXPLAIN		
FALSE		
FEICH		
FIND		
FIXED_STRING		
FLUAI		
FLUSH		

FROM
GEOGRAPHY
GET
GO
GRANT
IF
IGNORE_EXISTED_INDEX
IN
INDEX
INDEXES
INGEST
INSERT
INT
INT16
TNT32
11164
INTR
INTERSECT
IS
IFT I
I OKUP
NP
MICH MICH
NO NO
NT .
DF
RP CONTRACTOR CONTRA
GADER GADER
UNDER DE
OVERWITE DVFDNPTTF
Dath Dath
Periti D
KE VOIL SET
STEP
STEPS
STOP
STRING
UBMIT
TAG
TAGS
TIME
TIMESTAMP
TO
TRUE
UNION
UNWIND
UPDATE
UPSERT
UPTO
USE
VERTEX
VERTICES
Men Men Men
WHERE
WITH
XOR
YIELD

## Non-reserved keywords

CCOUNT	
MIN	
JENT	
L .	
LSHORTESTPATHS	
IALYZER	
14	
rowIC_EDGE	
JTO	
ASIC	
IDIRECT	
)TH	
IARSET	
LEAR	
IENTS	
DLLATE	
DLLATION	
MMENT	

CONFIGS CONTAINS DATA DATA DBA DEFAULT DIVIDE DRAINER DRAINERS ELASTICSEARCH ELSE EVID END END ENDS ES\_QUERY FORCE FORMAT FULLTEXT GOD GRANTS GRAPH GROUP GROUPS GUEST HDFS HOFS HOST HOSTS HTTP HTTPS INTO IP JOB JOBS KILL KILL LEADER LIMIT LINESTRING LISTENER LOCAL MERGE META NEW NOLOOP NONE OFFSET OFFSET OFFSET OPTIONAL OUT PART PARTITION\_NUM PARTS PASSWORD PLAN POINT POLYGON PROFILE QUERIES QUERY READ READ REDUCE RENAME REPLICA\_FACTOR RESET ROLE ROLES SULES S2\_MAX\_CELLS S2\_MAX\_CELLS S2\_MAX\_CELLS SEAVICE SEAVICE SEAVICE SEAVICE SESSION SESSIONS SHORTEST SPACE STARTS STARTS STARTS STARTS STARTS STARTS STORAGE SUBGRAPH SUBGRAPH SYNC TEXT TEXT\_SEARCH THEN TOP TTL\_COL TTL\_DURATION USER USEDS USERS UUID VALUE

VALUES VARIABLES VID\_TYPE WHITELIST WRITE ZONE ZONES

Last update: November 3, 2023

#### 4.1.6 nGQL style guide

nGQL does not have strict formatting requirements, but creating nGQL statements according to an appropriate and uniform style can improve readability and avoid ambiguity. Using the same nGQL style in the same organization or project helps reduce maintenance costs and avoid problems caused by format confusion or misunderstanding. This topic will provide a style guide for writing nGQL statements.

# Empatibility

The styles of nGQL and Cypher Style Guide are different.

#### Newline

1. Start a new line to write a clause.

#### Not recommended:

GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS id;

#### Recommended:

GO FROM "player100" \ OVER follow REVERSELY \ YIELD src(edge) AS id;

2. Start a new line to write different statements in a composite statement.

#### Not recommended:

```
GO FROM "player100" OVER follow REVERSELY YIELD src(edge) AS id | GO FROM -id \ 0 OVER serve WHERE properties(^{0}.age > 20 YIELD properties(^{0}.name AS FriendOf, properties(^{0}.name AS Team;
```

#### Recommended:

```
GO FROM "player100" \

OVER follow REVERSELY \

YIELD src(edge) AS id | \

GO FROM $-.id OVER serve \

WHERE properties($^).age > 20 \

YIELD properties($^).name AS FriendOf, properties($$).name AS Team;
```

3. If the clause exceeds 80 characters, start a new line at the appropriate place.

#### Not recommended:

```
MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
WHERE (v2.player.name STARTS WITH "Y" AND v2.player.age > 35 AND v2.player.age < v.player.age) OR (v2.player.name STARTS WITH "T" AND v2.player.age < 45 AND v2.player.age > v.player.age) \
RETURN v2;
```

#### Recommended:

```
MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
WHERE (v2.player.name STARTS WITH "Y" AND v2.player.age > 35 AND v2.player.age < v.player.age) \
OR (v2.player.name STARTS WITH "T" AND v2.player.age < 45 AND v2.player.age > v.player.age \
RETURN v2;
```

#### Q Note

If needed, you can also start a new line for better understanding, even if the clause does not exceed 80 characters.

#### **Identifier naming**

In nGQL statements, characters other than keywords, punctuation marks, and blanks are all identifiers. Recommended methods to name the identifiers are as follows.

1. Use singular nouns to name tags, and use the base form of verbs or verb phrases to form Edge types.

Not recommended:

MATCH p=(v:players)-[e:are\_following]-(v2) \
RETURN nodes(p);

#### Recommended:

MATCH p=(v:player)-[e:follow]-(v2) \
RETURN nodes(p);

2. Use the snake case to name identifiers, and connect words with underscores (\_) with all the letters lowercase.

```
Not recommended:
```

```
MATCH (v:basketballTeam) \
RETURN v;
```

Recommended:

MATCH (v:basketball\_team) \
RETURN v;

3. Use uppercase keywords and lowercase variables.

#### Not recommended:

match (V:player) return V limit 5;

## Recommended:

MATCH (v:player) RETURN v LIMIT 5;

#### Pattern

1. Start a new line on the right side of the arrow indicating an edge when writing patterns.

#### Not recommended:

```
MATCH (v:player{name: "Tim Duncan", age: 42}) \
-[e:follow]->()-[e:serve]->()<--(v2) \
RETURN v, e, v2;
```

#### Recommended:

```
MATCH (v:player{name: "Tim Duncan", age: 42})-[e:follow]-> \
()-[e:serve]->()<--(v2) \
RETURN v, e, v2;
```

2. Anonymize the vertices and edges that do not need to be queried.

#### Not recommended:

```
MATCH (v:player)-[e:follow]->(v2) \
RETURN v;
```

### Recommended:

MATCH (v:player)-[:follow]->() \
RETURN v;

3. Place named vertices in front of anonymous vertices.

Not recommended:

MATCH ()-[:follow]->(v) \ RETURN v;

#### Recommended:

MATCH (v)<-[:follow]-() \ RETURN v;

#### String

The strings should be surrounded by double quotes.

#### Not recommended:

RETURN 'Hello Nebula!';

#### Recommended:

RETURN "Hello Nebula!\"123\"";

#### Q Note

When single or double quotes need to be nested in a string, use a backslash () to escape. For example:

RETURN "\"NebulaGraph is amazing,\" the user says.";

#### Statement termination

1. End the nGQL statements with an English semicolon (;).

#### Not recommended:

```
FETCH PROP ON player "player100" YIELD properties(vertex)
```

#### Recommended:

FETCH PROP ON player "player100" YIELD properties(vertex);

2. Use a pipe (|) to separate a composite statement, and end the statement with an English semicolon at the end of the last line. Using an English semicolon before a pipe will cause the statement to fail.

#### Not supported:

```
GO FROM "player100" \

OVER follow \

YTELD dst(edge) AS id; | \

GO FROM 5-.id \

OVER serve \

YIELD properties($$).name AS Team, properties($^).name AS Player;
```

#### Supported:

```
GO FROM "player100" \

OVER follow \

YTELD dst(edge) AS id | \

GO FROM 5-.id \

OVER serve \

YIELD properties($$).name AS Team, properties($^).name AS Player;
```

3. In a composite statement that contains user-defined variables, use an English semicolon to end the statements that define the variables. If you do not follow the rules to add a semicolon or use a pipe to end the composite statement, the execution will fail.

#### Not supported:

```
$var = G0 FROM "player100" \
OVER follow \
YIELD follow._dst AS id \
G0 FROM $var.id \
```

OVER serve \ YIELD \$\$.team.name AS Team, \$^.player.name AS Player;

## Not supported:

\$var = G0 FROM "player100" \ OVER follow \ YIELD follow..dst AS id | \ G0 FROM \$var.id \ OVER serve \ YIELD \$\$.team.name AS Team, \$^.player.name AS Player;

#### Supported:

\$var = G0 FROM "player100" \
OVER follow \
YIELD follow\_\_dst AS id; \
G0 FROM \$var.id \
OVER serve \
YIELD \$\$.team.name AS Team, \$^.player.name AS Player;

Last update: October 25, 2023

## 4.2 Data types

### 4.2.1 Numeric types

nGQL supports both integer and floating-point number.

#### Integer

Signed 64-bit integer (INT64), 32-bit integer (INT32), 16-bit integer (INT16), and 8-bit integer (INT8) are supported.

Туре	Declared keywords	Range
INT64	INT64 or INT	$-9,223,372,036,854,775,808 \sim 9,223,372,036,854,775,807$
INT32	INT32	-2,147,483,648 ~ 2,147,483,647
INT16	INT16	-32,768 ~ 32,767
INT8	INT8	-128 ~ 127

#### Floating-point number

Both single-precision floating-point format (FLOAT) and double-precision floating-point format (DOUBLE) are supported.

Туре	Declared keywords	Range	Precision
FLOAT	FLOAT	3.4E +/- 38	6~7 bits
DOUBLE	DOUBLE	1.7E +/- 308	15~16 bits

Scientific notation is also supported, such as 1e2 , 1.1e2 , .3e4 , 1.e4 , and  $\,$  -1234E-10 .

#### Q Note

The data type of DECIMAL in MySQL is not supported.

#### Reading and writing of data values

When writing and reading different types of data, nGQL complies with the following rules:

Data type	Set as VID	Set as property	Resulted data type
INT64	Supported	Supported	INT64
INT32	Not supported	Supported	INT64
INT16	Not supported	Supported	INT64
INT8	Not supported	Supported	INT64
FLOAT	Not supported	Supported	DOUBLE
DOUBLE	Not supported	Supported	DOUBLE

For example, nGQL does not support setting VID as INT8, but supports setting a certain property type of TAG or Edge type as INT8. When using the nGQL statement to read the property of INT8, the resulted type is INT64.

- Multiple formats are supported:
- Decimal, such as 123456.
- Hexadecimal, such as 0x1e240.
- Octal, such as 0361100.

However, NebulaGraph will parse the written non-decimal value into a decimal value and save it. The value read is decimal.

For example, the type of the property score is INT. The value of 0xb is assigned to it through the INSERT statement. If querying the property value with statements such as FETCH, you will get the result 11, which is the decimal result of the hexadecimal 0xb.

• Round a FLOAT/DOUBLE value when inserting it to an INT column.

Last update: October 25, 2023

## 4.2.2 Boolean

A boolean data type is declared with the bool keyword and can only take the values true or false.

 $n GQL \ supports \ using \ boolean \ in the following \ ways:$ 

- Define the data type of the property value as a boolean.
- Use boolean as judgment conditions in the WHERE clause.

Last update: October 25, 2023

#### 4.2.3 String

Fixed-length strings and variable-length strings are supported.

#### Declaration and literal representation

The string type is declared with the keywords of:

- STRING : Variable-length strings.
- FIXED\_STRING(<length>) : Fixed-length strings. <length> is the length of the string, such as FIXED\_STRING(32) .

A string type is used to store a sequence of characters (text). The literal constant is a sequence of characters of any length surrounded by double or single quotes. For example, "Hello, Cooper" or 'Hello, Cooper'.

#### String reading and writing

Nebula Graph supports using string types in the following ways:

- Define the data type of VID as a fixed-length string.
- Set the variable-length string as the Schema name, including the names of the graph space, tag, edge type, and property.
- Define the data type of the property as a fixed-length or variable-length string.

### For example:

• Define the data type of the property as a fixed-length string

nebula> CREATE TAG IF NOT EXISTS t1 (p1 FIXED\_STRING(10));

• Define the data type of the property as a variable-length string

nebula> CREATE TAG IF NOT EXISTS t2 (p2 STRING);

When the fixed-length string you try to write exceeds the length limit:

- If the fixed-length string is a property, the writing will succeed, and NebulaGraph will truncate the string and only store the part that meets the length limit.
- If the fixed-length string is a VID, the writing will fail and NebulaGraph will return an error.

#### **Escape Characters**

In strings, the backslash (\) serves as an escape character used to denote special characters.

For example, to include a double quote (") within a string, you cannot directly write "Hello "world"" as it leads to a syntax error. Instead, use the backslash (\) to escape the double quote, such as "Hello \"world\"".

```
nebula> RETURN "Hello \"world\""
+-----+
| "Hello "world"" |
+-----+
| "Hello "world"" |
+-----+
```

The backslash itself needs to be escaped as it's a special character. For example, to include a backslash in a string, you need to write "Hello \\ world".

```
nebula> RETURN "Hello \\ world"
+-----+
| "Hello \ world" |
+-----+
| "Hello \ world" |
+-----+
```

For more examples of escape characters, see Escape character examples.

#### OpenCypher compatibility

There are some tiny differences between openCypher and Cypher, as well as nGQL. The following is what openCypher requires. Single quotes cannot be converted to double quotes.

```
# File: Literals.feature
Feature: Literals
Background:
    Given any graph
Scenario: Return a single-quoted string
    When executing query:
    """
    RETURN '' AS Literal
    """
    Then the result should be, in any order:
        | literal |
        | '' | # Note: it should return single-quotes as openCypher required.
    And no side effects
```

While Cypher accepts both single quotes and double quotes as the return results. nGQL follows the Cypher way.

```
nebula > YIELD '' AS quote1, "" AS quote2, "'" AS quote3, '"' AS quote4
+-----+
| quote1 | quote2 | quote3 | quote4 |
+-----+
| "" | "" | """ |
+----++
```

Last update: December 28, 2023

#### 4.2.4 Date and time types

This topic will describe the DATE, TIME, DATETIME, TIMESTAMP, and DURATION types.

#### Precautions

• While inserting time-type property values with DATE, TIME, and DATETIME, NebulaGraph transforms them to a UTC time according to the timezone specified with the timezone\_name parameter in the configuration files.

Note

To change the timezone, modify the timezone\_name value in the configuration files of all NebulaGraph services.

- date(), time(), and datetime() can convert a time-type property with a specified timezone. For example, datetime("2017-03-04 22:30:40.003000+08:00") or datetime("2017-03-04T22:30:40.003000[Asia/Shanghai]").
- date(), time(), datetime(), and timestamp() all accept empty parameters to return the current date, time, and datetime.
- date(), time(), and datetime() all accept the property name to return a specific property value of itself. For example, date().month returns the current month, while time("02:59:40").minute returns the minutes of the importing time.
- For time operations it is recommended to use duration() to calculate the offset of the moment. Addition and subtraction of date() and date(), timestamp() and timestamp() are also supported.
- When setting the year of the time as a negative number, you need to use Map type data.

#### **OpenCypher Compatibility**

In nGQL:

- Year, month, day, hour, minute, second, millisecond, and microsecond are supported, while the nanosecond is not supported.
- localdatetime() is not supported.
- Most string time formats are not supported. The exceptions are YYYY-MM-DDThh:mm:ss and YYYY-MM-DD hh:mm:ss.
- The single-digit string time format is supported. For example, time("1:1:1").

#### DATE

The DATE type is used for values with a date part but no time part. Nebula Graph retrieves and displays DATE values in the YYYY-MM-DD format. The supported range is -32768-01-01 to 32767-12-31.

The properties of date() include year, month, and day. date() supports the input of YYYYY, YYYYY-MM or YYYYY-MM-DD, and defaults to 01 for an untyped month or day.

| 1 |

#### TIME

The TIME type is used for values with a time part but no date part. Nebula Graph retrieves and displays TIME values in hh:mm:ss.msmsususus format. The supported range is 00:00:00.000000 to 23:59:59.9999999.

The properties of time() include hour, minute, and second.

#### DATETIME

The DATETIME type is used for values that contain both date and time parts. Nebula Graph retrieves and displays DATETIME values in YYYY-MM-DDThh:mm:ss.msmsmsususus format. The supported range is -32768-01-01T00:00:00.000000 to 32767-12-31T23:59:59.9999999.

- The properties of datetime() include year, month, day, hour, minute, and second.
- datetime() can convert TIMESTAMP to DATETIME. The value range of TIMESTAMP is 0~9223372036.
- datetime() supports an int argument. The int argument specifies a timestamp.

```
# To get the current date and time.
nebula> RETURN datetime();
+-----+
| datetime() |
+-----+
# To get the current hour.
nebula> RETURN datetime().hour;
+-----+
| datetime().hour |
+-----+
| datetime().hour |
+-----+
| datetime().hour |
+-----+
# To get date time from a given timestamp.
nebula> RETURN datetime(timestamp(1625469277));
+-----+
| datetime(timestamp(1625469277)) |
+-----+
| 2021-07-05T07:14:37.00000 |
+-----+
| datetime(1625469277) |
+-----+
| datetime(1625469277) |
+-----+
| datetime(1625469277) |
+-----+
| datetime(1625469277) |
+-----+
| 2021-07-05T07:14:37.00000 |
+-----+
| 2021-07-05T07:14:37.00000 |
+-----+
```

#### TIMESTAMP

The TIMESTAMP data type is used for values that contain both date and time parts. It has a range of 1970-01-01T00:00:01 UTC to 2262-04-11T23:47:16 UTC.

TIMESTAMP has the following features:

- Stored and displayed in the form of a timestamp, such as 1615974839, which means 2021-03-17T17:53:59.
- Supported TIMESTAMP querying methods: timestamp and timestamp() function.
- Supported TIMESTAMP inserting methods: timestamp, timestamp() function, and now() function.
- timestamp() function accepts empty arguments to get the current timestamp. It can pass an integer arguments to identify the integer as a timestamp and the range of passed integer is: 0~9223372036 °
- timestamp() function can convert DATETIME to TIMESTAMP, and the data type of DATETIME should be a string.
- The underlying storage data type is int64.

```
# To get the current timestamp.
nebula> RETURN timestamp();
+-----+
    timestamp() |
+-----+
    timestamp() |
+----+
    timestamp(----+
    timestamp(-----+
    to get a timestamp from given date and time.
nebula> RETURN timestamp("2022-01-05T06:18:43");
+------+
    timestamp("2022-01-05T06:18:43") |
+-----+
    timestamp("2022-01-05T06:18:43") |
+-----+
# To get a timestamp using datetime().
nebula> RETURN timestamp(datetime("2022-08-29T07:53:10.939000"));
+------+
```

#### Q Note

The date and time format string passed into timestamp() cannot include any millisecond and microsecond, but the date and time format string passed into timestamp(datetime()) can include a millisecond and a microsecond.

#### DURATION

The DURATION data type is used to indicate a period of time. Map data that are freely combined by years, months, days, hours, minutes, and seconds indicates the DURATION.

DURATION has the following features:

- Creating indexes for DURATION is not supported.
- DURATION can be used to calculate the specified time.

#### Examples

1. Create a tag named date1 with three properties: DATE, TIME, and DATETIME.

nebula> CREATE TAG IF NOT EXISTS date1(p1 date, p2 time, p3 datetime);

#### 2. Insert a vertex named test1.

nebula> INSERT VERTEX date1(p1, p2, p3) VALUES "test1":(date("2021-03-17"), time("17:53:59"), datetime("2017-03-04T22:30:40.003000[Asia/Shanghai]"));

3. Query whether the value of property p1 on the test1 tag is 2021-03-17.

```
nebula> MATCH (v:date1) RETURN v.date1.p1 == date("2021-03-17");
+----+
| (v.date1.p1==date("2021-03-17")) |
+----+
| true |
+----+
```

4. Return the content of the property p1 on test1.

```
nebula> CREATE TAG INDEX IF NOT EXISTS date1_index ON date1(p1);
nebula> REBUILD TAG INDEX date1_index;
nebula> MATCH (v:date1) RETURN v.date1.p1;
+------+
| v.date1.p1.month |
+-----+
| 3 |
+-----+
```

5. Search for vertices with p3 property values less than 2023-01-01T00:00:00.000000, and return the p3 values.

```
nebula> MATCH (v:date1) \
WHERE v.date1.p3 < datetime("2023-01-01T00:00:00.000000") \
RETURN v.date1.p3;
+-----+
| v.date1.p3 |
+------+
| 2017-03-04T14:30:40.003000 |
```

#### 6. Create a tag named school with the property of TIMESTAMP.

nebula> CREATE TAG IF NOT EXISTS school(name string , found\_time timestamp);

#### 7. Insert a vertex named DUT with a found-time timestamp of "1988-03-01T08:00:00".

# Insert as a timestamp. The corresponding timestamp of 1988-03-01T08:00:00 is 573177600, or 573206400 UTC. nebula> INSERT VERTEX school(name, found\_time) VALUES "DUT"; ("DUT", 573206400); # Insert in the form of date and time.

nebula> INSERT VERTEX school(name, found\_time) VALUES "DUT":("DUT", timestamp("1988-03-01T08:00:00"));

#### 8. Insert a vertex named dut and store time with now() or timestamp() functions.

# Use now() function to store time nebula> INSERT VERTEX school(name, found\_time) VALUES "dut":("dut", now()); # Use timestamp() function to store time

nebula> INSERT VERTEX school(name, found\_time) VALUES "dut":("dut", timestamp());

#### You can also use WITH statement to set a specific date and time, or to perform calculations. For example:

nebula> WITH time({hour: 12, minute: 31, second: 14, millisecond:111, microsecond: 222}) AS d RETURN d; +-----+ | d | | +-----+ | 12:31:14.111222 | +-----+ | 12:31:14.111222 | +-----+ | (x+1) | +-----+ | 1984-10-12 | +-----+ | 1984-10-12 | +-----+ | 1984-10-12 | +-----+ | sum | diff | +-----+ | 1996-10-29 | 1972-09-23 | +-----+

Last update: October 25, 2023

## 4.2.5 NULL

You can set the properties for vertices or edges to NULL. Also, you can set the NOT NULL constraint to make sure that the property values are NOT NULL. If not specified, the property is set to NULL by default.

#### Logical operations with NULL

Here is the truth table for AND ,  $\mathsf{OR}$  ,  $\mathsf{XOR}$  , and  $\mathsf{NOT}$  .

а	b	a AND b	a OR b	a XOR b	NOT a
false	false	false	false	false	true
false	null	false	null	null	true
false	true	false	true	true	true
true	false	false	true	true	false
true	null	null	true	null	false
true	true	true	true	false	false
null	false	false	null	null	null
null	null	null	null	null	null
null	true	null	true	null	null

#### OpenCypher compatibility

The comparisons and operations about NULL are different from openCypher. There may be changes later.

COMPARISONS WITH NULL

The comparison operations with NULL are incompatible with openCypher.

OPERATIONS AND RETURN WITH NULL

The NULL operations and RETURN with NULL are incompatible with openCypher.

#### Examples

USE NOT NULL

Create a tag named player. Specify the property name as NOT NULL.

nebula> CREATE TAG IF NOT EXISTS player(name string NOT NULL, age int);

Use SHOW to create tag statements. The property name is NOT NULL. The property age is NULL by default.

```
nebula> SHOW CREATE TAG player;
+----+
| Tag | Create Tag +
+---+
| "student" | "CREATE TAG `player` ( | |
| `name` string NOT NULL, |
| `age` int64 NULL
| | `age` int64 NULL
| | ) ttl_duration = 0, ttl_col = """
+----+
```

Insert the vertex Kobe. The property age can be NULL.

```
nebula> INSERT VERTEX player(name, age) VALUES "Kobe":("Kobe",null);
```

USE NOT NULL AND SET THE DEFAULT

Create a tag named  $\, {\tt player}$  . Specify the property  $\, {\tt age} \,$  as  $\, {\tt NOT} \, {\tt NULL}$  . The default value is  $\, {\tt 18}$  .

nebula> CREATE TAG IF NOT EXISTS player(name string, age int NOT NULL DEFAULT 18);

## Insert the vertex Kobe. Specify the property name only.

nebula> INSERT VERTEX player(name) VALUES "Kobe":("Kobe");

Query the vertex Kobe. The property age is 18 by default.

Last update: October 25, 2023
# 4.2.6 Lists

The list is a composite data type. A list is a sequence of values. Individual elements in a list can be accessed by their positions.

A list starts with a left square bracket [ and ends with a right square bracket ]. A list contains zero, one, or more expressions. List elements are separated from each other with commas ( , ). Whitespace around elements is ignored in the list, thus line breaks, tab stops, and blanks can be used for formatting.

#### OpenCypher compatibility

A composite data type (i.e. set, map, and list) CANNOT be stored as properties of vertices or edges.

#### List operations

You can use the preset list function to operate the list, or use the index to filter the elements in the list.

INDEX SYNTAX

[M] [M..N] [M..] [..N]

The index of nGQL supports queries from front to back, starting from 0. 0 means the first element, 1 means the second element, and so on. It also supports queries from back to front, starting from -1. -1 means the last element, -2 means the penultimate element, and so on.

- [M]: represents the element whose index is M.
- [M.N]: represents the elements whose indexes are greater or equal to M but smaller than N. Return empty when N is 0.
- [M..]: represents the elements whose indexes are greater or equal to M.
- [..N]: represents the elements whose indexes are smaller than N. Return empty when N is 0.

#### Q Note

• Return empty if the index is out of bounds, while return normally if the index is within the bound.

• Return empty if  $M \ge N$ .

• When querying a single element, if M is null, return BAD\_TYPE. When conducting a range query, if M or N is null, return null.

#### Examples

```
# The following query returns the list [1,2,3].
nebula> RETURN list[1, 2, 3] AS a;
+----+
| a |
+-----+
| [1, 2, 3] |
+-----+
```

# The following query returns the element whose index is 3 in the list [1,2,3,4,5]. In a list, the index starts from 0, and thus the return element is 4. nebula> RETURN range(1,5)[3];

+----+ | range(1,5)[3] | +----+ | 4 |

# The following query returns the element whose index is -2 in the list [1,2,3,4,5]. The index of the last element in a list is -1, and thus the return element is 4. nebula> RETURN range(1,5)[-2];

+	-+
range(1,5)[-(2)]	
+	-+
4	
+	-+

# The following query returns the elements whose indexes are from 0 to 3 (not including 3) in the list [1,2,3,4,5].
nebula> RETURN range(1,5)[0..3];

<pre>Interactions in the element show inters are proter that 2 is the Ust [1,2,3,4,5]; interface THER reget(5, [3, 4] 4; i</pre>	
<pre>shits Billing appry returns the intents show indexs are grater that 2 in the list [1,2,3,4,5]; whether BBBS requires the iteration where indexs are grater that 3. method with integrates (1, 2, 3, 4, 4) &amp; 4 &amp; 1 [1, 2, 3] ************************************</pre>	++   range(1,5)[03]   ++   [1, 2, 3]   ++
<pre>A The following party returns the elsents whose indexes are waller than 3. elsent with iter[1, 2, 3, 4, 4] AS 1, EEGM 4 [1, 2, 3, 4, 5] Formation of the elsents whose indexes are practer than 2 in the list [1, 2, 1, 4, 5], calculate them respectively, and returns these endules KHM 1 (eff), 2, 3, 4, 5] = 1 = 13 [A 3; Formation of the elsents the elsents for the first to the penultinets (inclusive) in the list [1, 2, 1], [1, 1] Formation of the elsent the elsents for the first to the penultinets (inclusive) in the list [1, 2, 1], [1, 1] Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 1] AS 1; Formation of the list [1, 2, 1], [1, 2] AS 1; Formation of the list [1, 2, 1], [1, 2] AS 1; Formation of the list [1, 2, 1], [1, 2] AS 1; Formation of the list [1, 2, 1], [1, 2] AS 1; Formation of the list [1, 2, 3], [1, 2] AS 1; Formation of the list [1, 2, 3], [1, 2] AS 1; Formation of the list [1, 2, 3], [1, 2] AS 2; Formation of the list [1, 2, 3], [1, 2] AS 1; Formation of the list [1, 2, 3], [1, 2] AS 2; Formation of the list [1, 2] AS 2; Formation of the list [1, 2, 3], [1, 2] AS 2; Formation of the list [1, 2] AS 2</pre>	<pre># The following query returns the elements whose indexes are greater than 2 in the list [1,2,3,4,5]. nebula&gt; RETURN range(1,5)[3] AS a; +++   a   +++   [4, 5]  </pre>
The second seco	# The following query returns the elements whose indexes are smaller than 3.
<pre>http://www.intermediate/content/operation/c</pre>	RETURN a[] AS r; ++
<pre>h = Definition gargery fitters the streamts does inforces are gratter than 2 in the list [1,2,3,4,5], calculate ther respectively, and returns then member EREME (in range(1,5), MEE (= 2   n + 10] AS a;</pre>	++   [1, 2, 3]   ++
<pre>[13, 14, 15]] * The following query returns empty because the index is out of bound. It will return normally when the index is within the bound. * The following query returns empty because the is a [0.0]. * The following query returns empty because the is a [0.0]. * The following query returns empty because of M ≥ M. * The following query returns empty because of M</pre>	<pre># The following query filters the elements whose indexes are greater than 2 in the list [1,2,3,4,5], calculate them respectively, and returns them. nebula&gt; RETURN [n IN range(1,5) WHERE n &gt; 2   n + 10] AS a; ++   a   ++</pre>
<pre># The following query returns the elements from the first to the penultinate (inclusive) in the list [1, 2, 3]. # The following query returns the elements from the first (exclusive) to the third backward in the list [1, 2, 3, 4, 5]. # The following query returns the elements from the first (exclusive) to the third backward in the list [1, 2, 3, 4, 5]. # The following query returns the elements from the first (exclusive) to the third backward in the list [1, 2, 3, 4, 5]. # The following query returns the elements the elements whose indexes are 1 and 2. medule: Star = THED 128 f, 3.8 t; THED 118 f, 3, 3[Star.f., 3ow.f.] AS s; # The following query returns empty because the index is out of bound. It will return normally when the index is within the bound. # The following query returns empty because there is a out of bound. It will return normally when the index is within the bound. # The following query returns empty because there is a [0, 0]. # The following query returns empty because there is a [0, 0]. # The following query returns empty because there is a [0, 0]. # The following query returns empty because there is a [0, 0]. # The following query returns empty because there is a [0, 0]. # The following query returns empty because there is a [0, 0]. # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3, 4, 5] [2, 1] AS s; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3, 4, 5] [2, 1] AS s; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3] A s j; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3, 4, 5] [2, 1] AS s; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3] A s j; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3] A s j; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3] A s j; # The following query returns empty because of M ≥ N. # Element ist[1, 2, 3] A s j; # The following query returns em</pre>	[13, 14, 15]   ++
<pre>i =</pre>	<pre># The following query returns the elements from the first to the penultimate (inclusive) in the list [1, 2, 3]. nebula&gt; YIELD list[1, 2, 3][01] AS a; ++</pre>
<pre># The following query returns the elements from the first (exclusive) to the third backward in the list [1, 2, 3, 4, 5]. nedular YEED List[1, 2, 3, 4, 5]:-3, -2] A5 a; ************************************</pre>	a   ++   [1, 2]
<pre>  a   ++   3, 4] ++ # The following query sets the variables and returns the elements whose indexes are 1 and 2. mebular Serr FileD 1 AS f; 3 AS t; \ VIED List[1, 2, 3](Sur.f. Sur.t) AS a; ++   a   ++ # The following query returns empty because the index is out of bound. It will return normally when the index is within the bound. nebular SETURU List[1, 2, 3, 4, 5] (0.10) AS a; ++   a   ++ # The following query returns empty because there is a [0.0]. nebular SETURU List[1, 2, 3] (-55] AS a; ++   a   ++ # The following query returns empty because there is a [0.0]. nebular SETURU List[1, 2, 3, 4, 5] [00] AS a; ++ # The following query returns empty because of M ≥ N. mebular SETURU List[1, 2, 3, 4, 5] [01] AS a; ++ # The following query returns empty because of M ≥ N. mebular SETURU List[1, 2, 3, 4, 5] [01] AS a; ++ # Amen conduct a range query, if "N' or "N' is null, return 'null'. mebular SETURU A list[1, 2, 3, 4, 5] [31] AS a; ++ # Amen conduct a range query, if "N' or "N' is null, return 'null'. mebular SETURU a list[1, 2, 3] A   # Men conduct a range query, if "N' or "N' is null, return 'null'. mebular SETURU a list[1, 2, 3] A   # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query, if "N' or "N' is null, return 'null'. # Men conduct a range query the seture due due due due due due due due due du</pre>	<pre># The following query returns the elements from the first (exclusive) to the third backward in the list [1, 2, 3, 4, 5]. nebula&gt; YIELD list[1, 2, 3, 4, 5][-31] AS a; ++</pre>
<pre># The following query sets the variables and returns the elements whose indexes are 1 and 2. metulas Syar = TEED 1A S f, 3 AS t; \</pre>	a   ++   [3, 4]
<pre>a</pre>	# The following query sets the variables and returns the elements whose indexes are 1 and 2. nebula> \$var = YIELD 1 AS f, 3 AS t; \ YIELD list[1, 2, 3][\$var.f\$var.t] AS a;
<pre>[ [2, 3] ++ # The following query returns empty because the index is out of bound. It will return normally when the index is within the bound. nebula* RETURN [13, 2, 3, 4, 5] [010] AS a; ++   a</pre>	++  a   ++
<pre># The TotComing query returns empty declase the index is due to bound. It will return homacity when the index is writing the bound. # "</pre>	<pre>[[2, 3]   ++ # The following quary returns empty because the index is out of bound. It will return normally when the index is within the bound.</pre>
	<pre># The following query recurs empty because the findex is out of bound. It will recurs normally when the fidex is written the bound. nebula&gt; RETURN list[1, 2, 3, 4, 5] [010] AS a; ++   a  </pre>
<pre>nebula&gt; RETURN List[1, 2, 3] [-55] AS a; ++   a</pre>	+   [1, 2, 3, 4, 5]   ++
<pre>++   [1, 2, 3] ++ # The following query returns empty because there is a [00]. nebula&gt; RETURN list[1, 2, 3, 4, 5] [00] AS a; ++   a   ++   []   ++ # The following query returns empty because of M ≥ N. nebula&gt; RETURN list[1, 2, 3, 4, 5] [31] AS a; ++   a   ++   a   ++ # When conduct a range query, if 'M' or 'N' is null, return 'null'. nebula&gt; RETURN a[0null] as r; ++   r     </pre>	nebula> RETURN list[1, 2, 3] [-55] AS a; ++   a
<pre># The following query returns empty because there is a [00]. nebula&gt; RETURN list[1, 2, 3, 4, 5] [00] AS a; ++   a  +   []   +++ # The following query returns empty because of M ≥ N. nebula&gt; RETURN list[1, 2, 3, 4, 5] [31] AS a; ++   a   +++   a   +++   []   +++ # When conduct a range query, if 'M' or 'N' is null, return 'null'. nebula&gt; WITH list[1,2,3] AS a \</pre>	++   [1, 2, 3]   ++
<pre>  a   ++ ++ # The following query returns empty because of M ≥ N. nebula&gt; RETURN list[1, 2, 3, 4, 5] [31] AS a; ++   a   ++  + # When conduct a range query, if 'M' or 'N' is null, return 'null'. nebula&gt; WITH list[1,2,3] AS a \ RETURN a[0null] as r; ++   r   ++</pre>	# The following query returns empty because there is a [00]. nebula> RETURN list[1, 2, 3, 4, 5] [00] AS a;
<pre># The following query returns empty because of M ≥ N. mebula&gt; RETURN list[1, 2, 3, 4, 5] [31] AS a; ++   a   ++ # When conduct a range query, if 'M' or 'N' is null, return `null`. nebula&gt; WITH list[1,2,3] AS a \</pre>	a   ++   []   ++
<pre>  a   ++   []   ++ # When conduct a range query, if `M` or `N` is null, return `null`. nebula&gt; WITH (ist[1,2,3] AS a \</pre>	# The following query returns empty because of $M \ge N$ . nebula> RETURN list[1, 2, 3, 4, 5] [31] AS a; ++
<pre># When conduct a range query, if `M` or `N` is null, return `null`. nebula&gt; WITH (ist[1,2,3] AS a \</pre>	a   ++   []
	# When conduct a range query, if `M` or `N` is null, return `null`. nebula> WITH list[1,2,3] AS a \ PETIDN alo_mull as r:
	kitom alo.mutij as i,       ++         r       ++         NIII

++
# The following query calculates the elements in the list [1,2,3,4,5] respectively and returns them without the list head. nebula> RETURN tail([n IN range(1, 5)   2 * n - 10]) AS a;
++   a   
[-6, -4, -2, 0]   ++
<pre># The following query takes the elements in the list [1,2,3] as true and return. nebula&gt; RETURN [n IN range(1, 3) WHERE true   n] AS r; ++   r   ++   [1, 2, 3]</pre>
<pre># Ihe following query returns the length of the list [1,2,3]. nebula&gt; RETURN size(list[1,2,3]); ++</pre>
size(list[1,2,3])   ++
3   ++
# The following query calculates the elements in the list [92,90] and runs a conditional judgment in a where clause. nebula> 60 FROM "player100" OVER follow WHERE properties(edge).degree NOT IN [x IN [92, 90]   x + \$\$.player.age] \ YIELD dst(edge) AS id, properties(edge).degree AS degree;
id   degree   ++
"player101"   95     "player102"   90   ++
<pre># The following query takes the query result of the MATCH statement as the elements in a list. Then it calculates and returns them. nebula&gt; MATCH p = (n:player{name:"Tim Duncan"})-[:follow]-&gt;(m) \ RETURN [n IN nodes(p)   n.player.age + 100] AS r;</pre>
++   r  +
[142, 136]     [142, 141]   +

# OpenCypher compatibility

• In openCypher, return null when querying a single out-of-bound element. However, in nGQL, return OUT\_OF\_RANGE when querying a single out-of-bound element.



• A composite data type (i.e., set, map, and list) CAN NOT be stored as properties for vertices or edges.

It is recommended to modify the graph modeling method. The composite data type should be modeled as an adjacent edge of a vertex, rather than its property. Each adjacent edge can be dynamically added or deleted. The rank values of the adjacent edges can be used for sequencing.

• Patterns are not supported in the list. For example, [(src)-[]->(m) | m.name].

```
Last update: October 25, 2023
```

# 4.2.7 Sets

The set is a composite data type. A set is a set of values. Unlike a List, values in a set are unordered and each value must be unique.

A set starts with a left curly bracket { and ends with a right curly bracket }. A set contains zero, one, or more expressions. Set elements are separated from each other with commas (,). Whitespace around elements is ignored in the set, thus line breaks, tab stops, and blanks can be used for formatting.

#### OpenCypher compatibility

- A composite data type (i.e. set, map, and list) CANNOT be stored as properties of vertices or edges.
- A set is not a data type in openCypher, but in nGQL, users can use the set.

#### Examples

```
# The following query returns the set \{1,2,3\}. nebula> RETURN set{1, 2, 3} AS a;
| a
| {3, 2, 1} |
# The following query returns the set \{1,2\}, Because the set does not allow repeating elements, and the order is unordered.
nebula> RETURN set{1, 2, 1} AS a;
| a
| {2, 1} |
# The following query checks whether the set has the specified element 1. nebula> RETURN 1 IN set{1, 2} AS a;
| a
+----
| true |
# The following query counts the number of elements in the set.
nebula> YIELD size(set{1, 2, 1}) AS a;
+----
| a |
 +---+
| 2 |
a
 {36, "Tony Parker"}
{41, "Manu Ginobili"}
```

# 4.2.8 Maps

The map is a composite data type. Maps are unordered collections of key-value pairs. In maps, the key is a string. The value can have any data type. You can get the map element by using map['key'].

A map starts with a left curly bracket { and ends with a right curly bracket }. A map contains zero, one, or more key-value pairs. Map elements are separated from each other with commas ( , ). Whitespace around elements is ignored in the map, thus line breaks, tab stops, and blanks can be used for formatting.

#### OpenCypher compatibility

- A composite data type (i.e. set, map, and list) **CANNOT** be stored as properties of vertices or edges.
- Map projection is not supported.

#### Examples

<pre># The following query returns the simple map. nebula&gt; YIELD map{key1: 'Value1', Key2: 'Value2'} as a;</pre>
a
+
++
<pre># The following query returns the list type map. nebula&gt; VIELD map[listKey: [{inner: 'Wap1'}, {inner: 'Map2'}]} as a; ++</pre>
a
<pre># The following query returns the hybrid type map. nebula&gt; RETURN map{a: LIST[1,2], b: SET{1,2,1}, c: "hee"} as a;</pre>
тт   а
++   {a: [1, 2], b: {2, 1}, c: "hee"}
++
# The following query returns the specified element in a map. nebula> RETURN map{a: LIST[1,2], b: SET{1,2,1}, c: "hee"}["b"] AS b;
++   b
++   {2 1}
(=) ->   ++
# The following query checks whether the map has the specified key, not support checks whether the map has the specified value yet. nebula> RETURN "a" IN MAP{a:1, b:2} AS a;
++   a
1 - 1 ++
++

# 4.2.9 Type Conversion/Type coercions

Converting an expression of a given type to another type is known as type conversion.

NebulaGraph supports converting expressions explicit to other types. For details, see Type conversion functions.

## Examples

nebula> UNWIND [true, false, 'true', 'false', NULL] AS b \ RETURN toBoolean(b) AS b;
++
b
++
true
false
true
false
NULL
++
<pre>nebula&gt; RETURN toFloat(1), toFloat('1.3'), toFloat('1e3'), toFloat('not a number');</pre>
++
toFloat(1)   toFloat("1.3")   toFloat("1e3")   toFloat("not a number")
++
1.0   1.3   1000.0  NULL
++

# 4.2.10 Geography

Geography is a data type composed of latitude and longitude that represents geospatial information. NebulaGraph currently supports Point, LineString, and Polygon in Simple Features and some functions in SQL-MM 3, such as part of the core geo parsing, construction, formatting, conversion, predicates, and dimensions.

## Type description

A point is the basic data type of geography, which is determined by a latitude and a longitude. For example, "POINT(3 8)" means that the longitude is 3° and the latitude is 8°. Multiple points can form a linestring or a polygon.

#### Q Note

You cannot directly insert geographic data of the following types, such as INSERT VERTEX any\_shape(geo) VALUES "1":("POINT(1 1)") . Instead, you need to use a geography function to specify the data type before inserting, such as INSERT VERTEX any\_shape(geo) VALUES "1": (ST\_GeogFromText("POINT(1 1)")); .

Shape Example		Description	
Point	"POINT(3 8)"	Specifies the data type as a point.	
LineString	"LINESTRING(3 8, 4.7 73.23)"	Specifies the data type as a linestring.	
Polygon	"POLYGON((0 1, 1 2, 2 3, 0 1))"	Specifies the data type as a polygon.	

#### Examples

//Create a Tag to allow storing any geography data type. nebula> CREATE TAG IF NOT EXISTS any_shape(geo geography);
//Create a Tag to allow storing a point only. nebula> CREATE TAG IF NOT EXISTS only_point(geo geography(point));
//Create a Tag to allow storing a linestring only. nebula> CREATE TAG IF NOT EXISTS only_linestring(geo geography(linestring));
//Create a Tag to allow storing a polygon only. nebula> CREATE TAG IF NOT EXISTS only_polygon(geo geography(polygon));
//Create an Edge type to allow storing any geography data type. nebula> CREATE EDGE IF NOT EXISTS any_shape_edge(geo geography);
<pre>//Create a vertex to store the geography of a polygon. nebula&gt; INSERT VERTEX any_shape(geo) VALUES "103":(ST_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))"));</pre>
<pre>//Create an edge to store the geography of a polygon. nebula&gt; INSERT EDGE any_shape_edge(geo) VALUES "201"-&gt;"302":(ST_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))"));</pre>
//Query the geography of Vertex 103. nebula> FETCH PROP ON any_shape "103" YIELD ST_ASText(any_shape.geo);
ST_ASText(any_shape.geo)
++   "POLYGON((0 1, 1 2, 2 3, 0 1))"   ++
//Query the geography of the edge which traverses from Vertex 201 to Vertex 302. nebula> FETCH PROP ON any_shape_edge "201"->"302" YIELD ST_ASText(any_shape_edge.geo);
++   "POLYGON((0 1, 1 2, 2 3, 0 1))"   ++
<pre>//Create an index for the geography of the Tag any_shape and run LOOKUP. nebula&gt; CREATE TAG INDEX IF NOT EXISTS any_shape_geo_index ON any_shape(geo); nebula&gt; REBUILD TAG INDEX any_shape_geo_index; nebula&gt; LOOKUP ON any_shape YIELD ST_ASText(any_shape.geo);</pre>
ST_ASText(any_shape.geo)
"POLYGON((0 1, 1 2, 2 3, 0 1))"

When creating an index for geography properties, you can specify the parameters for the index.

Parameter	Default value	Description
s2_max_level	30	The maximum level of S2 cell used in the covering. Allowed values: $1 \sim 30$ . Setting it to less than the default means that NebulaGraph will be forced to generate coverings using larger cells.
s2_max_cells	8	The maximum number of S2 cells used in the covering. Provides a limit on how much work is done exploring the possible coverings. Allowed values: $1 \sim 30$ . You may want to use higher values for odd-shaped regions such as skinny rectangles.

#### Q Note

Specifying the above two parameters does not affect the Point type of property. The s2\_max\_level value of the Point type is forced to be 30.

nebula> CREATE TAG INDEX IF NOT EXISTS any\_shape\_geo\_index ON any\_shape(geo) with (s2\_max\_level=30, s2\_max\_cells=8);

For more index information, see Index overview.

# 4.3 Operators

# 4.3.1 Comparison operators

NebulaGraph supports the following comparison operators.

Name	Name Description	
==	Equal operator	
!=, ⇔	Not equal operator	
>	Greater than operator	
>=	Greater than or equal operator	
<	Less than operator	
<=	Less than or equal operator	
IS NULL	NULL check	
IS NOT NULL	Not NULL check	
IS EMPTY	EMPTY check	
IS NOT EMPTY	Not EMPTY check	

The result of the comparison operation is true or false.

# Note

• Comparability between values of different types is often undefined. The result could be NULL or others.

• EMPTY is currently used only for checking, and does not support functions or operations such as GROUP BY, count(), sum(), max(), hash(), collect(), + or \*.

## OpenCypher compatibility

openCypher does not have EMPTY . Thus EMPTY is not supported in MATCH statements.

#### Examples

==

String comparisons are case-sensitive. Values of different types are not equal.

#### Q Note

The equal operator is = in nGQL, while in openCypher it is =.

nebula> RETURN 'A' == 'a', toUpper('A') == toUpper('a'), toLower('A') == toLower('a');

```
| ("A"=="a") | (toUpper("A")==toUpper("a")) | (toLower("A")==toLower("a")) |
+-----+
| false | true | true
```

```
nebula> RETURN '2' == 2, toInteger('2') == 2;
+-----+
| ("2"=2) | (toInteger("2")=2) | .
```

false	true	
+	+	+

>

nebula> RETURN 3 > 2;	
++	
(3>2)	
++	
true	
++	
nebula> WITH 4 AS one, 3 AS two RETURN one > two AS resu	\ lt;
++	
result	
++	
true	
++	

>=

nebula> RETURN 2 >= "2", 2 >= 2;
++
(2>="2")   (2>=2)
++
NULL   true
++

<

nebula>	<b>YTELD</b>	2 0	<	1 9
+	+	2.0		1.0
. (2<1.0	- -			
+	·/   +			
falso	i			
1 10130	1			

<=

nebu	la>	YIELD	0.11	<=	0.11;
+			+		
(0	. 11	=0.11)	)		
+			+		
tri	Je				
+			+		

!=

nebula> YIELD	1	!=	'1';
++			
(1!="1")			
++			
true			
++			

IS [NOT] NULL

nebula> RETURN null IS NULL AS value1, null == null AS value2, null != null AS value3; ++ + value1 value2   value3   ++   true  NULL_  NULL   ++							
nebula> RETURN length(M	NULL), size(NULL), c	ount(NULL), NULI	L IS NULL, NULL IS I	NOT NULL, sir	n(NULL), NULL +	NULL, [1, NULL]	IS NULL
length(NULL)   size(N	IULL)   count(NULL)	NULL IS NULL	+   NULL IS NOT NULL +	sin(NULL)	(NULL+NULL)	[1,NULL] IS NULL	-+   -+
NULL  NULL	0	true	false	NULL	NULL	false	1
++ nebula> WITH {name: null} AS `map` \							
Talse   ++ nebula> WITH {name: 'Mats', name2: 'Pontus'} AS map1, \ {name: null} AS map2, {notName: 0, notName2: null } AS map3 \ RETURN map1.name IS NULL, map2.name IS NOT NULL, map3.name IS NULL;							

| map1.name IS NULL | map2.name IS NOT NULL | map3.name IS NULL |

+	+	+	+
false	false	true	1

nebula> MATCH (n:player) \ RETURN n.player.age IS NULL, n.player.name IS NOT NULL, n.player.empty IS NULL;

+   n.player.age IS NULL	+   n.player.name IS NOT NULL	+   n.player.empty IS NULL	F 
+	+	+	ŀ
false	true	true	
false	true	true	

#### IS [NOT] EMPTY

nebula>	RETURN	null	IS	EMPTY;
+		+		
NULL ]	ES EMPTY	(		
+		+		
false				
+		+		

nebula> RETURN "a" IS NOT EMPTY;

+.					-+
L	"a"	IS	NOT	EMPTY	
+.					-+
L	true	2			
+.					-+

nebula> G0 FROM "player100" OVER \* WHERE properties(\$\$).name IS NOT EMPTY YIELD dst(edge);

Τ-		
l	dst(EDGE)	
+-		ė.
L	"team204"	
	"player101"	
	"player125"	
+-	4	1

# 4.3.2 Boolean operators

NebulaGraph supports the following boolean operators.

Name	Description
AND	Logical AND
NOT	Logical NOT
OR	Logical OR
XOR	Logical XOR

For the precedence of the operators, refer to Operator Precedence.

For the logical operations with  $\ensuremath{\operatorname{\mathsf{NULL}}}$  , refer to  $\ensuremath{\operatorname{\mathsf{NULL}}}$  .

# Legacy version compatibility

• Non-zero numbers cannot be converted to boolean values.

# 4.3.3 Pipe operators

Multiple queries can be combined using pipe operators in nGQL.

#### OpenCypher compatibility

Pipe operators apply to native nGQL only.

#### Syntax

One major difference between nGQL and SQL is how sub-queries are composed.

- In SQL, sub-queries are nested in the query statements.
- In nGQL, the shell style PIPE (|) is introduced into the sub-queries.

#### Examples

```
nebula> G0 FROM "player100" OVER follow \
    YIELD dst(edge) AS dstid, properties($$).name AS Name | \
    G0 FROM $-.dstid OVER follow YIELD dst(edge);
+----+
| dst(EDGE) |
+----+
"player100" |
"player102" |
"player102" |
"player100" |
+-----+
```

Users must define aliases in the YIELD clause for the reference operator \$- to use, just like \$-.dstid in the preceding example.

#### Performance tips

In NebulaGraph, pipes will affect the performance. Take A | B as an example, the effects are as follows:

- 1. Pipe operators operate synchronously. That is, the data can enter the pipe clause as a whole after the execution of clause A before the pipe operator is completed.
- 2. If A sends a large amount of data to ], the entire query request may be very slow. You can try to split this statement.
- a. Send A from the application,
- b. Split the return results on the application,
- c. Send to multiple graphd processes concurrently,
- d. Every graphd process executes part of B.

This is usually much faster than executing a complete  $A \mid B$  with a single graphd process.

```
Last update: January 5, 2024
```

#### 4.3.4 Set operators

This topic will describe the set operators, including UNION, UNION ALL, INTERSECT, and MINUS. To combine multiple queries, use these set operators.

All set operators have equal precedence. If a nGQL statement contains multiple set operators, NebulaGraph will evaluate them from left to right unless parentheses explicitly specify another order.



The names and order of the variables defined in the query statements before and after the set operator must be consistent. For example, the names and order of a,b,c in RETURN a,b,c UNION RETURN a,b,c need to be consistent.

#### UNION, UNION DISTINCT, and UNION ALL

<left> UNION [DISTINCT | ALL] <right> [ UNION [DISTINCT | ALL] <right> ...]

- Operator UNION DISTINCT (or by short UNION) returns the union of two sets A and B without duplicated elements.
- Operator UNION ALL returns the union of two sets A and B with duplicated elements.
- The <left> and <right> must have the same number of columns and data types. Different data types are converted according to the Type Conversion.

EXAMPLES

```
# The following statement returns the union of two query results without duplicated elements.
nebula> G0 FROM "player102" OVER follow YIELD dst(edge) \
        LINTON \
        GO FROM "player100" OVER follow YIELD dst(edge);
dst(EDGE)
 "player100"
  "plaver101
  "player125"
nebula> MATCH (v:player) \
        WITH v.player.name AS v \
RETURN n ORDER BY n LIMIT 3 \
       UNION \
UNWIND ["Tony Parker", "Ben Simmons"] AS n \
        RETURN n;
| n
 "Amar'e Stoudemire"
  "Aron Baynes"
 "Ben Simmons"
"Tony Parker"
# The following statement returns the union of two query results with duplicated elements.
nebula> GO FROM "player102" OVER follow YIELD dst(edge)
        UNTON ALL
        GO FROM "player100" OVER follow YIELD dst(edge);
| dst(EDGE)
  "player100"
  "player101'
  "player101"
  "player125"
nebula> MATCH (v:player) \
        WITH v.player.name AS n
        RETURN N ORDER BY N LIMIT 3 \
        UNION ALL \
        UNWIND ["Tony Parker", "Ben Simmons"] AS n \setminus
        RFTURN n.
| n
| "Amar'e Stoudemire" |
```

"Ar	ron Baynes"
"Be	en Simmons"
"To	ony Parker"
"Be	en Simmons"
+	+
# UNI	tow can also work with the YIELD statement. The DISTINCT keyword will check duplication by all the columns for every line, and remove duplicated lines if every column is the same.
nebul	la> GO FROM "player102" OVER follow \
	YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties(\$\$).age AS Age \
	UNION /* DISTINCT */ \
	G0 FROM "player100" OVER follow \
	YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties(\$\$).age AS Age;
+	+++
id	Degree   Age

	id		Degree		Age	
i	"player100"	1	75	Ì	42	Ì
Ì	"player101"	İ	75	İ	36	İ
	"player101"		95		36	
	"player125"		95		41	
+		+		+		+

#### INTERSECT

<left> INTERSECT <right>

- Operator INTERSECT returns the intersection of two sets A and B (denoted by A  $\cap$  B).
- Similar to UNION, the Left and right must have the same number of columns and data types. Different data types are converted according to the Type Conversion.

EXAMPLE

```
# The following statement returns the intersection of two query results. nebula> G0 FROM "player102" OVER follow \backslash
           YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties($$).age AS Age \
           INTERSECT \
           GO FROM "player100" OVER follow
           YIELD dst(edge) AS id, properties(edge).degree AS Degree, properties($$).age AS Age;
| id | Degree | Age
+----+
nebula> MATCH (v:player)-[e:follow]->(v2) \
    WHERE id(v) = "player102" \
           RETURN id(v2) As id, e.degree As Degree, v2.player.age AS Age \
           INTERSECT
           MATCH (v:player)-[e:follow]->(v2) \
          \label{eq:WERE} \begin{array}{l} \mathsf{id}(v) = \mathsf{"player100"} \setminus \\ \mathsf{RETURN} \ \mathsf{id}(v2) \ \mathsf{As} \ \mathsf{id}, \ \mathsf{e.degree} \ \mathsf{As} \ \mathsf{Degree}, \ v2.player.age \ \mathsf{AS} \ \mathsf{Age}; \end{array}
| id | Degree | Age |
               ---+-
+----+
nebula> UNWIND [1,2] AS a RETURN a \setminus
           INTERSECT
           UNWIND [1,2,3,4] AS a \backslash
           RETURN a;
+---+
| a |
+---+
| 1 |
| 2 |
```

#### MINUS

<left> MINUS <right>

Operator MINUS returns the subtraction (or difference) of two sets A and B (denoted by A-B). Always pay attention to the order of left and right. The set A-B consists of elements that are in A but not in B.

EXAMPLE

```
| "player125"
nebula> GO FROM "player102" OVER follow YIELD dst(edge) AS id\
         MINUS
        GO FROM "player100" OVER follow YIELD dst(edge) AS id;
| id
| "player100" |
nebula> MATCH (v:player)-[e:follow]->(v2) \
         WHERE id(v) == "player102"
         RETURN id(v2) AS id\
        MINUS
        MATCH (v:player)-[e:follow]->(v2) \setminus WHERE id(v) =="player100" \setminus
        RETURN id(v2) AS id;
| id
| "player100" |
nebula> UNWIND [1,2,3] AS a RETURN a \
        MINUS \
WITH 4 AS a \
        RETURN a;
+---+
| a |
| 1 |
| 2 |
3
```

# Precedence of the set operators and pipe operators

Please note that when a query contains a pipe | and a set operator, the pipe takes precedence. Refer to Pipe for details. The query GO FROM 1 UNION GO FROM 2 | GO FROM 3 is the same as the query GO FROM 1 UNION (GO FROM 2 | GO FROM 3).

EXAMPLES



The above query executes the statements in the red bar first and then executes the statement in the green box.

The parentheses can change the execution priority. For example:



In the above query, the statements within the parentheses take precedence. That is, the UNION operation will be executed first, and its output will be executed as the input of the next operation with pipes.

# 4.3.5 String operators

You can use the following string operators for concatenating, querying, and matching.

Name	Description	
+	Concatenates strings.	
CONTAINS	Performs searchings in strings.	
(NOT) IN	Checks whether a value is within a set of values.	
(NOT) STARTS WITH	Performs matchings at the beginning of a string.	
(NOT) ENDS WITH	Performs matchings at the end of a string.	
Regular expressions	Perform string matchings using regular expressions.	

Q Note

All the string searchings or matchings are case-sensitive.

# Examples

+

```
nebula> RETURN 'a' + 'b';
+------+
| ("a"+"b") |
''ab" |
+-----+
nebula> UNWIND 'a' AS a UNWIND 'b' AS b RETURN a + b;
+-----+
| (a+b) |
+-----+
| "ab" |
+-----+
```

#### CONTAINS

The  $\ensuremath{\texttt{CONTAINS}}$  operator requires string types on both left and right sides.

nebula> MATCH (s AND t.te	:player)-[e:ser am.name CONTAIN:	ve]->(t:team) S "ets" RETURM	WHERE id(s) = Is.player.na	== "player101" \ me, e.start_year, e.end	d_year, t.team.name;
s.player.name	+   e.start_year +	e.end_year	t.team.name	-+   .+	
"Tony Parker" +	2018	2019	"Hornets"	-,   -+	
nebula> GO FROM "player101" OVER serve WHERE (STRING)properties(edge).start_year CONTAINS "19" AND \ properties(\$^\).name CONTAINS "ny" \ YIELD properties(\$^\).name, properties(edge).start_year, properties(edge).end_year, properties(\$\$).name;					
properties(\$^)	.name   propert	ies(EDGE).star	rt_year   pro	perties(EDGE).end_year	properties(\$\$).name
"Tony Parker"	1999		2018	}	"Spurs" +
nebula> GO FROM "player101" OVER serve WHERE !(properties(\$\$).name CONTAINS "ets") \ YIELD properties(\$^).name, properties(edge).start_year, properties(edge).end_year, properties(\$\$).name;					
properties(\$^)	.name   propert	ies(EDGE).star	rt_year   pro	perties(EDGE).end_year	properties(\$\$).name
"Tony Parker"	1999		2018	}	"Spurs" +

#### (NOT) IN

true	true	NULL
+	+	+

## (NOT) STARTS WITH

nebula> RETURN 'apple' STARTS +	WITH 'app', 'apple' STARTS V +	VITH 'a', 'apple' STARTS WI	TH toUpper('a');	
("apple" STARTS WITH "app")	' ("apple" STARTS WITH "a")	("apple" STARTS WITH tou	Jpper("a"))	
true	true	false		
nobulas DETIDN 'appla' STADTC WITH 'b' 'appla' NOT STADTC WITH 'app'.				
++				
("apple" STARTS WITH "b")   ("apple" NOT STARTS WITH "app")				
++				

false	false	L
+	+	÷

#### (NOT) ENDS WITH

nebula> RETURN 'apple' ENDS	WITH 'app', 'apple' ENDS W	ITH 'e', 'apple' ENDS WITH	'E', 'apple' ENDS WITH 'b';
("apple" ENDS WITH "app")	("apple" ENDS WITH "e")	("apple" ENDS WITH "E")	("apple" ENDS WITH "b")
false	true	false	false

REGULAR EXPRESSIONS

# Note

Regular expressions cannot work with native nGQL statements ( GO , FETCH , LOOKUP , etc.). Use it in openCypher only ( MATCH , WHERE , etc.).

NebulaGraph supports filtering by using regular expressions. The regular expression syntax is inherited from std::regex . You can match on regular expressions by using =- 'regexp'. For example:

nebula> RETURN "384748.39" =- "\\d+(\\.\\d{2})?";
+-----+
| ("384748.39"=-"\d+(\.\d{2})?") |
+----+
| true |
+----+
nebula> MATCH (v:player) WHERE v.player.name =~ 'Tony.\*' RETURN v.player.name;
+----+
| v.player.name |
+-----+
| '.oplayer.name |
+-----+
| "Tony Parker" |

# 4.3.6 List operators

 $NebulaGraph \ supports \ the \ following \ list \ operators:$ 

List	t operator	Description	
+		Concatenates lists.	
IN		Checks if an element exists in a list.	
[]		Accesses an element(s) in a list using the index operator.	

# Examples

nebula> YIELD [1,2,3,4,5]+[6,7] AS myList;
+
++   [1, 2, 3, 4, 5, 6, 7]   ++
nebula> RETURN size([NULL, 1, 2]); ++   size([NULL,1,2])
3   ++
nebula> RETURN NULL IN [NULL, 1]; ++
(NULL IN [NULL,I])   +
nebula> WITH [2, 3, 4, 5] AS numberlist \ UNWIND numberlist AS number \ WITH number \ WHERE number IN [2, 3, 8] \ RETURN number;
++   number
++
2     3   ++
<pre>nebula&gt; WITH ['Anne', 'John', 'Bill', 'Diane', 'Eve'] AS names RETURN names[1] AS result; ++   result  </pre>
++   "John"   ++

# 4.3.7 Arithmetic operators

NebulaGraph supports the following arithmetic operators.

Name	Description	
+	Addition operator	
	Minus operator	
*	Multiplication operator	
1	Division operator	
%	Modulo operator	
	Changes the sign of the argument	

# Examples

nebula> RETURN 1+2 AS result;
++
result
++
3
3
++
nebula> RETURN -10+5 AS result;
++
result
++
-5
++
nebula> RETURN (3*8)%5 AS result:
++
result
++
4
++

# 4.3.8 Operator precedence

The following list shows the precedence of nGQL operators in descending order. Operators that are shown together on a line have the same precedence.

- - (negative number)
- !, NOT
- \*, /, %
- -, +
- ==, >=, >, <=, <, <>, !=
- AND
- OR, XOR
- = (assignment)

For operators that occur at the same precedence level within an expression, evaluation proceeds left to right, with the exception that assignments evaluate right to left.

The precedence of operators determines the order of evaluation of terms in an expression. To modify this order and group terms explicitly, use parentheses.

#### Examples

nebula> REIURN	2+3^5;
++   (2+(3*5))   ++	
17	
nebula> RETURN ++	(2+3)*5
((2+3)*5)	
25	

### OpenCypher compatibility

In openCypher, comparisons can be chained arbitrarily, e.g.,  $x < y \ll z$  is equivalent to x < y AND  $y \ll z$  in openCypher.

But in nGQL,  $x < y \ll z$  is equivalent to  $(x < y) \ll z$ . The result of (x < y) is a boolean. Compare it with an integer z, and you will get the final result NULL.

# 4.4 Functions and expressions

## 4.4.1 Built-in math functions

This topic describes the built-in math functions supported by NebulaGraph.

#### abs()

abs() returns the absolute value of the argument.

Syntax: abs(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

```
nebula> RETURN abs(-10);
+-----+
| abs(-10)) |
+-----+
| 10 |
+------+
| abs(5-6);
+------+
| abs((5-6)) |
+-----+
| 1 |
```

#### floor()

floor() returns the largest integer value smaller than or equal to the argument.(Rounds down)

Syntax: floor(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

nebula>	RETURN	floor(9.9);
+	+	
floor	(9.9)	
+	+	
9.0		

#### ceil()

ceil() returns the smallest integer greater than or equal to the argument.(Rounds up)

Syntax: ceil(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

# Example:

```
nebula> RETURN ceil(9.1);
+-----+
| ceil(9.1) |
+-----+
```

| 10.0 |

#### round()

round() returns the rounded value of the specified number. Pay attention to the floating-point precision when using this function.

Syntax: round(<expression>, <digit>)

- expression : An expression of which the result type is double.
- digit : Decimal digits. If digit is less than 0, round at the left of the decimal point.
- Result type: Double

Example:

```
nebula> RETURN round(314.15926, 2);
+------+
| round(314.15926,2) |
+------+
| 314.16 |
+------+
| round(314.15926, -1);
+------+
| round(314.15926, -(1)) |
+-----+
+ -----+
+ -----+
| 310.0 |
```

# sqrt()

sqrt() returns the square root of the argument.

Syntax: sqrt(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

Example:

```
nebula> RETURN sqrt(9);
+-----+
| sqrt(9) |
+-----+
| 3.0 |
+-----+
```

#### cbrt()

cbrt() returns the cubic root of the argument.

Syntax: cbrt(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

nebula>	RETURN	cbrt(8);
+	+	
cbrt(	3)	
+	+	
2.0		
+	+	

#### hypot()

hypot() returns the hypotenuse of a right-angled triangle.

```
Syntax: hypot(<expression_x>,<expression_y>)
```

- $expression_x$ ,  $expression_y$ : An expression of which the result type is double. They represent the side lengths x and y of a right triangle.
- Result type: Double

## Example:

```
nebula> RETURN hypot(3,2*2);
+-----+
| hypot(3,(2*2)) |
+----+
| 5.0 |
+----++
```

#### pow()

pow() returns the result of  $x^y$ .

Syntax: pow(<expression\_x>,<expression\_y>,)

- $expression_x$ : An expression of which the result type is double. It represents the base x.
- expression\_y : An expression of which the result type is double. It represents the exponential y.
- Result type: Double

## Example:

```
nebula> RETURN pow(3,3);
+----+
| pow(3,3) |
+----+
| 27 |
```

## exp()

exp() returns the result of  $e^{X}$ .

Syntax: exp(<expression>)

- expression : An expression of which the result type is double. It represents the exponential x.
- Result type: Double

Example:

```
nebula> RETURN exp(2);
+-----+
| exp(2) |
+----+
| 7.38905609893065 |
+----+
```

## exp2()

exp2() returns the result of  $2^{X}$ .

Syntax: exp2(<expression>)

- expression : An expression of which the result type is double. It represents the exponential x.
- Result type: Double

#### Example:

```
nebula> RETURN exp2(3);
+-----+
| exp2(3) |
+-----+
| 8.0 |
+-----+
```

### log()

log() returns the base-e logarithm of the argument. (\(log\_{e}{N}\))

Syntax: log(<expression>)

- expression : An expression of which the result type is double. It represents the antilogarithm N.
- Result type: Double

# Example:

nebula> RETURN log(8); +----+ | log(8) | +----+ | 2.0794415416798357 | +-----+

# log2()

log2() returns the base-2 logarithm of the argument. (\(log\_{2}{N}))

Syntax: log2(<expression>)

- expression : An expression of which the result type is double. It represents the antilogarithm N .
- Result type: Double

# Example:

```
nebula> RETURN log2(8);
+-----+
| log2(8) |
+----+
| 3.0 |
+----+
```

#### log10()

log10() returns the base-10 logarithm of the argument. (\(log\_{10}{N}))

#### Syntax: log10(<expression>)

- expression : An expression of which the result type is double. It represents the antilogarithm N.
- Result type: Double

# Example:

```
nebula> RETURN log10(100);
+-----+
| log10(100) |
+-----+
| 2.0 |
+
```

#### sin()

sin() returns the sine of the argument. Users can convert angles to radians using the function radians().

Syntax: sin(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

#### asin()

asin() returns the inverse sine of the argument. Users can convert angles to radians using the function radians().

Syntax: asin(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

```
nebula> RETURN asin(0.5);
+-----+
| asin(0.5) |
+----+
| 0.5235987755982989 |
+
```

#### cos()

cos() returns the cosine of the argument. Users can convert angles to radians using the function radians().

Syntax: cos(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

nebula> RETURN cos(0.5)	;
++	
cos(0.5)	
++	
0.8775825618903728	
++	

#### acos()

acos() returns the inverse cosine of the argument. Users can convert angles to radians using the function radians().

Syntax: acos(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

# Example:

```
nebula> RETURN acos(0.5);
+-----+
| acos(0.5) |
+-----+
```

| 1.0471975511965979 |

#### tan()

tan() returns the tangent of the argument. Users can convert angles to radians using the function radians().

Syntax: tan(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

Example:

<pre>nebula&gt; RETURN tan(0.5);</pre>	
++	
tan(0.5)	
++	
0.5463024898437905	
++	

## atan()

atan() returns the inverse tangent of the argument. Users can convert angles to radians using the function radians().

Syntax: atan(<expression>)

- expression : An expression of which the result type is double.
- Result type: Double

#### Example:

<pre>nebula&gt; RETURN atan(0.5);</pre>	
++	
atan(0.5)	
++	
0.4636476090008061	
+ +	

#### rand()

rand() returns a random floating point number in the range from 0 (inclusive) to 1 (exclusive); i.e.[0,1).

Syntax: rand()

• Result type: Double

Example:

nebula> RETURN rand(); +-----+ | rand() | +----+ | 0.6545837172298736 | +----+

# rand32()

rand32() returns a random 32-bit integer in [min, max).

Syntax: rand32(<expression\_min>,<expression\_max>)

- expression\_min : An expression of which the result type is int. It represents the minimum min.
- expression\_max : An expression of which the result type is int. It represents the maximum max .
- Result type: Int
- If you set only one argument, it is parsed as max and min is 0 by default. If you set no argument, the system returns a random signed 32-bit integer.

#### Example:

```
nebula> RETURN rand32(1,100);
+-----+
| rand32(1,100) |
+----+
| 63 |
+----+
```

#### rand64()

rand64() returns a random 64-bit integer in [min, max).

Syntax: rand64(<expression\_min>,<expression\_max>)

- expression\_min : An expression of which the result type is int. It represents the minimum min.
- expression\_max : An expression of which the result type is int. It represents the maximum max.
- Result type: Int
- If you set only one argument, it is parsed as max and min is 0 by default. If you set no argument, the system returns a random signed 64-bit integer.

#### Example:

```
nebula> RETURN rand64(1,100);
+-----+
| rand64(1,100) |
+----+
| 34 |
+-----+
```

### bit\_and()

bit\_and() returns the result of bitwise AND.

Syntax: bit\_and(<expression\_1>,<expression\_2>)

- expression\_1, expression\_2: An expression of which the result type is int.
- Result type: Int

Example:

```
nebula> RETURN bit_and(5,6);
+----+
| bit_and(5,6) |
+----+
| 4 |
+-----+
```

#### bit\_or()

bit\_or() returns the result of bitwise OR.

Syntax: bit\_or(<expression\_1>,<expression\_2>)

- expression\_1, expression\_2: An expression of which the result type is int.
- Result type: Int

#### Example:

```
nebula> RETURN bit_or(5,6);
+-----+
| bit_or(5,6) |
+----+
| 7 |
```

#### bit\_xor()

bit\_xor() returns the result of bitwise XOR.

Syntax: bit\_xor(<expression\_1>,<expression\_2>)

- expression\_1, expression\_2: An expression of which the result type is int.
- Result type: Int

#### Example:

```
nebula> RETURN bit_xor(5,6);
+----+
| bit_xor(5,6) |
+----+
| 3 |
+----+
```

#### size()

size() returns the number of elements in a list or a map, or the length of a string.

Syntax: size({<expression>|<string>})

- expression : An expression for a list or map.
- string : A specified string.
- Result type: Int

Example:

# range()

range() returns a list of integers from [start,end] in the specified steps.

Syntax: range(<expression\_start>,<expression\_end>[,<expression\_step>])

- expression\_start : An expression of which the result type is int. It represents the starting value start .
- expression\_end : An expression of which the result type is int. It represents the end value end .
- expression\_step : An expression of which the result type is int. It represents the step size step , step is 1 by default.
- Result type: List

## Example:

```
nebula> RETURN range(1,3*3,2);
+-----+
| range(1,(3*3),2) |
+----+
| [1, 3, 5, 7, 9] |
+----+
```

## sign()

sign() returns the signum of the given number. If the number is 0, the system returns 0. If the number is negative, the system returns -1. If the number is positive, the system returns 1.

Syntax: sign(<expression>)

- expression : An expression of which the result type is double.
- Result type: Int

Example:

```
nebula> RETURN sign(10);
+-----+
| sign(10) |
+-----+
| 1 |
```

#### e()

e() returns the base of the natural logarithm, e (2.718281828459045).

Syntax: e()

• Result type: Double

#### Example:

nebula> RETURN e(); +-----+ | e() | +-----+ | 2.718281828459045 | +-----+

## pi()

pi() returns the mathematical constant pi (3.141592653589793).

Syntax: pi()

• Result type: Double

# Example:

nebula> RETURN pi(); +----+ | pi() | +----+ | 3.141592653589793 |

# radians()

radians() converts angles to radians.

Syntax: radians(<angle>)

• Result type: Double

Example:

# 4.4.2 Aggregating functions

This topic describes the aggregating functions supported by NebulaGraph.

## avg()

avg() returns the average value of the argument.

Syntax: avg(<expression>)

• Result type: Double

#### Example:

```
nebula> MATCH (v:player) RETURN avg(v.player.age);
+-----+
| avg(v.player.age) |
+-----+
| 33.294117647058826 |
+------+
```

#### count()

count() returns the number of records.

- (Native nGQL) You can use count() and GROUP BY together to group and count the number of parameters. Use YIELD to return.
- (OpenCypher style) You can use count() and RETURN. GROUP BY is not necessary.

Syntax: count({<expression> | \*})

- count(\*) returns the number of rows (including NULL).
- Result type: Int

Example:

```
# The statement in the following example searches for the people whom `player101` follows and people who follow `player101`, i.e. a bidirectional query.
# Group and count the number of parameters.
nebula> GO FROM "player101" OVER follow BIDIRECT \
        YIELD properties($$).name AS Name
        | GROUP BY $-.Name YIELD $-.Name, count(*);
$-.Name
                        count(*)
  "LaMarcus Aldridge" | 2
  "Tim Duncan"
                          2
  "Marco Belinelli"
                        | 1
  "Manu Ginobili"
                         1
  "Boris Diaw"
                          1
  "Dejounte Murray"
                         | 1
# Count the number of parameters.
nebula> MATCH (v1:player)-[:follow]-(v2:player) \
WHERE id(v1)== "player101" \
        RETURN v2.player.name AS Name, count(*) as cnt ORDER BY cnt DESC;
        ----+-
| Name
                        | cnt |
  "LaMarcus Aldridge" | 2
 "Tim Duncan"
"Boris Diaw"
                          2
                         İ 1
  "Manu Ginobili"
                         | 1
 "Dejounte Murray"
"Marco Belinelli"
                          1
                         11
```

The preceding example retrieves two columns:

- \$-.Name: the names of the people.
- count(\*) : how many times the names show up.

Because there are no duplicate names in the basketballplayer dataset, the number 2 in the column count(\*) shows that the person in that row and player101 have followed each other.

```
# a: The statement in the following example retrieves the age distribution of the players in the dataset.
nebula> LOOKUP ON player
         YIELD player.age As playerage \
| GROUP BY $-.playerage \
         +---+
| age | number
  34
       | 4

    33
    4

    30
    4

    29
    4

38 3
# b: The statement in the following example retrieves the age distribution of the players in the dataset.
nebula> MATCH (n:player) \
         RETURN n.player.age as age, count(*) as number \
ORDER BY number DESC, age DESC;
| age | number |
       | 4
| 4
| 4
| 34
  33
30
 29 | 4
38 | 3
# The statement in the following example counts the number of edges that Tim Duncan relates.
nebula> MATCH (v:player{name:"Tim Duncan"}) -[e]- (v2) \
         RETURN count(e);
| count(e) |
| 13
# The statement in the following example counts the number of edges that Tim Duncan relates and returns two columns (no DISTINCT and DISTINCT) in multi-hop queries.
nebula> MATCH (n:player {name : "Tim Duncan"})-[]->(friend:player)-[]->(fof:player) \
RETURN count(fof), count(DISTINCT fof);
```

++				
count(fof)	<pre>count(distinct fof)  </pre>			
++	+			
4	3			
++	+			

#### max()

max() returns the maximum value.

Syntax: max(<expression>)

• Result type: Same as the original argument.

### Example:

```
nebula> MATCH (v:player) RETURN max(v.player.age);
+-----+
| max(v.player.age) |
+-----+
| 47 |
+-----+
```

#### min()

min() returns the minimum value.

Syntax: min(<expression>)

• Result type: Same as the original argument.

Example:

```
nebula> MATCH (v:player) RETURN min(v.player.age);
+-----+
| min(v.player.age) |
+-----+
| 20 |
+-----+
```

#### collect()

collect() returns a list containing the values returned by an expression. Using this function aggregates data by merging multiple records or values into a single list.

Syntax: collect(<expression>)

• Result type: List

Example:

```
nebula> UNWIND [1, 2, 1] AS a \backslash RETURN a;
+---+
| a |
+---+
| 1 |
| 2 |
| 1 |
nebula> UNWIND [1, 2, 1] AS a \setminus
         RETURN collect(a);
| collect(a)
[1, 2, 1]
nebula> UNWIND [1, 2, 1] AS a \
    RETURN a, collect(a), size(collect(a));
 | a | collect(a) | size(collect(a))
 | 2 | [2]
                     | 1
 1 [1, 1]
                      2
# The following examples sort the results in descending order, limit output rows to 3, and collect the output into a list.
nebula> UNNIND ["c", "b", "a", "d" ] AS p \
WITH p AS q \
ORDER BY q DESC LIMIT 3 \
RETURN collect(q);
 | collect(q)
 | ["d", "c", "b"] |
nebula> WITH [1, 1, 2, 2] AS coll \
UNWIND coll AS x \
          WITH DISTINCT x \
         RETURN collect(x) AS ss;
 ss
| [1, 2] |
nebula> MATCH (n:player)
         RETURN collect(n.player.age);
 collect(n.player.age)
[32, 32, 34, 29, 41, 40, 33, 25, 40, 37, ...
# The following example aggregates all the players' names by their ages.
nebula> MATCH (n:player) \
         RETURN n.player.age AS age, collect(n.player.name);
 +---+-
| age | collect(n.player.name)
```

++
<pre>24   ["Giannis Antetokounmpo"] 20   ["Luka Doncic"] 25   ["Joel Embiid", "Kyle Anderson"]</pre>
TT
nebula> GO FROM "player100" OVER serve \ YIELD properties(\$\$).name AS name \   GROUP BY \$name \ YIELD collect(\$name) AS name;
++
name
++
["Spurs"]
++
nebula> LOOKUP ON player \ YIELD player.age As playerage \   GROUP BY \$playerage \ YIELD collect(\$playerage) AS playerage;
++
playerage
++
[22]     [47]
[43]
[25, 25]
++

#### std()

std() returns the population standard deviation.

Syntax: std(<expression>)

• Result type: Double

## Example:

```
nebula> MATCH (v:player) RETURN std(v.player.age);
+-----+
| std(v.player.age) |
+-----+
| 6.423895701687502 |
```

# sum()

sum() returns the sum value.

Syntax: sum(<expression>)

• Result type: Same as the original argument.

# Example:

nebula> MAT	CH (v:player)	RETURN	<pre>sum(v.player.age);</pre>	
+	+			
sum(v.pla	yer.age)			
+	+			
1698				
+	+			

# Aggregating example

```
nebula> G0 FROM "player100" OVER follow YIELD dst(edge) AS dst, properties($$).age AS age \
    | GROUP BY $-.dst \
    YIELD \
    $-.dst AS dst, \
    toInteger((sum($-.age)/count($-.age)))+avg(distinct $-.age+1)+1 AS statistics;
+------+
| dst | statistics |
+-----+
| "player125" | 84.0 |
| "player125" | 84.0 |
| "player101" | 74.0 |
+-----+
```
## 4.4.3 Built-in string functions

This topic describes the built-in string functions supported by NebulaGraph.

#### Precautions

- A string type is used to store a sequence of characters (text). The literal constant is a sequence of characters of any length surrounded by double or single quotes.
- Like SQL, the position index of nGQL starts from 1, while in C language it starts from 0.

#### strcasecmp()

strcasecmp() compares string a and b without case sensitivity.

Syntax: strcasecmp(<string\_a>,<string\_b>)

- string\_a, string\_b: Strings to compare.
- Result type: Int
- When string\_a = string\_b, the return value is 0. When string\_a > string\_b, the return value is greater than 0. When string\_a < string\_b, the return value is less than 0.

#### Example:

```
nebula> RETURN strcasecmp("a","aa");
+-----+
| strcasecmp("a","aa") |
+-----+
| -97 |
+ -----+
```

#### lower() and toLower()

lower() and toLower() can both returns the argument in lowercase.

Syntax: lower(<string>) , toLower(<string>)

- string : A specified string.
- Result type: String

## Example:

```
nebula> RETURN lower("Basketball_Player");
+----+
| lower("Basketball_Player") |
+----+
| "basketball_player" |
```

# upper() and toUpper()

upper() and toUpper() can both returns the argument in uppercase.

Syntax: upper(<string>) , toUpper(<string>)

- string: A specified string.
- Result type: String

```
nebula> RETURN upper("Basketball_Player");
+----+
upper("Basketball_Player") |
+----+
"BASKETBALL_PLAYER" |
```

## length()

length() returns the length of the given string in bytes.

Syntax: length({<string>|<path>})

- string : A specified string.
- path : A specified path represented by a variable.
- Result type: Int

Example:

```
nebula> RETURN length("basketball");
+------+
+-------+
| 10 |
+------+
```

nebula> MATCH p=(v:player{name:"Tim Duncan"})-->(v2) return length(p);
+----+
| length(p) |
+----+

| 1 | 1 | 1

#### trim()

trim() removes the spaces at the leading and trailing of the string.

Syntax: trim(<string>)

- string: A specified string.
- Result type: String

Example:

```
nebula> RETURN trim(" basketball player ");
+-----+
| trim(" basketball player ") |
+----+
| "basketball player" |
+----+
```

# ltrim()

ltrim() removes the spaces at the leading of the string.

Syntax: ltrim(<string>)

- string: A specified string.
- Result type: String

```
nebula> RETURN ltrim(" basketball player ");
+-----+
| ltrim(" basketball player ") |
+----+
```

| "basketball player " |

#### rtrim()

rtrim() removes the spaces at the trailing of the string.

Syntax: rtrim(<string>)

- string: A specified string.
- Result type: String

Example:

```
nebula> RETURN rtrim(" basketball player ");
+-----+
| rtrim(" basketball player ") |
+-----+
| " basketball player" |
+-----+
```

## left()

left() returns a substring consisting of several characters from the leading of a string.

Syntax: left(<string>,<count>)

- string: A specified string.
- count : The number of characters from the leading of the string. If the string is shorter than count , the system returns the string itself.
- Result type: String

Example:

```
nebula> RETURN left("basketball_player",6);
+----+
| left("basketball_player",6) |
+---+
| "basket" |
+----+
```

## right()

right() returns a substring consisting of several characters from the trailing of a string.

Syntax: right(<string>,<count>)

- string: A specified string.
- count : The number of characters from the trailing of the string. If the string is shorter than count , the system returns the string itself.
- Result type: String

```
nebula> RETURN right("basketball_player",6);
+-----+
| right("basketball_player",6) |
+----+
| "player" |
+-----+
```

#### lpad()

lpad() pads a specified string from the left-side to the specified length and returns the result string.

Syntax: lpad(<string>,<count>,<letters>)

- string: A specified string.
- count : The length of the string after it has been left-padded. If the length is less than that of string, only the length of string characters **from front to back** will be returned.
- letters : A string to be padding from the leading.
- Result type: String

Example:

```
nebula> RETURN Lpad("abcd",10,"b");
+-----+
| Lpad("abcd",10,"b") |
+-----+
| "bbbbbabcd" |
+-----+
| Lpad("abcd",3,"b");
+------+
| Lpad("abcd",3,"b") |
+-----+
+ ------+
+ ------+
```

#### rpad()

rpad() pads a specified string from the right-side to the specified length and returns the result string.

Syntax: rpad(<string>,<count>,<letters>)

- string : A specified string.
- count : The length of the string after it has been right-padded. If the length is less than that of string, only the length of string characters **from front to back** will be returned.
- letters : A string to be padding from the trailing.
- Result type: String

Example:

```
nebula> RETURN rpad("abcd",10,"b");
+-----+
| rpad("abcd",10,"b") |
+-----+
| "abcdbbbbbb" |
+-----+
nebula> RETURN rpad("abcd",3,"b");
+-----+
| rpad("abcd",3,"b") |
+----+
| "abc" + +
```

# substr() and substring()

substr() and substring() return a substring extracting count characters starting from the specified position pos of a specified string.

Syntax: substr(<string>,<pos>,<count>) , substring(<string>,<pos>,<count>)

- string : A specified string.
- pos : The position of starting extract (character index). Data type is int.
- count : The number of characters extracted from the start position onwards.
- Result type: String

EXPLANATIONS FOR THE RETURN OF SUBSTR() AND SUBSTRING()

- If pos is 0, it extracts from the specified string leading (including the first character).
- If pos is greater than the maximum string index, an empty string is returned.
- If pos is a negative number, BAD\_DATA is returned.
- If count is omitted, the function returns the substring starting at the position given by pos and extending to the end of the string.
- If count is 0, an empty string is returned.
- Using NULL as any of the argument of substr() will cause an issue.

# LenCypher compatibility

In openCypher, if a is null, null is returned.

#### Example:

## reverse()

reverse() returns a string in reverse order.

Syntax: reverse(<string>)

- string : A specified string.
- Result type: String

nebula> RETURN r	everse("abcdefg")
+	+
reverse("abcde	fg")
+	+
gfedcba"	
+	+

#### replace()

replace() replaces string a in a specified string with string b.

Syntax: replace(<string>,<substr\_a>,<string\_b>)

- string: A specified string.
- substr\_a : String a.
- string\_b : String b.
- Result type: String

Example:

```
nebula> RETURN replace("abcdefg","cd","AAAAA");
+------+
| replace("abcdefg","cd","AAAAA") |
+-----+
| "abAAAAAefg" |
+------+
```

#### split()

split() splits a specified string at string b and returns a list of strings.

Syntax: split(<string>,<substr>)

- string: A specified string.
- substr : String b.
- Result type: List

Example:

```
nebula> RETURN split("basketballplayer","a");
+---+
| split("basketballplayer","a") |
+--+
| ["b", "sketb", "llpl", "yer"] |
+
```

## concat()

concat() returns strings concatenated by all strings.

Syntax: concat(<string1>,<string2>,...)

- The function requires at least two or more strings. If there is only one string, the string itself is returned.
- If any one of the strings is NULL , NULL is returned.
- Result type: String

```
//This example concatenates 1, 2, and 3.
nebula> RETURN concat("1","2","3") AS r;
+-----+
| r |
+-----+
| "123" |
+-----+
//In this example, one of the string is NULL.
nebula> RETURN concat("1","2",NULL) AS r;
+------+
| r |
+------+
| ...NULL__ |
```

## concat\_ws()

concat ws() returns strings concatenated by all strings that are delimited with a separator.

Syntax: concat\_ws(<separator>,<string1>,<string2>,...)

- The function requires at least two or more strings.
- If the separator is NULL , the  $\mbox{concat}_w\mbox{s}()$  function returns NULL .
- $\bullet$  If the separator is not  $\tt NULL$  and there is only one string, the string itself is returned.
- If there is a NULL in the strings, NULL is ignored during the concatenation.

#### Example:

<pre>//This example concatenates a, b, and c with the separator +. nebula&gt; RETURN concat_ws("+","a","b","c") AS r; ++   r  +   "a+b+c"   ++</pre>
<pre>//In this example, the separator is NULL. neubla&gt; RETURN concat_ws(NULL, "a", "b", "c") AS r; ++   r   ++  NULL   ++</pre>
<pre>//In this example, the separator is + and there is a NULL in the strings. nebula&gt; RETURN concat_ws("+","a",NULL,"b","c") AS r; ++   r   ++   "a+b+c"   ++</pre>
<pre>//In this example, the separator is + and there is only one string. nebula&gt; RETURN concat_ws("+","a") AS r; ++   r   ++   "a"   ++ nebula&gt; 60 FROM "player100" over follow \ YIELD concat_ws(" ",src(edge), properties(\$^).age, properties(\$\$).name, properties(edge).degree) AS</pre>
+

## extract()

extract() uses regular expression matching to retrieve a single substring or all substrings from a string.

Syntax: extract(<string>,"<regular\_expression>")

- string : A specified string
- regular\_expression : A regular expression
- Result type: List

```
nebula> MATCH (a:player)-[b:serve]-(c:team{name: "Lakers"}) \
    WHERE a.player.age > 45 \
    RETURN extract(a.player.name, "\\w+") AS result;
+------+
| result |
    "-----+
| ["Shaquille", "0", "Neal"] |
    +-----+
nebula> MATCH (a:player)-[b:serve]-(c:team{name: "Lakers"}) \
    WHERE a.player.age > 45 \
    RETURN extract(a.player.name, "hello") AS result;
+-----+
| result |
    result |
    -----+
| [] |
    +-----+
| [] |
    +----++
| [] |
```

# json\_extract()

json\_extract() converts the specified JSON string to a map.

Syntax: extract(<string>)

- string : A specified string, must be JSON string.
- Result type: Map

# Caution

• Only Bool, Double, Int, String value and NULL are supported.

• Only depth-1 nested Map is supported now. If nested Map depth is greater than 1, the nested item is left as an empty Map().

## Example:

```
nebula> YIELD json_extract('{"a": 1, "b": {}, "c": {"d": true}}') AS result;
+-----+
| result |
+----+
| {a: 1, b: {}, c: {d: true}} |
```

# 4.4.4 Built-in date and time functions

NebulaGraph supports the following built-in date and time functions:

Function	Description
int now()	Returns the current timestamp of the system.
timestamp timestamp()	Returns the current timestamp of the system.
date date()	Returns the current UTC date based on the current system.
time time()	Returns the current UTC time based on the current system.
datetime datetime()	Returns the current UTC date and time based on the current system.
map duration()	Returns the period of time. It can be used to calculate the specified time.

For more information, see Date and time types.

## Examples

nebula>	RETURN now(), ti	<pre>mestamp(), dat</pre>	e(), time(), d	latetime();	
+	+	+	-+	+	+
now()	timestamp(	()   date()	time()	datetime()	
164005 +	7560   1640057560	)   2021-12-21	03:32:40.35	51000   2021-12-21T03:32:4	40.351000   +

## 4.4.5 Schema-related functions

This topic describes the schema-related functions supported by NebulaGraph. There are two types of schema-related functions, one for native nGQL statements and the other for openCypher-compatible statements.

#### For nGQL statements

The following functions are available in YIELD and WHERE clauses of nGQL statements.

# Note

Since vertex, edge, vertices, edges, and path are keywords, you need to use AS <alias> to set the alias, such as GO FROM "player100" OVER follow YIELD edge AS e; .

ID(VERTEX)

id(vertex) returns the ID of a vertex.

Syntax: id(vertex)

• Result type: Same as the vertex ID.

Example:

nebula> LOOKUP ON player WHERE player.age > 45 YIELD id(vertex);
+------+
| id(VERTEX) |
+------+
| "player144" |
| "player140" |
+------+

PROPERTIES(VERTEX)

properties(vertex) returns the properties of a vertex.

Syntax: properties(vertex)

• Result type: Map

Example:

You can also use the property reference symbols ( \$^ and \$\$ ) instead of the vertex field in the properties() function to get all properties of a vertex.

- \$^ represents the data of the starting vertex at the beginning of exploration. For example, in GO FROM "player100" OVER follow reversely YIELD properties(\$^), \$^ refers to the vertex player100.
- \$\$ represents the data of the end vertex at the end of exploration.

properties(\$^) and properties(\$\$) are generally used in 60 statements. For more information, see Property reference.

# Caution

You can use properties().<property\_name> to get a specific property of a vertex. However, it is not recommended to use this method to obtain specific properties because the properties() function returns all properties, which can decrease query performance.

#### PROPERTIES(EDGE)

properties(edge) returns the properties of an edge.

Syntax: properties(edge)

• Result type: Map

Example:

# Caution

You can use properties(edge).<property\_name> to get a specific property of an edge. However, it is not recommended to use this method to obtain specific properties because the properties(edge) function returns all properties, which can decrease query performance.

TYPE(EDGE)

type(edge) returns the edge type of an edge.

Syntax: type(edge)

• Result type: String

#### Example:

nebula> GO FR	OM "player100"	OVER follow	\	
YIELD	<pre>src(edge), ds</pre>	t(edge), type	(edge), ra	nk(edge);
+	-+	-+	-+	+
src(EDGE)	dst(EDGE)	type(EDGE)	rank(ED	GE)
+	-+	-+	-+	+
"player100"	"player101"	"follow"	0	
"player100"	"player125"	"follow"	0	
		+	+	+

SRC(EDGE)

src(edge) returns the source vertex ID of an edge.

Syntax: src(edge)

• Result type: Same as the vertex ID.

Q Note

The semantics of the query for the starting vertex with src(edge) and  $properties(s^{A})$  are different. src(edge) indicates the starting vertex ID of the edge in the graph database, while properties( $s^{A}$ ) indicates the data of the starting vertex where you start to expand the graph, such as the data of the starting vertex player100 in the above GO statement.

DST(EDGE)

dst(edge) returns the destination vertex ID of an edge.

Syntax: dst(edge)

• Result type: Same as the vertex ID.

Example:

#### Q Note

dst(edge) indicates the destination vertex ID of the edge in the graph database.

RANK(EDGE)

rank(edge) returns the rank value of an edge.

Syntax: rank(edge)

• Result type: Int

Example:

VERTEX

vertex returns the information of vertices, including VIDs, tags, properties, and values. You need to use AS <alias> to set the alias.

Syntax: vertex

Example:

EDGE

edge returns the information of edges, including edge types, source vertices, destination vertices, ranks, properties, and values. You need to use AS <alias> to set the alias.

#### Syntax: edge

#### Example:

nebula> GO FROM "player100" OVER follow YIELD edge AS e	
++	
e	
++	
[:follow "player100"->"player101" @0 {degree: 95}]	
[:follow "player100"->"player125" @0 {degree: 95}]	
++	

#### VERTICES

vertices returns the information of vertices in a subgraph. For more information, see GET SUBGRAPH.

EDGES

edges returns the information of edges in a subgraph. For more information, see GET SUBGRAPH.

PATH

path returns the information of a path. For more information, see FIND PATH.

#### For statements compatible with openCypher

The following functions are available in RETURN and WHERE clauses of openCypher-compatible statements.

ID()

id() returns the ID of a vertex.

Syntax: id(<vertex>)

• Result type: Same as the vertex ID.

#### Example:

nebula> MATCH	(v:player)	RETURN	id(v)
+	-+		
id(v)			
+	-+		
"player129"	1		
"player115"			
player106"			
"player102"			

TAGS() AND LABELS()

tags() and labels() return the Tag of a vertex.

Syntax: tags(<vertex>) , labels(<vertex>)

#### • Result type: List

Example:

PROPERTIES()

properties() returns the properties of a vertex or an edge.

Syntax: properties(<vertex\_or\_edge>)

• Result type: Map

#### Example:

```
      nebula> MATCH (v:player)-[e:follow]-() RETURN properties(v), properties(e);

      +------+

      | properties(v)
      | properties(e) |

      +-----+

      | degree: 31, name: "Stephen Curry"}
      | {degree: 90} |

      | degree: 47, name: "Shaquille 0'Neal"}
      | degree: 100} |

      | degree: 34, name: "LeBron James"}
      | {degree: 13} |
```

TYPE()

type() returns the edge type of an edge.

Syntax: type(<edge>)

```
• Result type: String
```

Example:

TYPEID()

typeid() returns the internal ID value of the Edge type of the edge, which can be used to determine the direction by positive or negative.

Syntax: typeid(<edge>)

• Result type: Int

Example:

SRC()

src() returns the source vertex ID of an edge.

Syntax: src(<edge>)

• Result type: Same as the vertex ID.

DST()

dst() returns the destination vertex ID of an edge.

Syntax: dst(<edge>)

• Result type: Same as the vertex ID.

Example:

STARTNODE()

startNode() visits a path and returns its information of source vertex ID, including VIDs, tags, properties, and values.

Syntax: startNode(<path>)

Example:

#### ENDNODE()

endNode() visits a path and returns its information of destination vertex ID, including VIDs, tags, properties, and values.

Syntax: endNode(<path>)

Example:

RANK()

rank() returns the rank value of an edge.

Syntax: rank(<edge>)

• Result type: Int

Example:

Last update: January 31, 2024

# 4.4.6 List functions

This topic describes the list functions supported by NebulaGraph. Some of the functions have different syntax in native nGQL statements and openCypher-compatible statements.

#### Precautions

Like SQL, the position index in nGQL starts from 1, while in the C language it starts from 0.

#### General

RANGE()

range() returns the list containing all the fixed-length steps in [start,end].

```
Syntax: range(start, end [, step])
```

- step : Optional parameters. step is 1 by default.
- Result type: List

#### Example:

nebula> RETURN range(1,9,2); +-----+ | range(1,9,2) | +----+ | [1, 3, 5, 7, 9] | +----+

REVERSE()

reverse() returns the list reversing the order of all elements in the original list.

Syntax: reverse(<list>)

• Result type: List

Example:

TAIL()

tail() returns all the elements of the original list, excluding the first one.

Syntax: tail(<list>)

• Result type: List

Example:

HEAD()

head() returns the first element of a list.

Syntax: head(<list>)

• Result type: Same as the element in the original list.

#### Example:

LAST()

last() returns the last element of a list.

#### Syntax: last(<list>)

• Result type: Same as the element in the original list.

#### Example:

```
nebula> WITH [NULL, 4923, 'abc', 521, 487] AS ids \
RETURN last(ids);
+-----+
| last(ids) |
+------+
| 487 |
+-------+
```

REDUCE()

reduce() applies an expression to each element in a list one by one, chains the result to the next iteration by taking it as the initial value, and returns the final result. This function iterates each element le in the given list, runs the expression on le, accumulates the result with the initial value, and store the new result in the accumulator as the initial value of the next iteration. It works like the fold or reduce method in functional languages such as Lisp and Scala.

# $\mathcal{L}_{\mathbf{r}}$ enCypher compatibility

In openCypher, the reduce() function is not defined. nGQL will implement the reduce() function in the Cypher way.

Syntax: reduce(<accumulator> = <initial>, <variable> IN <list> | <expression>)

- accumulator : A variable that will hold the accumulated results as the list is iterated.
- initial : An expression that runs once to give an initial value to the accumulator.
- variable : A variable in the list that will be applied to the expression successively.
- list : A list or a list of expressions.
- expression : This expression will be run on each element in the list once and store the result value in the accumulator .
- Result type: Depends on the parameters provided, along with the semantics of the expression.

34   31   165     34   29   163     34   29   163     34   33   167     34   26   160     34   34   168     34   37   171   +++++ nebula> LOKUP ON play   GO FROM \$	er WHE Vertex	RE player. ID over fo	name == "Tony Parker" YIELD id(vertex) AS VertexID \ Llow \
WHERE properti YIELD properti	es(edg es(\$\$)	e).degree .name AS i	<pre>!= reduce(totalNum = 5, n IN range(1, 3)   properties(\$\$).age + totalNum + n) \ d, properties(\$\$).age AS age, properties(edge).degree AS degree;</pre>
+	+	++	
id	age	degree	
+	+	++	
"Tim Duncan"	42	95	
"LaMarcus Aldridge"	33	90	
"Manu Ginobili"	41	95	
+	+	++	

#### For nGQL statements

KEYS()

keys() returns a list containing the string representations for all the property names of vertices or edges.

Syntax: keys({vertex | edge})

• Result type: List

#### Example:

```
nebula> LOOKUP ON player \

WHERE player.age > 45 \

YIELD keys(vertex);

+------+

| keys(VERTEX) |

+----+

| ["age", "name"] |

| "age", "name"] |
```

LABELS()

labels() returns the list containing all the tags of a vertex.

Syntax: labels(verte)

• Result type: List

Example:

# For statements compatible with openCypher

KEYS()

keys() returns a list containing the string representations for all the property names of vertices, edges, or maps.

Syntax: keys(<vertex\_or\_edge>)

• Result type: List

LABELS()

labels() returns the list containing all the tags of a vertex.

Syntax: labels(<vertex>)

• Result type: List

#### Example:

NODES()

nodes() returns the list containing all the vertices in a path.

Syntax: nodes(<path>)

• Result type: List

Example:

<pre>nebula&gt; MATCH p=(v:player{name:"Tim Duncan"})&gt;(v2) \</pre>	
nodes(p)	
[("player100" :player{age: 42, name: "Tim Duncan"}), ("team204" :team{name: "Spurs"})]	
[("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker	'})]
[("player100" :player{age: 42, name: "Tim Duncan"}), ("player125" :player{age: 41, name: "Manu Ginobi	li"})]

RELATIONSHIPS()

relationships() returns the list containing all the relationships in a path.

Syntax: relationships(<path>)

• Result type: List

Example:

# 4.4.7 Type conversion functions

This topic describes the type conversion functions supported by NebulaGraph.

#### toBoolean()

toBoolean() converts a string value to a boolean value.

Syntax: toBoolean(<value>)

• Result type: Bool

Example:

## toFloat()

toFloat() converts an integer or string value to a floating point number.

Syntax: toFloat(<value>)

• Result type: Float

Example:

nebula> RETURN	toFloat(1), toFl	loat('1.3'), toFl	oat('1e3'),	toFloat('not a	number')
++-			+		+
toFloat(1)	toFloat("1.3")	toFloat("1e3")	toFloat("ne	ot a number")	
TT-			· · · · · · · · · · · · · · · · · · ·		T
1.0	1.3	1000.0	NULL		
++-			+		+

## toString()

toString() converts non-compound types of data, such as numbers, booleans, and so on, to strings.

Syntax: toString(<value>)

• Result type: String

Example:

nebula> RETURN to	String(9669) AS	int2str,	toString(null)	AS null2str;
+	+			
int2str   null2	str			
+	+			
"9669"  NUL	L			
++	+			

## toInteger()

toInteger() converts a floating point or string value to an integer value.

Syntax: toInteger(<value>)

• Result type: Int

#### Example:

nebula> RETURN t	toInteger(1), toIr	nteger('1'), toInteg	ger('1e3'), toInteger('no	t a number'); +
toInteger(1)	toInteger("1")	<pre>toInteger("1e3")</pre>	toInteger("not a number	")
	1	1000		+ 

#### toSet()

toSet() converts a list or set value to a set value.

Syntax: toSet(<value>)

• Result type: Set

## Example:

```
nebula> RETURN toSet(list[1,2,3,1,2]) AS list2set;
+-----+
| list2set |
+-----+
| {3, 1, 2} |
+-----+
```

#### hash()

hash() returns the hash value of the argument. The argument can be a number, a string, a list, a boolean, null, or an expression that evaluates to a value of the preceding data types.

The source code of the hash() function (MurmurHash2), seed ( 0xc70f6907UL ), and other parameters can be found in MurmurHash2.h.

For Java, the hash function operates as follows.

```
MurmurHash2.hash64("to_be_hashed".getBytes(),"to_be_hashed".getBytes().length, 0xc70f6907)
```

Syntax: hash(<string>)

• Result type: Int

Example:

# 4.4.8 Conditional expressions

This topic describes the conditional functions supported by NebulaGraph.

## CASE

The CASE expression uses conditions to filter the parameters. nGQL provides two forms of CASE expressions just like openCypher: the simple form and the generic form.

The CASE expression will traverse all the conditions. When the first condition is met, the CASE expression stops reading the conditions and returns the result. If no conditions are met, it returns the result in the ELSE clause. If there is no ELSE clause and no conditions are met, it returns NULL.

THE SIMPLE FORM OF CASE EXPRESSIONS

```
• Syntax
```

```
CASE <comparer>
WHEN <value> THEN <result>
[WHEN ...]
[ELSE <default>]
END
```

# Caution

Always remember to end the  $\ensuremath{\mathsf{CASE}}$  expression with an  $\ensuremath{\mathsf{END}}$  .

Parameter	Description
comparer	A value or a valid expression that outputs a value. This value is used to compare with the value .
value	It will be compared with the comparer . If the value matches the comparer , then this condition is met.
result	The result is returned by the CASE expression if the value matches the comparer.
default	The default is returned by the CASE expression if no conditions are met.

#### • Examples

nebula> RETURN \ CASE 2+3 \ WHEN 4 THEN 0 \ WHEN 5 THEN 1 \ ELSE -1 \ END \ AS result; ++
result
++
1
++
nebula> 60 FROM "player100" OVER follow \ YIELD properties(\$\$).name AS Name, \ CASE properties(\$\$).age > 35 \ WHEN true THEN "Yes" \ WHEN false THEN "No" \ ELSE "Nah" \ END \ AS Age_above_35;
++

Name	Age_above_35
"Tony Parker"	"Yes"
"Manu Ginobili"	"Yes"
+	+

#### THE GENERIC FORM OF CASE EXPRESSIONS

• Syntax

CASE WHEN <condition> THEN <result> [WHEN ...] [ELSE <default>] END

Parameter	Description
condition	If the condition is evaluated as true, the result is returned by the CASE expression.
result	The result is returned by the CASE expression if the condition is evaluated as true.
default	The default is returned by the CASE expression if no conditions are met.

• Examples

+   resul: +   1 +	YIELD CASE W WHEN 3 ELSE 2 END \ AS res + t   +	\ WHEN 4 > 5 <sup>-</sup> B+4==7 THEN 2 \ sult;	THEN O 1 \	\	
nobulas	матси	(venlavor)			

nebu ta~	RETURN v.playe CASE \ WHEN v.player.	er.name AS Name,	"T" THEN "Yes"
	ELSE "No" \		
	END \		
	AS Starts_with	ι_T;	
+		++	+
Name		Starts_with_T	
+		++	F
"Tim	Duncan"	"Yes"	
"LaMa	rcus Aldridge"	"No"	
"Tony	Parker"	"Yes"	
+		+	L .

DIFFERENCES BETWEEN THE SIMPLE FORM AND THE GENERIC FORM

200 > 20 \

To avoid the misuse of the simple form and the generic form, it is important to understand their differences. The following example can help explain them.

nebula> GO FROM "player100" OVER follow \	
YIELD properties(\$\$).name AS Name, proper	ies(\$\$).age AS Age, ∖
CASE properties(\$\$).age \	
WHEN properties(\$\$).age > 35 THEN "Yes" \	
ELSE "No" \	
END \	
AS Age_above_35;	
++	
Name Age Age_above_35	
++	
"Tony Parker"   36   "No"	
"Manu Ginobili"   41   "No"	
++	

The preceding 60 query is intended to output Yes when the player's age is above 35. However, in this example, when the player's age is 36, the actual output is not as expected: It is No instead of Yes.

This is because the query uses the CASE expression in the simple form, and a comparison between the values of \$\$.player.age and \$\$.player.age > 35 is made. When the player age is 36:

- The value of \$\$.player.age is 36. It is an integer.
- \$\$.player.age > 35 is evaluated to be true. It is a boolean.

The values of \$\$.player.age and \$\$.player.age > 35 do not match. Therefore, the condition is not met and No is returned.

# coalesce()

coalesce() returns the first not null value in all expressions.

Syntax: coalesce(<expression\_1>[,<expression\_2>...])

# • Result type: Same as the original element.

# Example:

<pre>nebula&gt; RETURN coalesce(null,[1,2,3]) as result;</pre>
++
result
++
[1, 2, 3]
++
<pre>nebula&gt; RETURN coalesce(null) as result;</pre>
++
result
++
NULL
++

# 4.4.9 Predicate functions

Predicate functions return true or false. They are most commonly used in WHERE clauses.

NebulaGraph supports the following predicate functions:

Functions	Description
exists()	Returns true if the specified property exists in the vertex, edge or map. Otherwise, returns false.
any()	Returns true if the specified predicate holds for at least one element in the given list. Otherwise, returns false .
all()	Returns true if the specified predicate holds for all elements in the given list. Otherwise, returns false.
none()	Returns true if the specified predicate holds for no element in the given list. Otherwise, returns false.
single()	Returns true if the specified predicate holds for exactly one of the elements in the given list. Otherwise, returns false.

# Note

NULL is returned if the list is NULL or all of its elements are NULL.

# Empatibility

In openCypher, only function exists() is defined and specified. The other functions are implement-dependent.

## Syntax

<predicate>(<variable> IN <list> WHERE <condition>)

#### Examples

```
nebula> RETURN any(n IN [1, 2, 3, 4, 5, NULL] \setminus
         WHERE n > 2) AS r;
| r |
+----+
| true |
+-----
nebula> RETURN single(n IN range(1, 5) \setminus
         WHERE n = 3) AS r;
+----
| r |
| true |
nebula> RETURN none(n IN range(1, 3) \setminus
         WHERE n == 0) AS r;
+----+
| r |
+----
| true |
nebula> WITH [1, 2, 3, 4, 5, NULL] AS a \backslash RETURN any(n IN a WHERE n > 2);
| any(n IN a WHERE (n>2)) |
| true
nebula> MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \</pre>
         RETURN nodes(p)[0].player.name AS n1, nodes(p)[1].player.name AS n2, \
all(n IN nodes(p) WHERE n.player.name NOT STARTS WITH "D") AS b;
                                             --+---
                                                     ---+
```

n1	n2	b	
	"Danny Green" "Dejounte Murray" "Chris Paul" "Kyrie Irving" "Carmelo Anthony" "Dwyane Wade"	+   false   true   true   true   false	+         
++ nebula> MATCH p = RETURN sir ++   b   ++   true   ++	(n:player{name:"LeB ggle(n IN nodes(p) W	+ ron Jame HERE n.p	+ s"})-[:follow]->(m) \ layer.age > 40) AS b;
nebula> MATCH (n:p RETURN exi	olayer)∖ ists(n.player.id), n +	IS NOT -+	NULL;
exists(n.player.	id)   n IS NOT NULL	1	
+   false 	true	-+	
nebula> MATCH (n:p WHERE exis	olayer) \ sts(n['name']) RETUR	N n;	
+			
("Grant Hill" :p   ("Marc Gasol" :p	olayer{age: 46, name olayer{age: 34, name	: "Grant : "Marc	Hill"}) Gasol"})
+			

# 4.4.10 Geography functions

Geography functions are used to generate or perform operations on the value of the geography data type.

For descriptions of the geography data types, see Geography.

# Descriptions

Function	Return Type	Description
ST_Point(longitude, latitude)	GEOGRAPHY	Creates the geography that contains a point.
ST_GeogFromText(wkt_string)	GEOGRAPHY	Returns the geography corresponding to the input WKT string.
ST_ASText(geography)	STRING	Returns the WKT string of the input geography.
ST_Centroid(geography)	GEOGRAPHY	Returns the centroid of the input geography in the form of the single point geography.
ST_ISValid(geography)	BOOL	Returns whether the input geography is valid.
ST_Intersects(geography_1, geography_2)	BOOL	Returns whether geography_1 and geography_2 have intersections.
ST_Covers(geography_1, geography_2)	BOOL	Returns whether geography_1 completely contains geography_2. If there is no point outside geography_1 in geography_2, return True.
ST_CoveredBy(geography_1, geography_2)	BOOL	Returns whether geography_2 completely contains geography_1.If there is no point outside geography_2 in geography_1, return True.
ST_DWithin(geography_1, geography_2, distance)	BOOL	If the distance between one point (at least) in geography_1 and one point in geography_2 is less than or equal to the distance specified by the distance parameter (measured by meters), return True.
ST_Distance(geography_1, geography_2)	FLOAT	Returns the smallest possible distance (measured by meters) between two non-empty geographies.
S2_CellIdFromPoint(point_geography)	INT	Returns the S2 Cell ID that covers the point geography.
S2_CoveringCellIds(geography)	ARRAY <int64></int64>	Returns an array of S2 Cell IDs that cover the input geography.

# Examples

++   ST_ASText(ST_Point(1,1))   ++   "POINT(1 1)"   ++
nebula> RETURN ST_ASText(ST_GeogFromText("POINT(3 8)")); +
<pre>nebula&gt; RETURN ST_ASTEXT(ST_Centroid(ST_GeogFromText("LineString(0 1,1 0)"))); ++   ST_ASTEXT(ST_Centroid(ST_GeogFromText("LineString(0 1,1 0)")))   ++   "POINT(0.5000380800773782 0.5000190382261059)"  </pre>

nebula> RETURN ST\_ISValid(ST\_GeogFromText("POINT(3 8)")); | ST\_ISValid(ST\_GeogFromText("POINT(3 8)")) | true nebula> RETURN ST\_Intersects(ST\_GeogFromText("LineString(0 1,1 0)"),ST\_GeogFromText("LineString(0 0,1 1)")); | ST\_Intersects(ST\_GeogFromText("LineString(0 1,1 0)"),ST\_GeogFromText("LineString(0 0,1 1)")) | | true nebula> RETURN ST\_Covers(ST\_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))"),ST\_Point(1,2)); | ST\_Covers(ST\_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))"),ST\_Point(1,2)) | | true nebula> RETURN ST\_CoveredBy(ST\_Point(1,2),ST\_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))")); | ST\_CoveredBy(ST\_Point(1,2),ST\_GeogFromText("POLYGON((0 0,10 0,10 10,0 10,0 0))")) | | true nebula> RETURN ST\_dwithin(ST\_GeogFromText("Point(0 0)"),ST\_GeogFromText("Point(10 10)"),20000000000.0); | ST\_dwithin(ST\_GeogFromText("Point(0 0)"),ST\_GeogFromText("Point(10 10)"),2000000000) | | true nebula> RETURN ST\_Distance(ST\_GeogFromText("Point(0 0)"),ST\_GeogFromText("Point(10 10)")); | ST\_Distance(ST\_GeogFromText("Point(0 0)"),ST\_GeogFromText("Point(10 10)")) | 1.5685230187677438e+06 nebula> RETURN S2\_CellIdFromPoint(ST\_GeogFromText("Point(1 1)")); | S2\_CellIdFromPoint(ST\_GeogFromText("Point(1 1)")) | | 1153277837650709461 nebula> RETURN S2\_CoveringCellIds(ST\_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))")); | S2\_CoveringCellIds(ST\_GeogFromText("POLYGON((0 1, 1 2, 2 3, 0 1))")) [1152391494368201343, 1153466862374223872, 1153554823304445952, 1153856298281156608, 1153959443583467520, 1154240918560178176, 1160503736791990272, 1160591697722212352]

# 4.5 General queries statements

## 4.5.1 Overview of NebulaGraph general query statements

This topic provides an overview of the general categories of query statements in NebulaGraph and outlines their use cases.

#### Background

NebulaGraph stores data in the form of vertices and edges. Each vertex can have zero or more tags and each edge has exactly one edge type. Tags define the type of a vertex and describe its properties, while edge types define the type of an edge and describe its properties. When querying, you can limit the scope of the query by specifying the tag of a vertex or the type of an edge. For more information, see Patterns.

#### Categories

The primary query statements in NebulaGraph fall into the following categories:

- FETCH PROP ON
- LOOKUP ON
- GO
- MATCH
- FIND PATH
- GET SUBGRAPH
- SHOW

FETCH PROP ON and LOOKUP ON statements are primarily for basic data queries, GO and MATCH for more intricate queries and graph traversals, FIND PATH and GET SUBGRAPH for path and subgraph queries, and SHOW for retrieving database metadata.

#### Usage and use cases

FETCH PROP ON

Usage: Retrieve properties of a specified vertex or edge.

Use case: Knowing the specific vertex or edge ID and wanting to retrieve its properties.

#### Note:

- Must specify the ID of the vertex or edge.
- Must specify the tag of the vertex or the edge type of the edge.
- Must use the YIELD clause to specify the returned properties.

## Example:



For more information, see FETCH PROP ON.

# LOOKUP ON

Usage: Index-based querying of vertex or edge IDs.

Use case: Finding vertex or edge IDs based on property values.

**Note:** - Must pre-define indexes for the tag, edge type, or property. - Must specify the tag of the vertex or the edge type of the edge. - Must use the YIELD clause to specify the returned IDs.

#### **Example:**



For more information, see LOOKUP ON.

GO

**Usage:** Traverse the graph based on a given vertex and return information about the starting vertex, edges, or target vertices as needed. **Use case:** Complex graph traversals, such as finding friends of a vertex, friends' friends, etc.

**Note:** - Use property reference symbols (\$^ and \$\$) to return properties of the starting or target vertices, e.g., YIELD \$^.player.name. - Use the functions properties(\$^) and properties(\$\$) to return all properties of the starting or target vertices. Specify property names in the function to return specific properties, e.g., YIELD properties(\$^).name. - Use the functions src(edge) and dst(edge) to return the starting or destination vertex ID of an edge, e.g., YIELD src(edge).

#### Example:



For more information, see GO.

МАТСН

Usage: Execute complex graph pattern matching queries.

Use case: Complex graph pattern matching, such as finding combinations of vertices and edges that satisfy a specific pattern.

#### Note:

MATCH statements are compatible with the OpenCypher syntax but with some differences:

- Use == for equality instead of =, e.g., WHERE player.name == "Tony Parker".
- When referencing properties of vertices, you need to specify the vertex's tag, e.g., YIELD player.name.
- Introduces the WHERE id(v) == "player100" syntax.
- Must use the RETURN clause to specify what information to return.

#### Example:

```
MATCH (v:player{name:"Tim Duncan"})-->(v2:player) \
RETURN v2.player.name AS Name;
```

For more information, see MATCH.

FIND PATH

Usage: Query paths between given starting and target vertices or query properties of vertices and edges along paths.

# Use case: Querying paths between two vertices.

Note: Must use the YIELD clause to specify returned information.

## Example:



## For more information, see FIND PATH.

#### GET SUBGRAPH

Usage: Extract a portion of the graph that satisfies specific conditions or query properties of vertices and edges in the subgraph.

**Use case:** Analyzing structures of the graph or specific regions, such as extracting the social network subgraph of a person or the transportation network subgraph of an area.

Note: Must use the YIELD clause to specify returned information.

#### Example:

GET SUBGRAPH 5 STEPS FROM "player101" YIELD VERTICES AS nodes, EDG	ES AS relationships;
++	+
+ Starts from "player101".	+ Returns all vertices and edges.
+ Gets exploration of 5 steps	

For more information, see GET SUBGRAPH.

#### SHOW

SHOW statements are mainly used to obtain metadata information from the database, not for retrieving the actual data stored in the database. These statements are typically used to query the structure and configuration of the database.

Statement	Syntax	Example	Description
SHOW CHARSET	SHOW CHARSET	SHOW CHARSET	Shows the available character sets.
SHOW COLLATION	SHOW COLLATION	SHOW COLLATION	Shows the collations supported by NebulaGraph.
SHOW CREATE SPACE	SHOW CREATE SPACE <space_name></space_name>	SHOW CREATE SPACE basketballplayer	Shows the creating statement of the specified graph space.
SHOW CREATE TAG/EDGE	SHOW CREATE {TAG <tag_name>   EDGE <edge_name>}</edge_name></tag_name>	SHOW CREATE TAG player	Shows the basic information of the specified tag.
SHOW HOSTS	SHOW HOSTS [GRAPH   STORAGE   META]	SHOW HOSTS SHOW HOSTS GRAPH	Shows the host and version information of Graph Service, Storage Service, and Meta Service.
SHOW INDEX STATUS	SHOW {TAG   EDGE} INDEX STATUS	SHOW TAG INDEX STATUS	Shows the status of jobs that rebuild native indexes, which helps check whether a native index is successfully rebuilt or not.
SHOW INDEXES	SHOW {TAG   EDGE} INDEXES	SHOW TAG INDEXES	Shows the names of existing native indexes.
SHOW PARTS	SHOW PARTS [ <part_id>]</part_id>	SHOW PARTS	Shows the information of a specified partition or all partitions in a graph space.
SHOW ROLES	SHOW ROLES IN <space_name></space_name>	SHOW ROLES in basketballplayer	Shows the roles that are assigned to a user account.
SHOW SNAPSHOTS	SHOW SNAPSHOTS	SHOW SNAPSHOTS	Shows the information of all the snapshots.
SHOW SPACES	SHOW SPACES	SHOW SPACES	Shows existing graph spaces in NebulaGraph.
SHOW STATS	SHOW STATS	SHOW STATS	Shows the statistics of the graph space collected by the latest STATS job.
SHOW TAGS/ EDGES	SHOW TAGS   EDGES	SHOW TAGS, SHOW EDGES	Shows all the tags in the current graph space.
SHOW USERS	SHOW USERS	SHOW USERS	Shows the user information.
SHOW SESSIONS	SHOW SESSIONS	SHOW SESSIONS	Shows the information of all the sessions.
SHOW SESSIONS	SHOW SESSION <session_id></session_id>	SHOW SESSION 1623304491050858	Shows a specified session with its ID.
SHOW QUERIES	SHOW [ALL] QUERIES	SHOW QUERIES	Shows the information of working queries in the current session.
SHOW META LEADER	SHOW META LEADER	SHOW META LEADER	Shows the information of the leader in the current Meta cluster.

# **Compound queries**

Query statements in NebulaGraph can be combined to achieve more complex queries.

When referencing the results of a subquery in a compound statement, you need to create an alias for the result and use the pipe symbol() to pass it to the next subquery. Use \$- in the next subquery to reference the alias of that result. See Pipe Symbol for details.

# Example:

The pipe symbol || is applicable only in nGQL and cannot be used in OpenCypher statements. If you need to perform compound queries using MATCH statements, you can use the WITH clause.

Example:

## More information

nGQL command cheatsheet

Last update: January 31, 2024

## 4.5.2 MATCH

The MATCH statement provides pattern-based search functionality, allowing you to retrieve data that matches one or more patterns in NebulaGraph. By defining one or more patterns, you can search for data that matches the patterns in NebulaGraph. Once the matching data is retrieved, you can use the RETURN clause to return it as a result.

The examples in this topic use the basketballplayer dataset as the sample dataset.

#### Syntax

The syntax of MATCH is relatively more flexible compared with that of other query statements such as 60 or LOOKUP. The path type of the MATCH statement is trait. That is, only vertices can be repeatedly visited in the graph traversal. Edges cannot be repeatedly visited. For details, see path. But generally, it can be summarized as follows.

MATCH <pattern> [<clause\_1>] RETURN <output> [<clause\_2>];

- pattern: The MATCH statement supports matching one or multiple patterns. Multiple patterns are separated by commas (,). For example: (a)-[]->(b),(c)-[]->(d). For the detailed description of patterns, see Patterns.
- clause\_1: The WHERE, WITH, UNWIND, and OPTIONAL MATCH clauses are supported, and the MATCH clause can also be used.
- output : Define the list name for the output results to be returned. You can use AS to set an alias for the list.
- clause\_2: The ORDER BY and LIMIT clauses are supported.

# **P**Jacy version compatibility

• Starting from version 3.5.0, the MATCH statement supports full table scans. It can traverse vertices or edges in the graph without using any indexes or filter conditions. In previous versions, the MATCH statement required an index for certain queries or needed to use LIMIT to restrict the number of output results.

• Starting from NebulaGraph version 3.0.0, in order to distinguish the properties of different tags, you need to specify a tag name when querying properties. The original statement RETURN <variable\_name>.<property\_name> is changed to RETURN <variable\_name>.<property\_name> .

#### Notes

- Avoid full table scans, as they may result in decreased query performance, and if there is insufficient memory during a full table scan, the query may fail, and the system will report an error. It is recommended to use queries with filter conditions or specifying tags and edge types, such as v:player and v.player.name in the statement MATCH (v:player) RETURN v.player.name AS Name.
- You can create an index for a tag, edge type, or a specific property of a tag or edge type to improve query performance. For example, you can create an index for the player tag or the name property of the player tag. For more information about the usage and considerations for indexes, see Must-read for using indexes.
- The MATCH statement cannot query dangling edges.

#### Using patterns in MATCH statements

#### MATCH VERTICES

You can use a user-defined variable in a pair of parentheses to represent a vertex in a pattern. For example: (v).

nebula≻ MATCH RETURN LIMIT ∶	(v) \ v \ 3;			
+				+
V				
+				+
("player102"	:player{age:	33, name:	"LaMarcus Aldridge'	'})
("player106"	:player{age:	25, name:	"Kyle Anderson"})	

| ("player115" :player{age: 40, name: "Kobe Bryant"}) |

#### MATCH TAGS

# L Jacy version compatibility

• In NebulaGraph versions earlier than 3.0.0, the prerequisite for matching a tag is that the tag itself has an index or a certain property of the tag has an index.

• Starting from NebulaGraph 3.0.0, you can match tags without creating an index, but you need to use LIMIT to restrict the number of output results.

Starting from NebulaGraph 3.5.0, the MATCH statement supports full table scans. There is no need to create an index for a tag or a specific property of a tag, nor use LIMIT to restrict the number of output results in order to execute the MATCH statement.

You can specify a tag with :<tag\_name> after the vertex in a pattern.

nebu	ula> MATCH ( RETURN	[v:player) ∖ v;						
+								-+
v								I
+								·+
('	"player105"	:player{age:	31,	name:	"Danny	Green"})		
('	"player109"	:player{age:	34,	name:	"Tiago	<pre>Splitter"})</pre>	)	1
('	"player111"	:player{age:	38,	name:	"David	West"})		

To match vertices with multiple tags, use colons (:).

nebula> CREATE TAG actor (name string, age int); nebula> INSERT VERTEX actor(name, age) VALUES "player100":("Tim Duncan", 42); nebula> MATCH (v:player:actor) \ RETURN v \
++
V
++
("player100" :actor{age: 42, name: "Tim Duncan"} :player{age: 42, name: "Tim Duncan"})

MATCH VERTEX PROPERTIES

#### Q Note

The prerequisite for matching a vertex property is that the tag itself has an index of the corresponding property. Otherwise, you cannot execute the MATCH statement to match the property.

You can specify a vertex property with {<prop\_name>: <prop\_value>} after the tag in a pattern.

<pre># The following example uses the name property to match a vertex. nebula&gt; MATCH (v:player{name:"Tim Duncan"}) \</pre>
++
v
++
("player100" :player{age: 42, name: "Tim Duncan"})
++

#### The WHERE clause can do the same thing:

nebula> MATCH (v:player) \ WHERE v.player.name == "Tim Duncan" \ RETURN v;
++
v
++
("player100" :player{age: 42, name: "Tim Duncan"})
++
# OpenCypher compatibility

```
In openCypher 9, = is the equality operator. However, in nGQL, == is the equality operator and = is the assignment operator (as in C++ or Java).
```

Use the WHERE clause to directly get all the vertices with the vertex property value Tim Duncan.

<pre>nebula&gt; MATCH (v) \     WITH v, properties(v) as props, keys(properties(v)) as kk \     WHERE [i in kk where props[i] == "Tim Duncan"] \     RETURN v; ++   v</pre>			
("player100" :player{age: 42, name: "Tim Duncan"})			
*	+		
nebula> WITH ['Tim Duncan', 'Yao Ming'] AS names \ MATCH (v1:player)>(v2:player) \ WHERE v1.player.name in names \ return v1, v2;			
v1	v2		
<pre>("player133" :player{age: 38, name: "Yao Ming"}) ("player133" :player{age: 38, name: "Yao Ming"}) ("player100" :player{age: 42, name: "Tim Duncan"}) ("player100" :player{age: 42, name: "Tim Duncan"})</pre>	<pre>("player114" :player{age: 39, name: "Tracy McGrady"}) ( ("player144" :player{age: 47, name: "Shaquille 0'Neal"}) ( "player101" :player{age: 36, name: "Tony Parker"}) ( "player125" :player{age: 41, name: "Manu Ginobili"})</pre>		

#### MATCH VIDS

You can use the VID to match a vertex. The id() function can retrieve the VID of a vertex.

To match multiple VIDs, use where id(v) IN [vid\_list] or where id(v) IN {vid\_list} .

<pre>nebula&gt; MATCH (v:player { name: 'Tim Duncan' })(v2) \ WHERE id(v2) IN ["player101", "player102"] \ RETURN v2;</pre>		
++		
v2		
++		
("player101" :player{age: 36, name: "Tony Parker"})		
("player101" :player{age: 36, name: "Tony Parker"})		
("player102" :player{age: 33, name: "LaMarcus Aldridge"})		
++		
nebula> MATCH (v) WHERE id(v) IN {"player100", "player101"} $\setminus$		
RETURN v.player.name AS name;		
++		
name		
++		
"Tony Parker"		
"Tim Duncan"		
++		

MATCH CONNECTED VERTICES

You can use the -- symbol to represent edges of both directions and match vertices connected by these edges.

# P\_jacy version compatibility

In nGQL 1.x, the -- symbol is used for inline comments. Starting from nGQL 2.x, the -- symbol represents an incoming or outgoing edge.

+----

You can add a > or < to the -- symbol to specify the direction of an edge.

In the following example,  $\rightarrow$  represents an edge that starts from v and points to v2. To v, this is an outgoing edge, and to v2 this is an incoming edge.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2:player) \
RETURN v2.player.name AS Name;
+-----+
| Name |
+-----+
| "Manu Ginobili" |
| "Tony Parker" |
```

To query the properties of the target vertices, use the CASE expression.

To extend the pattern, you can add more vertices and edges.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2)<--(v3) \
RETURN v3.player.name AS Name;
+------+
| Name |
'------+
| "Dejounte Murray" |
| "LaMarcus Aldridge" |
| "Warco Belinelli" |
```

If you do not need to refer to a vertex, you can omit the variable representing it in the parentheses.

MATCH PATHS

Connected vertices and edges form a path. You can use a user-defined variable to name a path as follows.

₽. enCypher compatibility

In nGQL, the @ symbol represents the rank of an edge, but openCypher has no such concept.

MATCH EDGES

nebula> MATCH ()<-[e]-() \ RETURN e \ LIMIT 3;				
+				+
e				ļ
<pre>  [:follow "player101"-&gt;"player102"   [:follow "player103"-&gt;"player102"   [:follow "player135"-&gt;"player102" +</pre>	09 09 09	{degree: {degree: {degree:	90}] 70}] 80}]	

MATCH EDGE TYPES

Just like vertices, you can specify edge types with :-edge\_type> in a pattern. For example: -[e:follow]-.

# **P**enCypher compatibility

• In NebulaGraph versions earlier than 3.0.0, the prerequisite for matching a edge type is that the edge type itself has an index or a certain property of the edge type has an index.

Starting from version 3.0.0, there is no need to create an index for matching a edge type, but you need to use LIMIT to limit the number of output results and you must specify the direction of the edge.

• Starting from NebulaGraph 3.5.0, you can use the MATCH statement to match edges without creating an index for edge type or using LIMIT to restrict the number of output results.



MATCH EDGE TYPE PROPERTIES

# Note

The prerequisite for matching an edge type property is that the edge type itself has an index of the corresponding property. Otherwise, you cannot execute the MATCH statement to match the property.

You can specify edge type properties with {<prop\_name>: <prop\_value>} in a pattern. For example: [e:follow{likeness:95}].

nebula>	MATCH (v:player{name:"Tim Duncan"})-	<pre>[e:follow{degree:95}]-&gt;(v2)</pre>
	REIURN e;	
+		+
e		
+		+
[:fol	low "player100"->"player101" @0 {degr	ee: 95}]
[:fol	low "player100"->"player125" @O {degr	ee: 95}]
+		+

Use the WHERE clause to directly get all the edges with the edge property value 90.

nebula>	MATCH ()-[e]->() \
	WITH e, properties(e) as props, keys(properties(e)) as kk \
	WHERE [i in kk where props[i] == 90] \
	RETURN e;
+	+
e e	
+	+

	[:follow	"player125"->"player100"	@0	{degree:	90}]	1
	[:follow	"player140"->"player114"	@0	{degree:	90}]	1
	[:follow	"player133"->"player144"	@0	{degree:	90}]	1
	[:follow	"player133"->"player114"	@0	{degree:	90}]	
+.						+

MATCH MULTIPLE EDGE TYPES

The | symbol can help matching multiple edge types. For example: [e:follow|:serve]. The English colon (:) before the first edge type cannot be omitted, but the English colon before the subsequent edge type can be omitted, such as [e:follow|serve].

nebula> MATCH (v:player{name:"Tim Duncan"})-[e:follow :serve]->(v2) \ RETURN e;
++
e
++
[:follow "player100"->"player101" @0 {degree: 95}]
[:follow "player100"->"player125" @0 {degree: 95}]
[:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]
++

MATCH MULTIPLE EDGES

You can extend a pattern to match multiple edges in a path.

nebula> MATCH (v:player{name:"Tim Duncan"})-[]->(v2)<-[e:serve]-(v3) \ RETURN v2, v3;				
v2	v3			
<pre>  ("team204" :team{name: "Spurs"})   ("team204" :team{name: "Spurs"})   ("team204" :team{name: "Spurs"})</pre>	<pre>("player104" :player{age: 32, name: "Marco Belinelli"}) ("player101" :player{age: 36, name: "Tony Parker"}) ("player102" :player{age: 33, name: "LaMarcus Aldridge"})</pre>			

MATCH FIXED-LENGTH PATHS

You can use the :<edge\_type>\*<hop> pattern to match a fixed-length path. hop must be a non-negative integer.

If hop is 0, the pattern will match the source vertex of the path.

<pre>nebula&gt; MATCH (v:player{name:"Tim Duncan"}) -[*0]-&gt; (v2)</pre>	) \
++	
v2	
++	
("player100" :player{age: 42, name: "Tim Duncan"})	
++	

#### Q Note

When you conditionally filter on multi-hop edges, such as -[e:follow\*2]->, note that the e is a list of edges instead of a single edge.

For example, the following statement is correct from the syntax point of view which may not get your expected query result, because the e is a list without the .degree property.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
WHERE e.degree > 1 \
RETURN DISTINCT v2 AS Friends;
```

The correct statement is as follows:

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
WHERE ALL(e_ in e WHERE e_.degree > 0) \
RETURN DISTINCT v2 AS Friends;
```

Further, the following statement is for filtering the properties of the first-hop edge in multi-hop edges:

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
WHERE e[0].degree > 98 \
RETURN DISTINCT v2 AS Friends;
```

MATCH VARIABLE-LENGTH PATHS

You can use the :<edge\_type>\*[minHop..maxHop] pattern to match variable-length paths. minHop and maxHop are optional and default to 1 and infinity respectively.

Caution			
If maxHop is not set, it may cause the Graph service to OOM. Execute this command with caution.			
Parameter	Description		
minHop	Optional. minHop indicates the minimum length of the path, which must be a non-negative integer. The default value is 1.		
тахНор	Optional. maxHop indicates the maximum length of the path, which must be a non-negative integer. The default value is infinity.		

If neither minHop nor maxHop is specified, and only :<edge\_type>\* is set, the default values are applied to both, i.e., minHop is 1 and maxHop is infinity.

You can use the DISTINCT keyword to aggregate duplicate results.

<pre>nebula&gt; MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1 RETURN DISTINCT v2 AS Friends, count(v2); *-</pre>	3]->(v2:player)
Friends	count(y2)
+	-++
("plaver102" :plaver{age: 33. name: "LaMarcus Aldridge"})	
<pre>("player100" :player{age: 42, name: "Tim Duncan"})</pre>	4
("player101" :player{age: 36, name: "Tony Parker"})	3
("player125" :player{age: 41, name: "Manu Ginobili"})	3
+	-++

If minHop is 0, the pattern will match the source vertex of the path. Compared to the preceding statement, the following example uses 0 as the minHop. So in the following result set, "Tim Duncan" is counted one more time than it is in the preceding result set because it is the source vertex.

#### Q Note

When using the variable e to match fixed-length or variable-length paths in a pattern, such as -[e:follow\*0..3]->, it is not supported to reference e in other patterns. For example, the following statement is not supported.

```
nebula> MATCH (v:player)-[e:like*1..3]->(n) \
    WHERE (n)-[e*1..4]->(:player) \
    RETURN v;
```

MATCH VARIABLE-LENGTH PATHS WITH MULTIPLE EDGE TYPES

You can specify multiple edge types in a fixed-length or variable-length pattern. In this case, hop, minHop, and maxHop take effect on all edge types.

MATCH MULTIPLE PATTERNS

#### You can separate multiple patterns with commas (,).

<pre>nebula&gt; CREATE TAG INDEX IF NOT EXISTS team_index ON team(name(20)); nebula&gt; REBUILD TAG INDEX team_index; nebula&gt; MATCH (v1:player{name:"Tim Duncan"}), (v2:team{name:"Spurs"}) \ RETURN v1,v2;</pre>		
+	-++   v2   -++	
("player100" :player{age: 42, name: "Tim Duncan"})   ("team204" :team{name: "Spurs"})   ++		

MATCH SHORTEST PATHS

The allShortestPaths function can be used to find all shortest paths between two vertices.

<pre>nebula&gt; MATCH p = allShortestPaths((a:player{name:"Tim Duncan"})-[e*5]-(b:player{name:"Tony Parker"})) \</pre>	
RETURN p; +	+
p	

I	<("player100"	:player{age:	42,	name:	"Tim Duncan"})<-[:follow@0	{degree:	: 95}]-("player101"	:player{age:	36,	name:	"Tony	Parker"})>	1
I	<("player100"	:player{age:	42,	name:	"Tim Duncan"})-[:follow@0	{degree:	95}]->("player101"	:player{age:	36,	name:	"Tony	Parker"})>	
+													+

The shortestPath function can be used to find a single shortest path between two vertices.

<pre>nebula&gt; MATCH p = shortestPath((a:player{name:"Tim Duncan"})-[e*5]-(b:player{name:"Tony Parker"})) \</pre>	
+	+
p	
+	+
<("player100" :player{age: 42, name: "Tim Duncan"})<-[:follow@0 {degree: 95}]-("player101" :player{age: 36, name:	"Tony Parker"})>
+	+

# Retrieve with multiple match

Multiple MATCH can be used when different patterns have different filtering criteria and return the rows that exactly match the pattern.

nebula>	MATCH MATCH RETURN	(m)-[]->(n) (n)-[]->(l) id(m),id(n)	WHERE WHERE ),id(l	id(m)=="pl id(n)=="pl );	ayer100" ayer125"
+		+	+		+
id(m)		id(n)	i	d(l)	[
"playe   "playe	er100" er100"	+12   "player12   "player12 +	5"   " 5"   " 5"   "	team204" player100"	-+     +

# Retrieve with optional match

See OPTIONAL MATCH.

# Caution

In NebulaGraph, the performance and resource usage of the MATCH statement have been optimized. But we still recommend to use 60, LOOKUP, |, and FETCH instead of MATCH when high performance is required.

Last update: December 5, 2023

# 4.5.3 OPTIONAL MATCH

# Caution

The feature is still in beta. It will continue to be optimized.

The OPTIONAL MATCH clause is used to search for the pattern described in it. OPTIONAL MATCH matches patterns against your graph database, just like MATCH does. The difference is that if no matches are found, OPTIONAL MATCH will use a null for missing parts of the pattern.

# **OpenCypher Compatibility**

This topic applies to the openCypher syntax in nGQL only.

## Limitations

The WHERE clause cannot be used in an OPTIONAL MATCH clause.

# Example

The example of the use of <code>OPTIONAL MATCH</code> in the <code>MATCH</code> statement is as follows:

nebula> MATCH OPTION RETURN	(m)-[]->(n) WH AL MATCH (n)-[ id(m),id(n),i	ERE id(m)=="p ]->(l) \ d(l);	layer100" \
+	+	+	-+
id(m)	id(n)	id(l)	
+	+	+	-+
"player100"	"team204"	NULL	1
"player100"	player101"	"team204"	1
player100"	player101"	"team215"	i i
player100"	player101	player100"	i i
"player100"	player101"	"player102"	1
player100"	player101"	player125"	i i
"player100"	"player125"	"team204"	i i
"player100"	player125	"player100"	1
1			

Using multiple MATCH instead of OPTIONAL MATCH returns rows that match the pattern exactly. The example is as follows:

nebula> MATCH MATCH RETURN	(m)-[]->(n) WH (n)-[]->(l) \ V id(m),id(n),i	ERE id(m)=="pl d(l);	ayer100" \.
+	+	+	+
id(m)	id(n)	id(l)	1
+	+	+	+
player100"	player101"	"team204"	1
player100"	player101"	"team215"	1
"player100"	player101"	"player100"	
player100"	player101"	"player102"	1
"player100"	"player101"	player125"	1
"player100"	"player125"	"team204"	Í
player100"	player125"	"player100"	1
+	+	+	+

# 4.5.4 LOOKUP

The LOOKUP statement traverses data based on indexes. You can use LOOKUP for the following purposes:

- $\bullet$  Search for the specific data based on conditions defined by the  ${\tt WHERE}$  clause.
- List vertices with a tag: retrieve the VID of all vertices with a tag.
- List edges with an edge type: retrieve the source vertex IDs, destination vertex IDs, and ranks of all edges with an edge type.
- Count the number of vertices or edges with a tag or an edge type.

#### OpenCypher compatibility

This topic applies to native nGQL only.

#### Precautions

- Correct use of indexes can speed up queries, but indexes can dramatically reduce the write performance. The performance can be greatly reduced. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.
- If the specified property is not indexed when using the LOOKUP statement, NebulaGraph randomly selects one of the available indexes.

For example, the tag player has two properties, name and age. Both the tag player itself and the property name have indexes, but the property age has no indexes. When running LOOKUP ON player WHERE player.age == 36 YIELD player.name; , NebulaGraph randomly uses one of the indexes of the tag player and the property name. You can use the EXPLAIN statement to check the selected index.

# LJacy version compatibility

Before the release 2.5.0, if the specified property is not indexed when using the LOOKUP statement, NebulaGraph reports an error and does not use other indexes.

#### Prerequisites

Before using the LOOKUP statement, make sure that at least one index is created. If there are already related vertices, edges, or properties before an index is created, the user must rebuild the index after creating the index to make it valid.

#### Syntax

```
LOOKUP ON {<vertex_tag> | <edge_type>}
[WHERE <expression> [AND <expression> ...]]
YIELD [DISTINCT] <return_list> [AS <alias>];
<return_list>
<prop_name> [AS <col_alias>] [, <prop_name> [AS <prop_alias>] ...];
```

- WHERE <expression> : filters data with specified conditions. Both AND and OR are supported between different expressions. For more information, see WHERE.
- YIELD : Define the output to be returned. For details, see YIELD.
- DISTINCT : Aggregate the output results and return the de-duplicated result set.
- AS : Set an alias.

#### Limitations of using WHERE in LOOKUP

The WHERE clause in a LOOKUP statement does not support the following operations:

- \$- and \$^.
- Filter rank().
- In relational expressions, operators are not supported to have field names on both sides, such as tagName.prop1> tagName.prop2.
- Nested AliasProp expressions in operation expressions and function expressions are not supported.
- The XOR operation is not supported.
- String operations other than STARTS WITH are not supported.
- Graph patterns.

## **Retrieve vertices**

The following example returns vertices whose name is Tony Parker and the tag is player.

```
nebula> CREATE TAG INDEX IF NOT EXISTS index_player ON player(name(30), age);
nebula> REBUILD TAG INDEX index_player;
| New Job Id |
| 15
+----+
nebula> LOOKUP ON player \
        WHERE player.name == "Tony Parker" \
YIELD id(vertex);
| id(VERTEX)
| "player101"
nebula> LOOKUP ON player \
    WHERE player.name == "Tony Parker" \
        YIELD properties(vertex).name AS name, properties(vertex).age AS age;
| name
                 age
| "Tony Parker" | 36 |
nebula> LOOKUP ON player \
        WHERE player.age > 45 \
YIELD id(vertex);
| id(VERTEX)
 "player144"
 "player140"
nebula> LOOKUP ON player \
WHERE player.name STARTS WITH "B" \
        AND player.age IN [22,30] \
        YIELD properties(vertex).name, properties(vertex).age;
| properties(VERTEX).name | properties(VERTEX).age |
  "Ben Simmons"
                            | 22
 "Blake Griffin"
                            30
nebula> LOOKUP ON player \
         WHERE player.name == "Kobe Bryant"\
        YIELD id(vertex) AS VertexID, properties(vertex).name AS name |\backslash GO FROM $-.VertexID OVER serve \backslash
        YIELD $-.name, properties(edge).start_year, properties(edge).end_year, properties($$).name;
              | properties(EDGE).start_year | properties(EDGE).end_year | properties($$).name |
$-.name
 "Kobe Bryant" | 1996
                                                                                | "Lakers"
                                                 2016
```

# Retrieve edges

The following example returns edges whose  ${\tt degree}$  is 90 and the edge type is  ${\tt follow}\,.$ 

nebula> CREATE EDGE INDEX IF	NOT EXISTS index	<_follow ON follow(degree);							
nebula> REBUILD EDGE INDEX index_follow; ++   New Job Id   ++   62   ++									
nebula> LOOKUP ON follow \ WHERE follow.degree == 90 YIELD edge AS e;									
+   e		+							
+ [:follow "player109"->"player125" @0 {degree: 90}]   [:follow "player118"->"player120" @0 {degree: 90}]   [:follow "player118"->"player131" @0 {degree: 90}]   									
nebula> LOOKUP ON follow \ WHERE follow.degree = YIELD properties(edg	== 90 \ e).degree;								
SrcVID   DstVID	Ranking   prop	erties(EDGE).degree							
"player150"   "player143"   "player150"   "player137"   "player148"   "player136" 	0   90   0   90   0   90								
<pre>nebula&gt; LOOKUP ON follow \ WHERE follow.degree == 60 \ YIELD dst(edge) AS DstVID, properties(edge).degree AS Degree  \ G0 FROM \$DstVID OVER serve \ YIELD \$DstVID, properties(edge).start_year, properties(edge).end_year, properties(\$\$).name;</pre>									
\$DstVID   properties(E	DGE).start_year	properties(EDGE).end_year	properties(\$\$).name						
"player105"   2010   "player105"   2009   "player105"   2009   "player105"   2018		2018 2010 2019	"Spurs"   "Cavaliers"   "Raptors"						

# List vertices or edges with a tag or an edge type

To list vertices or edges with a tag or an edge type, at least one index must exist on the tag, the edge type, or its property.

For example, if there is a player tag with a name property and an age property, to retrieve the VID of all vertices tagged with player, there has to be an index on the player tag itself, the name property, or the age property.

• The following example shows how to retrieve the VID of all vertices tagged with player .

• The following example shows how to retrieve the source Vertex IDs, destination vertex IDs, and ranks of all edges of the follow edge type.

```
nebula> CREATE EDGE IF NOT EXISTS follow(degree int);
nebula> CREATE EDGE INDEX IF NOT EXISTS follow_index on follow();
nebula> REBUILD EDGE INDEX follow_index;
+-------+
| New Job Id |
+------+
| 88 |
+------+
nebula> INSERT EDGE follow(degree) \
VALUES "player100"->"player100":(95);
The following statement retrieves all edges with the edge type `follow`. It is similar to `MATCH (s)-[e:follow]->(d) RETURN id(s), rank(e), id(d) /*, type(e) */`.
nebula> LOOKUP ON follow YIELD edge AS e;
+------+
| e |
+-----+
| [:follow "player105"->"player100" @0 (degree: 70)] |
| [:follow "player105"->"player105" @0 (degree: 80)] |
```

[:follow "player109"->"player100" @0 {degree: 80}]

#### Count the numbers of vertices or edges

The following example shows how to count the number of vertices tagged with player and edges of the follow edge type.

nebula> LOOKUP ON player YIELD id(vertex) \
YIELD COUNT(*) AS Player_Number;
++
Player_Number
++
51
++
nebula> LOOKUP ON follow YIELD edge AS e  $\backslash$
<pre>nebula&gt; LOOKUP ON follow YIELD edge AS e  \     YIELD COUNT(*) AS Follow_Number;</pre>
<pre>nebula&gt; LOOKUP ON follow YIELD edge AS e  \     YIELD COUNT(*) AS Follow_Number; ++</pre>
<pre>nebula&gt; LOOKUP ON follow YIELD edge AS e  \     YIELD COUNT(*) AS Follow_Number; ++   Follow_Number  </pre>
<pre>nebula&gt; LOOKUP ON follow YIELD edge AS e  \</pre>
nebula> LOOKUP ON follow YIELD edge AS e  \ YIELD COUNT(*) AS Follow_Number; ++   Follow_Number   ++   81

Q Note

You can also use  $\ensuremath{\texttt{SHOW}}\xspace$  stats to count the numbers of vertices or edges.

# 4.5.5 GO

The 60 statement is used in the NebulaGraph database to traverse the graph starting from a given starting vertex with specified filters and return results.

## OpenCypher compatibility

This topic applies to native nGQL only.

# Syntax

```
<return_list> ::=
<col_name> [AS <col_alias>] [, <col_name> [AS <col_alias>] ...]
```

• <N> {STEP|STEPS} : specifies the hop number. If not specified, the default value for N is one. When N is zero, NebulaGraph does not traverse any edges and returns nothing.

# Note

The path type of the 60 statement is walk, which means both vertices and edges can be repeatedly visited in graph traversal. For more information, see Path.

- M TO N {STEP|STEPS} : traverses from M to N hops. When M is zero, the output is the same as that of M is one. That is, the output of GO O TO 2 and GO 1 TO 2 are the same.
- <vertex\_list> : represents a list of vertex IDs separated by commas.
- <edge\_type\_list> : represents a list of edge types which the traversal can go through.
- REVERSELY | BIDIRECT : defines the direction of the query. By default, the GO statement searches for outgoing edges of <vertex\_list>. If REVERSELY is set, GO searches for incoming edges. If BIDIRECT is set, GO searches for edges of both directions. The direction of the query can be checked by returning the <edge\_type>.\_type field using YIELD. A positive value indicates an outgoing edge, while a negative value indicates an incoming edge.
- WHERE <expression> : specifies the traversal filters. You can use the WHERE clause for the source vertices, the edges, and the destination vertices. You can use it together with AND, OR, NOT, and XOR. For more information, see WHERE.

#### Q Note

- There are some restrictions for the WHERE clause when you traverse along with multiple edge types. For example, WHERE edge1.prop1 > edge2.prop2 is not supported.
- The GO statement is executed by traversing all the vertices and then filtering according to the filter condition.
- YIELD [DISTINCT] <return\_list>: defines the output to be returned. It is recommended to use the Schema-related functions to fill in <return\_list>. src(edge), dst(edge), type(edge)), rank(edge), etc., are currently supported, while nested functions are not. For more information, see YIELD.
- SAMPLE <sample\_list> : takes samples from the result set. For more information, see SAMPLE.
- imit\_by\_list\_clause> : limits the number of outputs during the traversal process. For more information, see LIMIT.
- GROUP BY : groups the output into subgroups based on the value of the specified property. For more information, see GROUP BY. After grouping, you need to use YIELD again to define the output that needs to be returned.
- ORDER BY : sorts outputs with specified orders. For more information, see ORDER BY.

#### Q Note

When the sorting method is not specified, the output orders can be different for the same query.

• LIMIT [<offset>,] <number\_rows>] : limits the number of rows of the output. For more information, see LIMIT.

#### Notes

- The WHERE and YIELD clauses in 60 statements usually utilize property reference symbols ( \$^ and \$\$ ) or the properties(\$^) and properties(\$\$) functions to specify the properties of a vertex; use the properties(edge) function to specify the properties of an edge. For details, see Property Reference Symbols and Schema-related Functions.
- When referring to the result of a subquery in a compound 60 statement, you need to set a name for the result and pass it to the next subquery using the pipe symbol ||, and reference the name of the result in the next subquery using \$-. See the Pipe Operator for details.
- When the queried property has no value, the returned result displays NULL.

#### Cases and examples

TO QUERY THE IMMEDIATE NEIGHBORS OF A VERTEX

For example, to query the team that a person belongs to, assuming that the person is connected to the team by the serve edge and the person's ID is player102.

nebula>	GO	FROM	"player102"	OVER	serve	YIELD	dst(edge);
+		+					
dst(E	DGE	)					
+		+					
"team	203'	"					
"team	204'	"					
+		+					

TO QUERY ALL VERTICES WITHIN A SPECIFIED NUMBER OF HOPS FROM A STARTING VERTEX

For example, to query all vertices within two hops of a person vertex, assuming that the person is connected to other people by the follow edge and the person's ID is player102.

```
# Return all vertices that are 2 hops away from the player102 vertex.
nebula> G0 2 STEPS FROM "player102" OVER follow YIELD dst(edge);
dst(EDGE)
  "player101"
  "player125"
  "plaver100
  "player102"
  "player125"
# Return all vertices within 1 or 2 hops away from the player102 vertex.
nebula> GO 1 TO 2 STEPS FROM "player100" OVER follow \
        YIELD dst(edge) AS destination;
destination
  "player101"
| "player125"
# The following MATCH query has the same semantics as the previous GO query.
nebula> MATCH (v) -[e:follow*1..2]->(v2) \
WHERE id(v) == "player100" \
RETURN id(v2) AS destination;
destination
  "player100"
 "player102"
```

TO ADD FILTERING CONDITIONS

Case: To query the vertices and edges that meet specific conditions.

For example, use the WHERE clause to query the edges with specific properties between the starting vertex and the destination vertex.

```
nebula> G0 FROM "player100", "player102" OVER serve \
WHERE properties(edge).start_year > 1995 \
YIELD DISTINCT properties($$).name AS team_name, properties(edge).start_year AS start_year, properties($^).name AS player_name;
+-----+
```

team_name	start_year   +	player_name
"Snurs"	1997	"Tim Duncan"
"Trail Blazers"	2006	"LaMarcus Aldridge"
"Snurs"	2015	"LaMarcus Aldridge"
+	++	+

TO QUERY ALL EDGES

Case: To query all edges that are connected to the starting vertex.

TO QUERY MULTIPLE EDGE TYPES

Case: To query multiple edge types that are connected to the starting vertex. You can specify multiple edge types or the \* symbol to query multiple edge types.

For example, to query the follow and serve edges that are connected to the starting vertex.

<pre>nebula&gt; 60 FROM "player100" OVER follow, serve \     YIELD properties(edge).degree, properties(edge).start_year;</pre>
++
properties(EDGE).degree   properties(EDGE).start_year
95  NULL
95NULL
NULL   1997
++

TO QUERY INCOMING VERTICES USING THE REVERSELY KEYWORD



TO USE SUBQUERIES AS THE STARTING VERTICE OF A GRAPH TRAVERSAL

Friend	Of	Team	
+			+
"Boris   "Boris   "Boris	Diaw" Diaw" Diaw"	"Spurs" "Jazz" "Suns"	   

TO USE GROUP BY TO GROUP THE OUTPUT

You need to use  $\ensuremath{\texttt{YIELD}}$  to define the output that needs to be returned after grouping.

<pre># The following example collect nebula&gt; GO 2 STEPS FROM "player</pre>	ts the outputs according to age. r100" OVER follow ∖
YIELD src(edge) AS src,   GROUP BY \$dst \	, dst(edge) AS dst, properties(\$\$).age AS age \
YIELD \$dst AS dst, co	ollect_set(\$src) AS src, collect(\$age) AS age
+	+
dst   src	age
TT	T
"player125"   {"player101"}	[[41]]
"player100"   {"player125", "	"player101"}   [42, 42]
"player102"   {"player101"}	[33]
+	+

TO USE ORDER BY AND LIMIT TO SORT AND LIMIT THE OUTPUT

# The following example groups the outputs and restricts the number of rows of the outputs.
nebula> \$a = GO FROM "player100" OVER follow YIELD src(edge) AS src, dst(edge) AS dst; \
GO 2 STEPS FROM \$a.dst OVER follow \
YIELD \$a.src AS src, \$a.dst, src(edge), dst(edge) \
ORDER BY \$src   OFFSET 1 LIMIT 2;
++
src   \$a.dst   src(EDGE)   dst(EDGE)
++
"player100"   "player101"   "player100"   "player101"
"player100"   "player125"   "player100"   "player125"
++

OTHER EXAMPLES

# The following example determines if \$\$.player.name IS NOT EMPTY. nebula> G0 FROM "player100" OVER follow WHERE properties(\$\$).name IS NOT EMPTY YIELD dst(edge);

| dst(EDGE)

| "player125" | "player101"

Last update: November 29, 2023

# 4.5.6 FETCH

The FETCH statement retrieves the properties of the specified vertices or edges.

# **OpenCypher Compatibility**

This topic applies to native nGQL only.

#### Fetch vertex properties

SYNTAX

FETCH PROP ON {<tag\_name>[, tag\_name ...] | \*}
<vid>[, vid ...]
YIELD [DISTINCT] <return\_list> [AS <alias>];

Parameter	Description
tag_name	The name of the tag.
*	Represents all the tags in the current graph space.
vid	The vertex ID.
YIELD	Define the output to be returned. For details, see <b>WIELD</b> .
AS	Set an alias.

FETCH VERTEX PROPERTIES BY ONE TAG

# Specify a tag in the FETCH statement to fetch the vertex properties by that tag.

```
nebula> FETCH PROP ON player "player100" YIELD properties(vertex);
+-----+
| properties(VERTEX) |
+----+
| {age: 42, name: "Tim Duncan"} |
+-----+
```

FETCH SPECIFIC PROPERTIES OF A VERTEX

Use a YIELD clause to specify the properties to be returned.

FETCH PROPERTIES OF MULTIPLE VERTICES

Specify multiple VIDs (vertex IDs) to fetch properties of multiple vertices. Separate the VIDs with commas.

nebula> FETCH PROP ON player "player101", "player102", "player103" YIELD properties(vertex); +------+ | properties(VERTEX) | +-----+ | {age: 33, name: "LaMarcus Aldridge"} | | {age: 36, name: "Tony Parker"} | | {age: 32, name: "Rudy Gay"} | +------+

FETCH VERTEX PROPERTIES BY MULTIPLE TAGS

Specify multiple tags in the FETCH statement to fetch the vertex properties by the tags. Separate the tags with commas.

# The following example creates a new tag t1.
nebula> CREATE TAG IF NOT EXISTS t1(a string, b int);

# The following example attaches t1 to the vertex "player100". nebula> INSERT VERTEX t1(a, b) VALUES "player100":("Hello", 100);

# The following example fetches the properties of vertex "player100" by the tags player and t1.

nebuta-	FEICH	PRUP	un playe	, LI	playe	1100	TIELD	vertex P	15 V;		
+											+
V											
+											+
("pla	yer100"	:pla	yer{age:	42, n	ame: "	'Tim Du	ncan"}	:t1{a:	"Hell	o", b:	100})
+											+

You can combine multiple tags with multiple VIDs in a FETCH statement.

nebula>	FETCH	PROP	ON pl	.ayer,	t1	"play	/er100	", "pla	yer103"	YIELD	vertex	AS	v;
+													+
v													
+													+
("play	yer100"	:pla	yer{a	ge: 4	2,	name:	"Tim	Duncan"	} :t1{a	: "Hel	lo", b:	100	)})
("play	yer103"	:pla	yer{a	ige: 3	2,	name:	"Rudy	Gay"})					
+													+

FETCH VERTEX PROPERTIES BY ALL TAGS

Set an asterisk symbol \* to fetch properties by all tags in the current graph space.

nebula> FETCH PROP ON * "player100", "player106", "team200" YIELD vertex AS v;
++
v
++
("player100" :player{age: 42, name: "Tim Duncan"} :t1{a: "Hello", b: 100})
("player106" :player{age: 25, name: "Kyle Anderson"})
("team200" :team{name: "Warriors"})
++

#### Fetch edge properties

#### SYNTAX

FETCH PROP ON <edge\_type> <src\_vid> -> <dst\_vid>[@<rank>] [, <src\_vid> -> <dst\_vid> ...]
YIELD <output>;

Parameter	Description
edge_type	The name of the edge type.
src_vid	The VID of the source vertex. It specifies the start of an edge.
dst_vid	The VID of the destination vertex. It specifies the end of an edge.
rank	The rank of the edge. It is optional and defaults to 0. It distinguishes an edge from other edges with the same edge type, source vertex, destination vertex, and rank.
YIELD	Define the output to be returned. For details, see <b>VIELD</b> .

FETCH ALL PROPERTIES OF AN EDGE

The following statement fetches all the properties of the serve edge that connects vertex "player100" and vertex "team204".

nebula> FETCH PROP ON serve "player100" -> "team204" YIELD properties(edge);
+-----+
| properties(EDGE) |
+----+
| {end\_year: 2016, start\_year: 1997} |

FETCH SPECIFIC PROPERTIES OF AN EDGE

Use a YIELD clause to fetch specific properties of an edge.

nebula>	FETCH YIELD	PROP prope	ON s ertie	serve es(edg	"playe ge).sta	r100" rt_yea	-> " r;	'team	204"	
+   prope	rties(I	EDGE).	star	rt_yea	ar					
1997					Ì					

FETCH PROPERTIES OF MULTIPLE EDGES

Specify multiple edge patterns (  $<src_vid> -> <dst_vid>[@<rank>]$  ) to fetch properties of multiple edges. Separate the edge patterns with commas.

<pre>nebula&gt; FETCH PROP ON serve "player100" -&gt; "team204", "player133"</pre>	" -> "team202"	YIELD edge AS e;
+	+	
e	1	
+	+	
<pre>[:serve "player100"-&gt;"team204" @0 {end_year: 2016, start_year:</pre>	1997}]	
<pre>  [:serve "player133"-&gt;"team202" @0 {end_year: 2011, start_year:</pre>	2002}]	
1	+	

#### Fetch properties based on edge rank

If there are multiple edges with the same edge type, source vertex, and destination vertex, you can specify the rank to fetch the properties on the correct edge.

# Use FETCH in composite queries

A common way to use  $\ensuremath{\mathsf{FETCH}}$  is to combine it with native nGQL such as  $\ensuremath{\mathsf{GO}}$  .

The following statement returns the degree values of the follow edges that start from vertex "player101".

Or you can use user-defined variables to construct similar queries.

```
nebula> $var = G0 FROM "player101" OVER follow \
    YIELD src(edge) AS s, dst(edge) AS d; \
    FETCH PROP ON follow $var.s -> $var.d \
    YIELD properties(edge).degree;
+-----+
| properties(EDGE).degree |
+----+
| 95 |
90 |
95 |
95 |
```

For more information about composite queries, see Composite queries (clause structure).

```
Last update: October 25, 2023
```

# 4.5.7 SHOW

# SHOW CHARSET

The  $\ensuremath{\mathsf{SHOW}}$  CHARSET statement shows the available character sets.

 $Currently\ available\ types\ are\ utf8\ and\ utf8mb4\ .\ The\ default\ charset\ type\ is\ utf8\ .\ NebulaGraph\ extends\ the\ uft8\ to\ support\ fourboxing the\ support\ fourboxing the\ support\ f$ 

# SYNTAX

SHOW CHARSET;

## EXAMPLE

nebula> SHOW CHARSET;

+	++	+
Charset   Description	Default collation   !	Maxlen
"utf8"   "UTF-8 Unicode"	"utf8_bin"   4	4
+	++	+

Parame	eter	Description
Charset		The name of the character set.
Descripti	on	The description of the character set.
Default c	collation	The default collation of the character set.
Maxlen		The maximum number of bytes required to store one character.

# SHOW COLLATION

The  $\ensuremath{\mathsf{SHOW}}$  COLLATION statement shows the collations supported by NebulaGraph.

Currently available types are:  $\tt utf8\_bin$  and  $\tt utf8mb4\_bin$  .

- $\bullet$  When the character set is <code>utf8</code>, the default collate is <code>utf8\_bin</code>.
- When the character set is utf8mb4, the default collate is utf8mb4\_bin.

# SYNTAX

SHOW COLLATION;

# EXAMPLE

nebula> SHOW COLLATION; ++   Collation   Charset   ++   "utf8_bin"   "utf8"   ++	
Parameter	Description
Collation	The name of the collation.
Charset	The name of the character set with which the collation is associated.

# SHOW CREATE SPACE

The  $\ensuremath{\mathsf{SHOW}}$  CREATE <code>SPACE</code> statement shows the creating statement of the specified graph space.

For details about the graph space information, see CREATE SPACE.

#### SYNTAX

SHOW CREATE SPACE <space\_name>;

#### EXAMPLE

nebula> SHOW CREATE S	SPACE basketballplayer;
Space	Create Space
"basketballplayer" +	<pre>" "CREATE SPACE `basketballplayer` (partition_num = 10, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type = FIXED_STRING(32))"   +</pre>

# SHOW CREATE TAG/EDGE

The SHOW CREATE TAG statement shows the basic information of the specified tag. For details about the tag, see CREATE TAG.

The SHOW CREATE EDGE statement shows the basic information of the specified edge type. For details about the edge type, see CREATE EDGE.

#### SYNTAX

SHOW CREATE {TAG <tag\_name> | EDGE <edge\_name>};

## EXAMPLES

nebula> SHOW	CREATE TAG player;
Tag   ++	Create Tag
"player"               ++	<pre>"CREATE TAG `player` (</pre>
nebula> SHOW ++	CREATE EDGE follow;
Edge   ++	Create Edge
"follow"         	<pre>"CREATE EDGE `follow` (     `degree` int64 NULL   ) ttl_duration = 0, ttl_col = """   </pre>

## SHOW HOSTS

The SHOW HOSTS statement shows the cluster information, including the port, status, leader, partition, and version information. You can also add the service type in the statement to view the information of the specific service.

SYNTAX

SHOW HOSTS [GRAPH | STORAGE | META];

#### Q Note

For a NebulaGraph cluster installed with the source code, the version of the cluster will not be displayed in the output after executing the command SHOW HOSTS (GRAPH | STORAGE | META) with the service name.

EXAMPLES

nebula> SHOW HOS	STS;				+	++
Host	Port	Status	Leader count	Leader distribution	Partition distribution	Version
"storaged0"     "storaged1"     "storaged2"	9779   9779   9779	"ONLINE"   "ONLINE"   "ONLINE"	8 9 8	"docs:5, basketballplayer:3" "basketballplayer:4, docs:5" "basketballplayer:3, docs:5"	"docs:5, basketballplayer:3"   "docs:5, basketballplayer:4"   "docs:5, basketballplayer:3"	"3.6.0"     "3.6.0"     "3.6.0"

nebula> SHOW HOSTS GRAPH;

++	+		++	+
Host	Port   Status	Role	Git Info Sha	Version
<pre>""""""""""""""""""""""""""""""""""""</pre>	9669   "ONLINE"   9669   "ONLINE"   9669   "ONLINE"   9669   "ONLINE"	"GRAPH" "GRAPH" "GRAPH"	"3ba41bd"   "3ba41bd"   "3ba41bd"	"3.6.0"   "3.6.0"   "3.6.0"

nebula> SHOW HOSTS STORAGE;

+		+.	+-				+		+-		+
I	Host		Port	Status		Role	L	Git Info S	ha	Version	I
+		+-	+-	+	+-		+-		+-		+
L	"storaged0"		9779	"ONLINE"		"STORAGE"	L	"3ba41bd"	1	"3.6.0"	I
	"storaged1"		9779	"ONLINE"		"STORAGE"	L	"3ba41bd"		"3.6.0"	1
Í	"storaged2"	Í	9779	"ONLINE"		"STORAGE"	Í	"3ba41bd"	i	"3.6.0"	Î

#### nebula> SHOW HOSTS META;

+	+	+	+	+		++
l	Host	Port	Status	Role	Git Info Sha	Version
+	+	+	+	+		++
L	"metad2"	9559	"ONLINE"	"META"	"3ba41bd"	3.6.0"
	"metad0"	9559	"ONLINE"	"META"	"3ba41bd"	3.6.0"
l	"metad1"	9559	"ONLINE"	"META"	"3ba41bd"	3.6.0"
+	+	+	+	+		++

# SHOW INDEX STATUS

The SHOW INDEX STATUS statement shows the status of jobs that rebuild native indexes, which helps check whether a native index is successfully rebuilt or not.

SYNTAX

SHOW {TAG | EDGE} INDEX STATUS;

#### EXAMPLES

nebula> SHOW TAG INDEX STATUS;	++
Name	Index Status
"date1_index"   "basketballplayer_all_tag_indexes"   "any_shape_geo_index" +	"FINISHED"     "FINISHED"     "FINISHED"   ++

# nebula> SHOW EDGE INDEX STATUS;

+	++
Name	Index Status
+	++
"follow_index"	"FINISHED"
+	++

# RELATED TOPICS

- Job manager and the JOB statements
- REBUILD NATIVE INDEX

# SHOW INDEXES

The  $\ensuremath{\mathsf{SHOW}}$  INDEXES statement shows the names of existing native indexes.

#### SYNTAX

SHOW {TAG | EDGE} INDEXES;

# EXAMPLES

nebula> SHOW TAG I	NDEXES;	.++
Index Name	By Tag	Columns
"player_index_0"   "player_index_1" +	"player"   "player" -+	[]     ["name"]   ++
nebula> SHOW EDGE ++	INDEXES;	+
Index Name	By Edge	Columns
"follow_index"   ++	"follow"	[] [

#### L Jacy version compatibility

In NebulaGraph 2.x, SHOW TAG/EDGE INDEXES only returns  ${\tt Names}\,.$ 

Last update: November 3, 2023

# SHOW PARTS

The SHOW PARTS statement shows the information of a specified partition or all partitions in a graph space.

### SYNTAX

SHOW PARTS [<part\_id>];

# EXAMPLES

nebula> SHOW PARTS;						
+1	+	D	++			
Partition ID	Leader	Peers	LOSTS			
+	++		++			
1	"192.168.2.1:9779"	"192.168.2.1:9779"	""			
2	"192.168.2.2:9779"	"192.168.2.2:9779"				
3	"192.168.2.3:9779"	"192.168.2.3:9779"				
4	"192.168.2.1:9779"	"192.168.2.1:9779"				
5	"192.168.2.2:9779"	"192.168.2.2:9779"				
6	"192.168.2.3:9779"	"192.168.2.3:9779"				
7	"192.168.2.1:9779"	"192.168.2.1:9779"	""			
8	"192.168.2.2:9779"	"192.168.2.2:9779"				
9	"192.168.2.3:9779"	"192.168.2.3:9779"				
10	"192.168.2.1:9779"	"192.168.2.1:9779"	""			
+	+		++			

# nebula> SHOW PARTS 1;

+	+		++
Partition ID	Leader	Peers	Losts
+	++		++
1	"192.168.2.1:9779"	"192.168.2.1:9779"	""
+	++		++

The descriptions are as follows.

Parameter	Description
Partition ID	The ID of the partition.
Leader	The IP (or hostname) and the port of the leader.
Peers	The IPs (or hostnames) and the ports of all the replicas.
Losts	The IPs (or hostnames) and the ports of replicas at fault.

Last update: November 22, 2023

## SHOW ROLES

The  $\ensuremath{\mathsf{SHOW}}$  ROLES statement shows the roles that are assigned to a user account.

The return message differs according to the role of the user who is running this statement:

- If the user is a GOD or ADMIN and is granted access to the specified graph space, NebulaGraph shows all roles in this graph space except for GOD.
- If the user is a DBA, USER, or GUEST and is granted access to the specified graph space, NebulaGraph shows the user's own role in this graph space.
- If the user does not have access to the specified graph space, NebulaGraph returns PermissionError.

For more information about roles, see Roles and privileges.

SYNTAX

SHOW ROLES IN <space\_name>;

EXAMPLE

```
nebula> SHOW ROLES in basketballplayer;
+-----+
| Account | Role Type |
+-----+
| "user1" | "ADMIN"
+-----+
```

#### SHOW SNAPSHOTS

The  $\ensuremath{\mathsf{SHOW}}$  SNAPSHOTS statement shows the information of all the snapshots.

For how to create a snapshot and backup data, see Snapshot.

ROLE REQUIREMENT

Only the root user who has the  ${\tt GOD}$  role can use the  ${\tt SHOW}$   ${\tt SNAPSHOTS}$  statement.

# SYNTAX

SHOW SNAPSHOTS;

#### EXAMPLE

nebula> SHOW SNAPSHOTS;	+	
Name	Status	Hosts
"SNAPSHOT_2020_12_16_11_13_55"	"VALID"	"storaged0:9779, storaged1:9779, storaged2:9779"
+	+	++

# SHOW SPACES

The  $\ensuremath{\mathsf{SHOW}}$  Spaces is statement shows existing graph spaces in NebulaGraph.

For how to create a graph space, see CREATE SPACE.

#### SYNTAX

SHOW SPACES;

#### EXAMPLE

## SHOW STATS

The SHOW STATS statement shows the statistics of the graph space collected by the latest SUBMIT JOB STATS job.

The statistics include the following information:

- The number of vertices in the graph space
- The number of edges in the graph space
- The number of vertices of each tag
- The number of edges of each edge type

# Arning

The data returned by SHOW STATS is not real-time. The returned data is collected by the latest SUBMIT JOB STATS job and may include TTL-expired data. The expired data will be deleted and not included in the statistics the next time the Compaction operation is performed.

#### PREREQUISITES

You have to run the SUBMIT JOB STATS statement in the graph space where you want to collect statistics. For more information, see SUBMIT JOB STATS.

Caution

The result of the SHOW STATS statement is based on the last executed SUBMIT JOB STATS statement. If you want to update the result, run SUBMIT JOB STATS again. Otherwise the statistics will be wrong.

#### SYNTAX

SHOW STATS;

#### EXAMPLES

# Choose a graph space. nebula> USE basketballplayer;

# Start SUBMIT JOB STATS. nebula> SUBMIT JOB STATS; +----+ | New Job Id | +----+ | 98 |

# Make sure the job executes successfully.

#### nebula> SHOW JOB 98;

Job Id(TaskId)   C	Command(Dest)	Status	Start Time	Stop Time	Error Code
98   "	STATS"	"FINISHED"	2021-11-01T09:33:21.000000	2021-11-01T09:33:21.000000	"SUCCEEDED"
0 "	storaged2"	"FINISHED"	2021-11-01T09:33:21.000000	2021-11-01T09:33:21.000000	"SUCCEEDED"
1   "	storaged0"	"FINISHED"	2021-11-01T09:33:21.000000	2021-11-01T09:33:21.000000	"SUCCEEDED"
2 "	storaged1"	"FINISHED"	2021-11-01T09:33:21.000000	2021-11-01T09:33:21.000000	"SUCCEEDED"
"Total:3"   "	Succeeded:3"	"Failed:0"	"In Progress:0"		

# Show the statistics of the graph space.

#### nebula> SHOW STATS;

"Tag"   "player"   51     "Tag"   "team"   30     "Edge"   "follow"   81     "Edge"   "serve"   152     "Space"   "vertices"   81	+   Type +	Name	+   Count   +
"Space"   "edges"   233	"Tag"	"player"	51
	"Tag"	"team"	30
	"Edge"	"follow"	81
	"Edge"	"serve"	152
	"Space"	"vertices"	81
	"Space"	"edges"	233

# SHOW TAGS/EDGES

The  $\ensuremath{\mathsf{SHOW}}$  TAGS statement shows all the tags in the current graph space.

The  $\ensuremath{\operatorname{SHOW}}$  EDGES statement shows all the edge types in the current graph space.

#### SYNTAX

SHOW {TAGS | EDGES};

### EXAMPLES

nebula> SHOW	TAGS;
++	,
Name	
++	
"player"	
star"	
"team"	
++	
nebula> SHOW ++	EDGES;
Name	
++	
"follow"	
serve"	
++	

# SHOW USERS

The SHOW USERS statement shows the user information.

ROLE REQUIREMENT

Only the root user who has the  $\ensuremath{\operatorname{GOD}}$  role can use the  $\ensuremath{\operatorname{SHOW}}$  USERS statement.

SYNTAX

SHOW USERS;

EXAMPLE

nebula> SHOW USERS; +-----+ | Account | IP Whitelist | +-----+ | "root" | "" | | "user1" | "" | | "user2" | "192.168.10.10" | +----+
#### SHOW SESSIONS

When a user logs in to the database, a corresponding session will be created and users can query for session information.

The SHOW SESSIONS statement shows the information of all the sessions. It can also show a specified session with its ID.

PRECAUTIONS

- The client will call the API release to release the session and clear the session information when you run exit after the operation ends. If you exit the database in an unexpected way and the session timeout duration is not set via session\_idle\_timeout\_secs in nebula-graphd.conf, the session will not be released automatically. For those sessions that are not automatically released, you need to delete them manually. For details, see KILL SESSIONS.
- SHOW SESSIONS queries the session information of all the Graph services.
- SHOW LOCAL SESSIONS queries the session information of the currently connected Graph service and does not query the session information of other Graph services.
- SHOW SESSION <Session\_Id> queries the session information with a specific session id.

SYNTAX

SHOW [LOCAL] SESSIONS; SHOW SESSION <Session\_Id>;

#### EXAMPLES

nebula> SHOW SESSIONS;							
SessionId	UserName	SpaceName	CreateTime	UpdateTime	GraphAddr	Timezone	ClientIp
	"root"   "root"   "root"   "root"   "root"   "root"	<pre>"basketballplayer" "basketballplayer" "basketballplayer" "basketballplayer" "basketballplayer" "basketballplayer"</pre>	2022-04-29T08:27:38.102296 2022-04-29T02:28:50.300991 2022-04-28T02:28:19.847744 2022-04-27T06:31:32.662100 2022-04-26T07:50:29.593975	2022-04-29T08:50:46.282921 2022-04-29T08:16:28.339038 2022-04-28T08:17:44.470210 2022-04-27T07:01:25.200978 2022-04-26T07:51:47.184810	"127.0.0.1:9669" "127.0.0.1:9669" "127.0.0.1:9669" "127.0.0.1:9669" "127.0.0.1:9669" "127.0.0.1:9669"	0   0   0   0   0	
1650958897679595       "root"       ""       2022-04-26T07:41:37.679595       2022-04-26T07:41:37.683802       "127.0.0.1:9669"       0       "127.0.0.1"         +							
+   SessionId	+   UserName	+   SpaceName	+   CreateTime	⊦ UpdateTime	⊦   GraphAddr	+   Timezone	++   ClientIp

SessionId	UserName	SpaceName	CreateTime	UpdateTime	GraphAddr	Timezone	ClientIp
1651220858102296	"root"	"basketballplayer"	2022-04-29T08:27:38.102296	2022-04-29T08:50:54.254384	"127.0.0.1:9669"	0	"127.0.0.1"

Parameter	Description
SessionId	The session ID, namely the identifier of a session.
UserName	The username in a session.
SpaceName	The name of the graph space that the user uses currently. It is null ( "") when you first log in because there is no specified graph space.
CreateTime	The time when the session is created, namely the time when the user logs in. The time zone is specified by timezone_name in the configuration file.
UpdateTime	The system will update the time when there is an operation. The time zone is specified by timezone_name in the configuration file.
GraphAddr	The IP (or hostname) and port of the Graph server that hosts the session.
Timezone	A reserved parameter that has no specified meaning for now.
ClientIp	The IP or hostname of the client.

Last update: November 22, 2023

### SHOW QUERIES

The SHOW QUERIES statement shows the information of working queries in the current session.

Note	
To terminate queries, see Kill Ouery.	

PRECAUTIONS

- The SHOW LOCAL QUERIES statement gets the status of queries in the current session from the local cache with almost no latency.
- The SHOW QUERIES statement gets the information of queries in all the sessions from the Meta Service. The information will be synchronized to the Meta Service according to the interval defined by session\_reclaim\_interval\_secs. Therefore the information that you get from the client may belong to the last synchronization interval.

SYNTAX

CHUM	FLOCAL 1	OUEDTEC -
SHOW	LUCAL	QUERTES:

#### EXAMPLES

nebula> SHOW LOCAL QUERIES;				
SessionID   ExecutionPlanID	User   Host	StartTime	+   DurationInUSec	Status   Query
++-   1625463842921750   46	-++	+ 68.x.x":9669"   2021-07-05T05:44:1	9.502903   0	"RUNNING"   "SHOW LOCAL QUERIES;"
+++	-+++	+	+-	+

nebula> SHOW QUERIES;

+	+	+	+	+	+	+	++
SessionID	ExecutionPlanID	User	Host	StartTime	DurationInUSec	Status	Query
1625456037718757		"user1"	""192.168.x.x":9669"	2021-07-05T05:51:08.691318	1504502	"RUNNING"	<pre>"MATCH p=(v:player)-[*14]-(v2) RETURN v2 AS Friends;"</pre>

#### # The following statement returns the top 10 queries that have the longest duration. nebula> SHOW QUERIES | ORDER BY \$-.DurationInUSec DESC | LIMIT 10;

	-							
+	+	+	++	+	+	+	+	+
SessionID	ExecutionPlanID	User	Host	StartTime	DurationInUSec	Status	Query	L
+	+	+4	+	+	+	+	+	+
1625471375320831	98	"user2"	""192.168.x.x":9669"	2021-07-05T07:50:24.461779	2608176	"RUNNING"	"MATCH (v:player)-[*14]-(v2) RETURN v2 AS Friends;"	L
1625456037718757	99	"user1"	""192.168.x.x":9669"	2021-07-05T07:50:24.910616	2159333	"RUNNING"	"MATCH (v:player)-[*14]-(v2) RETURN v2 AS Friends;"	

The descriptions are as follows.

Parameter		Description
	SessionID	The session ID.
	ExecutionPlanID	The ID of the execution plan.
	User	The username that executes the query.
	Host	The IP address and port of the Graph server that hosts the session.
	StartTime	The time when the query starts.
	DurationInUSec	The duration of the query. The unit is microsecond.
	Status	The current status of the query.
	Query	The query statement.

Last update: October 25, 2023

### SHOW META LEADER

The SHOW META LEADER statement shows the information of the leader in the current Meta cluster.

For more information about the Meta service, see Meta service.

#### SYNTAX

STREAM	
SHOW META LEADER;	
EXAMPLE	
nebula> SHOW META LEADER; +	+ art beat   +
<b>Parameter</b> Meta Leader	<b>Description</b> Shows the information of the leader in the Meta cluster, including the IP (or hostname) and port of the server where the leader is located.
secs from last heart beat	Indicates the time interval since the last heartbeat. This parameter is measured in seconds.

Last update: November 22, 2023

### 4.5.8 FIND PATH

The FIND PATH statement finds the paths between the selected source vertices and destination vertices.

### Note

To improve the query performance with the FIND PATH statement, you can add the num\_operator\_threads parameter in the nebula-graphd.conf configuration file. The value range of the num\_operator\_threads parameter is [2, 10] and make sure that the value is not greater than the number of CPU cores of the machine where the graphd service is deployed. It is recommended to set the value to the number of CPU cores of the machine where the graphd service is deployed. For more information about the nebula-graphd.conf configuration file, see nebula-graphd.conf.

### Syntax

```
FIND { SHORTEST | ALL | NOLOOP } PATH [WITH PROP] FROM <vertex_id_list> TO <vertex_id_list>
OVER <edge_type_list> [REVERSELY | BIDIRECT]
[<wHERE clause>] [UPTO <\> {STEP|STEPS}]
YTELD path as <alias>
[| ORDER BY $-.path] [| LIMIT <\>];
<vertex_id_list> ::=
    [vertex_id [, vertex_id] ...]
```

- SHORTEST finds all the shortest path.
- ALL finds all the paths.
- NOLOOP finds the paths without circles.
- WITH PROP shows properties of vertices and edges. If not specified, properties will be hidden.
- <vertex\_id\_list> is a list of vertex IDs separated with commas (,). It supports \$- and \$var.
- <edge\_type\_list> is a list of edge types separated with commas (,). \* is all edge types.
- REVERSELY | BIDIRECT specifies the direction. REVERSELY is reverse graph traversal while BIDIRECT is bidirectional graph traversal.
- <WHERE clause> filters properties of edges.
- UPTO <N> {STEP|STEPS} is the maximum hop number of the path. The default value is 5.
- ORDER BY \$-.path specifies the order of the returned paths. For information about the order rules, see Path.
- LIMIT <M> specifies the maximum number of rows to return.

#### O Note

The path type of FIND PATH is trail. Only vertices can be repeatedly visited in graph traversal. For more information, see Path.

### Limitations

- When a list of source and/or destination vertex IDs are specified, the paths between any source vertices and the destination vertices will be returned.
- $\ensuremath{\bullet}$  There can be cycles when searching all paths.
- FIND PATH only supports filtering properties of edges with WHERE clauses. Filtering properties of vertices and functions are not supported for now.
- FIND PATH is a single-thread procedure, so it uses much memory.

#### Examples

A returned path is like (<vertex\_id>)-[:<edge\_type\_name>@<rank>]->(<vertex\_id) .

nebula> FIND SHORTEST PATH FROM "player102" TO "team204" OVER \* YIELD path AS p;

+-----+

nebula> FIND SHORTEST PATH WITH PROP FROM "team204" TO "player100" OVER \* REVERSELY YIELD path AS p;

nebula> FIND SHORTEST PATH FROM "player100", "player130" TO "player132", "player133" OVER \* BIDIRECT UPTO 18 STEPS YIELD path as p;

*   p   *
<pre>+ + (&lt;("player100")&lt;-[:follow@0 {}]-("player144")&lt;-[:follow@0 {}]- ("player133")&gt;</pre>
<pre>("player132")&gt;   &lt;("player132")&gt;   &lt;("player130")-[:serve@0 {}]-&gt;("team219")&lt;-[:serve@0 {}]-&gt;("team204")&lt;-[:serve@0 {}]-("player138")-[:serve@0 {}]-&gt;("team225")&lt;-[:serve@0 {}]- ("player132")&gt;       +</pre>

nebula> FIND ALL PATH FROM "player100" TO "team204" OVER \* WHERE follow.degree is EMPTY or follow.degree ≻=0 YIELD path AS p; +------+ | p |

+	+
<pre>("player100")-[:serve@0 {}]-&gt;("team204")&gt;</pre>	
<pre>("player100")-[:follow@0 {}]-&gt;("player125")-[:serve@0 {}]-&gt;("team204")&gt;</pre>	
<pre>  &lt;("player100")-[:follow@0 {}]-&gt;("player101")-[:serve@0 {}]-&gt;("team204")&gt;</pre>	
+	+

### FAQ

DOES IT SUPPORT THE WHERE CLAUSE TO ACHIEVE CONDITIONAL FILTERING DURING GRAPH TRAVERSAL?

FIND PATH only supports filtering properties of edges with WHERE clauses, such as WHERE follow.degree is EMPTY or follow.degree >= 0.

Filtering properties of vertices is not supported for now.

```
Last update: May 9, 2024
```

### 4.5.9 GET SUBGRAPH

The GET SUBGRAPH statement returns a subgraph that is generated by traversing a graph starting from a specified vertex. GET SUBGRAPH statements allow you to specify the number of steps and the type or direction of edges during the traversal.

#### Syntax

```
GET SUBGRAPH [WITH PROP] [<step_count> {STEP|STEPS}] FROM {<vid>, <vid>...}
[{IN | OUT | BOTH} <edge_type>, <edge_type>...]
[WHERE <expression> [AND <expression> ...]]
YIELD {[VERTICES AS <vertex_alias>] [,EDGES AS <edge_alias>]};
```

- WITH PROP shows the properties. If not specified, the properties will be hidden.
- step\_count specifies the number of hops from the source vertices and returns the subgraph from 0 to step\_count hops. It must be a non-negative integer. Its default value is 1.
- vid specifies the vertex IDs.
- edge\_type specifies the edge type. You can use IN, OUT, and BOTH to specify the traversal direction of the edge type. The default is BOTH.
- system clause> specifies the filter conditions for the traversal, which can be used with the boolean operator AND.
- YIELD defines the output that needs to be returned. You can return only vertices or edges. A column alias must be set.

### Note

The path type of GET SUBGRAPH is trait. Only vertices can be repeatedly visited in graph traversal. For more information, see Path.

#### Limitations

While using the WHERE clause in a GET SUBGRAPH statement, note the following restrictions:

- Only support the AND operator.
- Only support filter destination vertex, the vertex format must be \$\$.tagName.propName.
- Support filter edge, the edge format must be edge\_type.propName.
- **Support** math functions, aggregate functions, string functions, datetime functions, type conversion functions and general functions in list functions.
- Not support aggregate functions, schema-related functions, conditional expression, predicate functions, geography function and user-defined functions.

### Examples

The following graph is used as the sample.



### Insert the test data:

nebula> CREATE SPACE IF NOT EXISTS subgraph(partition\_num=15, replica\_factor=1, vid\_type=fixed\_string(30)); nebula> USE subgraph; nebula> CREATE TAG IF NOT EXISTS player(name string, age int);

- nebula> CREATE TAG IF NOT EXISTS team(name string); nebula> CREATE TAG IF NOT EXISTS team(name string); nebula> CREATE EDGE IF NOT EXISTS follow(degree int);

- nebula> CREATE EDGE IF NOT EXISTS follow(degree int); nebula> CREATE EDGE IF NOT EXISTS serve(start\_year int, end\_year int); nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42); nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36); nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33); nebula> INSERT VERTEX team(name) VALUES "player102":("LaMarcus Aldridge", 33); nebula> INSERT VERTEX team(name) VALUES "player101":- "team204":("Spurs"); nebula> INSERT EDGE follow(degree) VALUES "player101" -> "player100":(95);

nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player100":(75); nebula> INSERT EDGE serve(start\_year, end\_year) VALUES "player101" -> "team204":(1999, 2018),"player102" -> "team203":(2006, 2015);

• This example goes one step from the vertex player101 over all edge types and gets the subgraph.

nebula> GET SUBGRAPH 1 STEPS FROM "player101" YIELD VERTICES AS nodes, +	EDGES AS relationships;
+   nodes relationships	
+	
[("player101" :player{})] >"player102" @0 {}]]	[[:serve "player101"->"team204" @0 {}], [:follow "player101"->"player100" @0 {}], [:follow "player101"-
<pre>  [("team204" :team{}), ("player100" :player{}), ("player102" :player{ {}]]</pre>	<pre>})]   [[:follow "player102"-&gt;"player100" @0</pre>
+	+

The returned subgraph is as follows.



• This example goes one step from the vertex player101 over incoming follow edges and gets the subgraph.

+-			+	+
	[("player101"	:player{})]	[]	- I
+-			+	+

There is no incoming follow edge to player101, so only the vertex player101 is returned.

• This example goes one step from the vertex player101 over outgoing serve edges, gets the subgraph, and shows the property of the edge.

nebula> GET SUBGRAPH WITH PROP 1 STEPS FROM "player101"	OUT serve YIELD VERTICES AS nodes, EDGES AS relationships;
nodes	relationships
<pre>[("player101" :player{age: 36, name: "Tony Parker"})] [("team204" :team{name: "Spurs"})] </pre>	<pre>[[:serve "player101"-&gt;"team204" @0 {end_year: 2018, start_year: 1999}]]   []</pre>

The returned subgraph is as follows.



• This example goes two steps from the vertex player101 over follow edges, filters by degree > 90 and age > 30, and shows the properties of edges.

<pre>nebula&gt; GET SUBGRAPH WITH PROP 2 STEPS FROM "player101" WHERE follow.degree &gt; 90 AND \$\$.player.age &gt; 30 \ YIELD VERTICES AS nodes, EDGES AS relationships;</pre>	
nodes	relationships
+	++
<pre>  [("player101" :player{age: 36, name: "Tony Parker"})] [("player100" :player{age: 42, name: "Tim Duncan"})] +</pre>	[[:follow "player101"->"player100" @0 {degree: 95}]]     [] 

### FAQ

WHY IS THE NUMBER OF HOPS IN THE RETURNED RESULT GREATER THAN STEP\_COUNT ?

To show the completeness of the subgraph, an additional hop is made on all vertices that meet the conditions. The following graph is used as the sample.



- The returned paths of GET SUBGRAPH 1 STEPS FROM "A"; are  $A \rightarrow B$ ,  $B \rightarrow A$ , and  $A \rightarrow C$ . To show the completeness of the subgraph, an additional hop is made on all vertices that meet the conditions, namely  $B \rightarrow C$ .
- The returned path of GET SUBGRAPH 1 STEPS FROM "A" IN follow; is B->A. To show the completeness of the subgraph, an additional hop is made on all vertices that meet the conditions, namely A->B.

If you only query paths or vertices that meet the conditions, we suggest you use MATCH or GO. The example is as follows.

nebula> MATCH p= (v:player) -- (v2) WHERE id(v)=="A" RETURN p; nebula> 60 1 STEPS FROM "A" OVER follow YIELD src(edge),dst(edge);

WHY IS THE NUMBER OF HOPS IN THE RETURNED RESULT LOWER THAN STEP\_COUNT ?

The query stops when there is not enough subgraph data and will not return the null value.

nebula> GET SUBGRAPH 100 STEPS FROM "player101" OUT f +	ollow YIELD VERTICES AS nodes, EDGES AS relationships; +
nodes	relationships

Last update: November 17, 2023

### 4.6 Clauses and options

### 4.6.1 GROUP BY

The GROUP BY clause can be used to aggregate data.

### **OpenCypher Compatibility**

This topic applies to native nGQL only.

You can also use the count() function to aggregate data.

nebula> MATCH (v:player)<-[:follow]-(:player) RETURN v.player.name AS Name, count(\*) as cnt ORDER BY cnt DESC;</pre>

| Name | cnt | + ----+ | "Tim Duncan" | 10 | | "LeBron James" | 6 | | "Tony Parker" | 5 | "Chris Paul" | 4 | | "Manu Ginobili" | 4 | + ----+

### Syntax

The GROUP BY clause groups the rows with the same value. Then operations such as counting, sorting, and calculation can be applied.

The GROUP BY clause works after the pipe symbol (|) and before a YIELD clause.

| GROUP BY <var> YIELD <var>, <aggregation\_function(var)>

The aggregation\_function() function supports avg(), sum(), max(), min(), count(), collect(), and std().

#### Examples

The following statement finds all the vertices connected directly to vertex "player100", groups the result set by player names, and counts how many times the name shows up in the result set.

```
nebula> G0 FROM "player100" OVER follow BIDIRECT \
        YIELD properties($$).name as Name \
          GROUP BY $-.Name
        YIELD $-.Name as Player, count(*) AS Name_Count;
| Player
                       | Name_Count
  "Shaquille O'Neal" | 1
  "Tiago Splitter"
                        1
  "Manu Ginobili"
                         2
 "Boris Diaw"
"LaMarcus Aldridge"
                         1
                         1
  "Tony Parker"
  "Marco Belinelli"
                         1
  "Deiounte Murrav'
                        1
  "Danny Green"
 "Aron Baynes"
                        1
```

The following statement finds all the vertices connected directly to vertex "player100", groups the result set by source vertices, and returns the sum of degree values.

```
nebula> G0 FROM "player100" OVER follow \
    YIELD src(edge) AS player, properties(edge).degree AS degree \
    | GROUP BY S-.player \
    YIELD sum($-.degree);
+------+
    | sum($-.degree) |
+------+
```

| 190 |

For more information about the sum() function, see Built-in math functions.

### Implicit GROUP BY

The usage of GROUP BY in the above nGQL statements that explicitly write GROUP BY and act as grouping fields is called explicit GROUP BY, while in openCypher, the GROUP BY is implicit, i.e., GROUP BY groups fields without explicitly writing GROUP BY. The explicit GROUP BY in nGQL is the same as the implicit GROUP BY in openCypher, and nGQL also supports the implicit GROUP BY. For the implicit usage of GROUP BY, see Stack Overflow.

For example, to look up the players over 34 years old with the same length of service, you can use the following statement:

nebula> LOOKUP ON player WHERE player.age > 34 YIELD id(vertex) AS v   \
G0 FROM \$v OVER serve YIELD serve.start_year AS start_year, serve.end_year AS end_year   \
YIELD \$start_year, \$end_year, count(*) AS count   \
ORDER BY \$count DESC   LIMIT 5;
++
\$start_year   \$end_year   count
++
2018 2019 3
2007 2012 2
1998   2004   2
2017 2018 2
2010   2011   2
++++++-

Last update: November 6, 2023

### 4.6.2 LIMIT AND SKIP

The LIMIT clause constrains the number of rows in the output. The usage of LIMIT in native nGQL statements and openCypher compatible statements is different.

- Native nGQL: Generally, a pipe | needs to be used before the LIMIT clause. The offset parameter can be set or omitted directly after the LIMIT statement.
- OpenCypher compatible statements: No pipes are permitted before the LIMIT clause. And you can use SKIP to indicate an offset.

Note

When using LIMIT in either syntax above, it is important to use an ORDER BY clause that constrains the output into a unique order. Otherwise, you will get an unpredictable subset of the output.

#### LIMIT in native nGQL statements

In native nGQL, LIMIT has general syntax and exclusive syntax in GO statements.

GENERAL LIMIT SYNTAX IN NATIVE NGQL STATEMENTS

In native nGQL, the general LIMIT syntax works the same as in SQL. The LIMIT clause accepts one or two parameters. The values of both parameters must be non-negative integers and be used after a pipe. The syntax and description are as follows:

... | LIMIT [<offset>,] <number\_rows>;

Param	eter	Description
offset		The offset value. It defines the row from which to start returning. The offset starts from 0. The default value is 0, which returns from the first row.
number_r	OWS	It constrains the total number of returned rows.

#### For example:

# The following example returns the top 3 rows of data from the result. nebula> LOOKUP ON player YIELD id(vertex) |\ LIMIT 3; | id(VERTEX) "player100" "player101" "player102" # The following example returns the 3 rows of data starting from the second row of the sorted output. nebula> GO FROM "player100" OVER follow REVERSELY  $\setminus$ YIELD properties(\$\$).name AS Friend, properties(\$\$).age AS Age \ ORDER BY \$-.Age, \$-.Friend  $\setminus$ | LIMIT 1. 3: | Friend Age "Danny Green" | 31 "Aron Bavnes' 32 "Marco Belinelli" | 32

LIMIT IN GO STATEMENTS

In addition to the general syntax in the native nGQL, the LIMIT in the 60 statement also supports limiting the number of output results based on edges.

Syntax:

#### <go\_statement> LIMIT <limit\_list>;

limit\_list is a list. Elements in the list must be natural numbers, and the number of elements must be the same as the maximum
number of STEPS in the GO statement. The following takes GO 1 TO 3 STEPS FROM "A" OVER \* LIMIT <limit\_list> as an example to introduce
this usage of LIMIT in detail.

- The list limit\_list must contain 3 natural numbers, such as GO 1 TO 3 STEPS FROM "A" OVER \* LIMIT [1,2,4].
- 1 in LIMIT [1,2,4] means that the system automatically selects 1 edge to continue traversal in the first step. 2 means to select 2 edges to continue traversal in the second step. 4 indicates that 4 edges are selected to continue traversal in the third step.
- Because 60 1 T0 3 STEPS means to return all the traversal results from the first to third steps, all the red edges and their source and destination vertices in the figure below will be matched by this 60 statement. And the yellow edges represent there is no path selected when the GO statement traverses. If it is not 60 1 T0 3 STEPS but 60 3 STEPS, it will only match the red edges of the third step and the vertices at both ends.





```
nebula> GO 3 STEPS FROM "player100" \
       OVER * \
        YIELD properties($$).name AS NAME, properties($$).age AS Age \
       LIMIT [3,3,3];
| NAME
                  | Age
 "Tony Parker"
                  | 36
  "Manu Ginobili"
                  | 41
  "Spurs"
                   __NULL_
nebula> GO 3 STEPS FROM "player102" OVER * BIDIRECT\
        YIELD dst(edge)
       LIMIT [rand32(5),rand32(5),rand32(5)];
| dst(EDGE)
```

| "player100" | | "player100" | +----+

#### LIMIT in openCypher compatible statements

In openCypher compatible statements such as MATCH, there is no need to use a pipe when LIMIT is used. The syntax and description are as follows:

•	[SKIP <ottset>] [LIMI1 <number_rows>];</number_rows></ottset>				
	Parameter	Description			
	offset	The offset value. It defines the row from which to start returning. The offset starts from $0$ . The default value is $0$ , which returns from the first row.			
	number_rows	It constrains the total number of returned rows.			

Both offset and number\_rows accept expressions, but the result of the expression must be a non-negative integer.

### Note

Fraction expressions composed of two integers are automatically floored to integers. For example, 8/6 is floored to 1.

EXAMPLES OF LIMIT

LIMIT can be used alone to return a specified number of results.

nebula> MATCH (v:player) R	ETURN v.player.name AS Name, v.player.age AS Age '
ORDER BY Age LIMIT	5;
+	++
Name	Age
+	++
"Luka Doncic"	20
"Ben Simmons"	22
"Kristaps Porzingis"	23
"Giannis Antetokounmpo"	24
"Kyle Anderson"	25
+	++

EXAMPLES OF SKIP

SKIP can be used alone to set the offset and return the data after the specified position.

EXAMPLE OF SKIP AND LIMIT

SKIP and LIMIT can be used together to return the specified amount of data starting from the specified position.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
RETURN v2.player.name AS Name, v2.player.age AS Age \
ORDER BY Age DESC SKIP 1 LIMIT 1;
+-----+
| Name | Age |
+-----+
```

| "Manu Ginobili" | 41 | +----+

Last update: October 25, 2023

### 4.6.3 SAMPLE

The SAMPLE clause takes samples evenly in the result set and returns the specified amount of data.

SAMPLE can be used in GO statements only. The syntax is as follows:

<go\_statement> SAMPLE <sample\_list>;

sample\_list is a list. Elements in the list must be natural numbers, and the number of elements must be the same as the maximum number of STEPS in the GO statement. The following takes GO 1 TO 3 STEPS FROM "A" OVER \* SAMPLE <sample\_list> as an example to introduce this usage of SAMPLE in detail.

- The list sample\_list must contain 3 natural numbers, such as 60 1 TO 3 STEPS FROM "A" OVER \* SAMPLE [1,2,4].
- 1 in SAMPLE [1,2,4] means that the system automatically selects 1 edge to continue traversal in the first step. 2 means to select 2 edges to continue traversal in the second step. 4 indicates that 4 edges are selected to continue traversal in the third step. If there is no matched edge in a certain step or the number of matched edges is less than the specified number, the actual number will be returned.
- Because 60 1 T0 3 STEPS means to return all the traversal results from the first to third steps, all the red edges and their source and destination vertices in the figure below will be matched by this 60 statement. And the yellow edges represent there is no path selected when the GO statement traverses. If it is not 60 1 T0 3 STEPS but 60 3 STEPS, it will only match the red edges of the third step and the vertices at both ends.



#### In the basketballplayer dataset, the example is as follows:

YIELD properties(\$\$).name AS NAME, properties(\$\$).age AS Age \
SAMPLE [1,2,3];
+----+
HANT

NAME	Age
"Tony Parker"   "Manu Ginobili'   "Spurs" +	36   41  NULL

Last update: October 25, 2023

### 4.6.4 ORDER BY

The ORDER BY clause specifies the order of the rows in the output.

- Native nGQL: You must use a pipe ( | ) and an ORDER BY clause after YIELD clause.
- OpenCypher style: No pipes are permitted. The ORDER BY clause follows a RETURN clause.

There are two order options:

- ASC : Ascending. ASC is the default order.
- DESC : Descending.

### Native nGQL Syntax

```
<YIELD clause>
| ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

# **P**mpatibility

In the native nGQL syntax, \$-. must be used after ORDER BY. But it is not required in releases prior to 2.5.0.

EXAMPLES

#### **OpenCypher Syntax**

```
<RETURN clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

EXAMPLES

nebula> MATCH (v:player) RETURN v.player.name AS Name, v.player.age AS Age \ ORDER BY Name DESC;

•••

# In the following example, nGQL sorts the rows by age first. If multiple people are of the same age, nGQL will then sort them by name. nebula> MATCH (v:player) RETURN v.player.age AS Age, v.player.name AS Name \ ORDER BY Age DESC, Name ASC;

+----+ | Age | Name | +----+ | 47 | "Shaquille O'Neal" |

46	"Grant	Hill"	
45	"Jason	Kidd"	
45	"Steve	Nash"	
++-		+	

### Order of NULL values

nGQL lists NULL values at the end of the output for ascending sorting, and at the start for descending sorting.



Last update: October 25, 2023

### **4.6.5 RETURN**

The RETURN clause defines the output of an nGQL query. To return multiple fields, separate them with commas.

RETURN can lead a clause or a statement:

- A RETURN clause can work in openCypher statements in nGQL, such as MATCH or UNWIND.
- A RETURN statement can work independently to output the result of an expression.

#### OpenCypher compatibility

This topic applies to the openCypher syntax in nGQL only. For native nGQL, use VIELD.

 ${\tt RETURN}$  does not support the following openCypher features yet.

• Return variables with uncommon characters, for example:

```
MATCH (`non-english_characters`:player) \
RETURN `non-english_characters`;
```

• Set a pattern in the RETURN clause and return all elements that this pattern matches, for example:

```
MATCH (v:player) \
RETURN (v)-[e]->(v2);
```

#### Map order description

When RETURN returns the map data structure, the order of key-value pairs is undefined.

```
nebula> RETURN {age: 32, name: "Marco Belinelli"};
+-----+
| {age:32, name: "Marco Belinelli"}
| +----+
| {age: 32, name: "Marco Belinelli"}
+----+
nebula> RETURN {zage: 32, name: "Marco Belinelli"}
+---+
| {zage:32, name: "Marco Belinelli"}
+---+
| {name: "Marco Belinelli", zage: 32}
+----+
```

#### Return vertices or edges

Use the RETURN {<vertex\_name> | <edge\_name>} to return vertices and edges all information.

#### **Return VIDs**

Use the id() function to retrieve VIDs.

#### **Return Tag**

Use the labels() function to return the list of tags on a vertex.

To retrieve the nth element in the labels(v) list, use labels(v)[n-1]. The following example shows how to use labels(v)[0] to return the first tag in the list.

### **Return properties**

When returning properties of a vertex, it is necessary to specify the tag to which the properties belong because a vertex can have multiple tags and the same property name can appear on different tags.

It is possible to specify the tag of a vertex to return all properties of that tag, or to specify both the tag and a property name to return only that property of the tag.

nebula>	MATCH (v:player) \ RETURN v.player, v.player.name, LIMIT 3;	v.player.age \	
+	yer	v.player.name	v.player.age
{age:   {age:   {age:   {age:	<pre>33, name: "LaMarcus Aldridge"} 25, name: "Kyle Anderson"} 40, name: "Kobe Bryant"}</pre>	"LaMarcus Aldridge"     "Kyle Anderson"     "Kobe Bryant"	33   25   40

When returning edge properties, it is not necessary to specify the edge type to which the properties belong, because an edge can only have one edge type.

e.start_year   e.degree	
++   NULL   95	
NULL 95	
1997  NULL   ++	

#### Return edge type

Use the type() function to return the matched edge types.

#### **Return paths**

Use RETURN <path\_name> to return all the information of the matched paths.

RETURN VERTICES IN A PATH

Use the nodes() function to return all vertices in a path.

RETURN EDGES IN A PATH

Use the relationships() function to return all edges in a path.

RETURN PATH LENGTH

Use the length() function to return the length of a path.

+----+

Paths
Length
+
++
<("player100" :player1age: 42, name: "Tim Duncan"})-[:serye@0 {end year: 2016, start year: 1997}]->("team204" :team1name:
"Spurs"})>  1
<("player100" :player14ge: 42, name: "Tim Duncan"})-[:follow00 {degree: 95}]->("player101" :player13ge: 36, name: "Tony
Parker"})>  1
<("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu
Ginobili"}>>   1
<("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:serve@0 {end_year: 2018, start_year: 1999}]-
>("team204" :team{name: "Spurs"})> 2
<("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:serve@0 {end_year: 2019, start_year: 2018}]-
>("team215" :team{name: "Hornets"})>   2
<("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 95}]->("player100" :player{age: 42,
name: "Tim Duncan"})>   2
<pre></pre> ("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 90}]->("player102" :player{age: 33,
name: "LaMarcus Aldridge"})>   2
<pre></pre> ("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41,
name: "Manu Ginobili"}>>   2
<pre></pre>
>("team204" :team{name: "Spurs"})>   2
<("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})-[:follow@0 {degree: 90}]->("player100" :player125
name: "Tim Duncan"})>   2
+

### **Return all elements**

To return all the elements that this pattern matches, use an asterisk  $(\ensuremath{^*}).$ 

<pre>nebula&gt; MATCH (v:player{name:"Tim Duncan"}) \</pre>		
v		
("player100" :player{age: 42, name: "Tim Duncan"})		
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) RETURN *;	\	
v	e	v2
<pre>("player100" :player{age: 42, name: "Tim Duncan"}) ("player100" :player{age: 42, name: "Tim Duncan"}) ("player100" :player{age: 42, name: "Tim Duncan"}))</pre>	<pre>[:follow "player100"-&gt;"player101" @0 {degree: 95}] [:follow "player100"-&gt;"player125" @0 {degree: 95}] [:serve "player100"-&gt;"team204" @0 {end_year: 2016, start_year: 1997}]</pre>	("player101" :player{age: 36, name: "Tony Parker"})     ("player125" :player{age: 41, name: "Manu Ginobili"})     ("team204" :team{name: "Spurs"}) + 

### Rename a field

Use the AS <alias> syntax to rename a field in the output.

<pre>nebula&gt; MATCH (v:player{name:"Tim Duncan"})-[:serve]-&gt;(v2) \</pre>
RETURN v2.team.name AS Team;
++
Team
++
"Spurs"
++
nebula> RETURN "Amber" AS Name;
++
Name
++
"Amber"
++

### Return a non-existing property

If a property matched does not exist,  $\ensuremath{\operatorname{\mathsf{NULL}}}$  is returned.

nebula> MATCH (v:p RETURN v2	olayer{name:" .player.name,	Tim Duncan"})-[e type(e), v2.pla	e]->(v2) ∖ ayer.age;
+	-++		ŀ
v2.player.name	type(e)	v2.player.age	
+	++		F
"Manu Ginobili"	"follow"	41	
NULL	"serve"	NULL	
"Tony Parker"	"follow"	36	
1			

### **Return expression results**

To return the results of expressions such as literals, functions, or predicates, set them in a  $\ensuremath{\mathsf{RETURN}}$  clause.

<pre>nebula&gt; MATCH (v:player{name:"Tony Parker"})&gt;(v2:player) \</pre>			
v2.player.name	("Hello"+" graphs!")	(v2.player.age>35)	
-   "LaMarcus Aldridge"   "Tim Duncan"   "Manu Ginobili" +	"Hello graphs!"   "Hello graphs!"   "Hello graphs!"   "Hello graphs!"	false     true     true	
nebula> RETURN 1+1; ++   (1+1)   ++   2   ++			
nebula> RETURN 11; ++   (1(1))   ++   2   ++			
<pre>nebula&gt; RETURN 3 &gt; 1; ++   (3&gt;1)   ++   true   ++</pre>			
nebula> RETURN 1+1, ra ++   (1+1)   rand32(1,5) ++   2   1 ++	nd32(1, 5); +   +   +		

### Return unique fields

Use  $\ensuremath{\texttt{DISTINCT}}$  to remove duplicate fields in the result set.

<pre># Before using DISTINCT. nebula&gt; MATCH (v:player{name:"Tony Parker"})(v2:player) \</pre>			
   v2.player.age   +			
41       36       32       29       42       43       33			
<pre># After using DISTINCT. nebula&gt; MATCH (v:player{name:"Tony Parker"})(v2:player) \ RETURN DISTINCT v2.player.name, v2.player.age; +</pre>			
v2.player.age			
41 36 32 29 42 33			

Last update: October 25, 2023

### 4.6.6 TTL

TTL (Time To Live) is a mechanism in NebulaGraph that defines the lifespan of data. Once the data reaches its predefined lifespan, it is automatically deleted from the database. This feature is particularly suitable for data that only needs temporary storage, such as temporary sessions or cached data.

### **OpenCypher Compatibility**

This topic applies to native nGQL only.

### Precautions

- You CANNOT modify a property schema with TTL options on it.
- TTL options and indexes have coexistence issues.
- TTL options and indexes CANNOT coexist on a tag or an edge type. If there is an index on a property, you cannot set TTL options on other properties.
- If there are TTL options on a tag, an edge type, or a property, you can still add an index on them.

#### TTL options

The native nGQL TTL feature has the following options.

Option	Description
ttl_col	Specifies an existing property to set a lifespan on. The data type of the property must be $int$ or timestamp.
ttl_duration	Specifies the timeout adds-on value in seconds. The value must be a non-negative int64 number. A property expires if the sum of its value and the ttl_duration value is smaller than the current timestamp. If the ttl_duration value is 0, the property never expires. You can set ttl_use_ms to true in the configuration file nebula-storaged.conf (default path: /usr/local/nightly/etc/) to set the default unit to milliseconds.

### Arning

• Before setting ttl\_use\_ms to true, make sure that no TTL has been set for any property, as shortening the expiration time may cause data to be erroneously deleted.

• After setting ttl\_use\_ms to true, which sets the default TTL unit to milliseconds, the data type of the property specified by ttl\_col must be int, and the property value needs to be manually converted to milliseconds. For example, when setting ttl\_col to a, you need to convert the value of a to milliseconds, such as when the value of a is now(), you need to set the value of a to now() \* 1000.

### Use TTL options

You must use the TTL options together to set a lifespan on a property.

Before using the TTL feature, you must first create a timestamp or integer property and specify it in the TTL options. NebulaGraph will not automatically create or manage this timestamp property for you.

When inserting the value of the timestamp or integer property, it is recommended to use the now() function or the current timestamp to represent the present time.

SET A TIMEOUT IF A TAG OR AN EDGE TYPE EXISTS

If a tag or an edge type is already created, to set a timeout on a property bound to the tag or edge type, use ALTER to update the tag or edge type.

# Create a tag. nebula> CREATE TAG IF NOT EXISTS t1 (a timestamp);

# Use ALTER to update the tag and set the TTL options. nebula> ALTER TAG t1 TTL\_COL = "a", TTL\_DURATION = 5;

# Insert a vertex with tag t1. The vertex expires 5 seconds after the insertion. nebula> INSERT VERTEX t1(a) VALUES "101":(now());

# Create a edge type.
nebula> CREATE EDGE IF NOT EXISTS e1 (a timestamp);

# Use ALTER to update the edge type and set the TTL options. nebula> ALTER EDGE e1 TTL\_COL = "a", TTL\_DURATION = 5;

# Insert a dangling edge with edge type el. The edge expires 5 seconds after the insertion. nebula> INSERT EDGE el (a) VALUES "10"->"11":(now());

SET A TIMEOUT WHEN CREATING A TAG OR AN EDGE TYPE

Use TTL options in the CREATE statement to set a timeout when creating a tag or an edge type. For more information, see CREATE TAG and CREATE EDGE.

```
# Create a tag and set the TTL options.
nebula> CREATE TAG IF NOT EXISTS t2(a int, b int, c string) TTL_DURATION= 100, TTL_COL = "a";
# Insert a vertex with tag t2. The timeout timestamp is 1648197238 (1648197138 + 100).
```

nebula> INSERT VERTEX t2(a, b, c) VALUES "102":(1648197138, 30, "Hello");

## Caution

Data expiration and deletion

• When the TTL options are set for a property of a tag or an edge type and the property's value is NULL, the property never expires.

• If a property with a default value of now() is added to a tag or an edge type and the TTL options are set for the property, the history data related to the tag or the edge type will never expire because the value of that property for the history data is the current timestamp.

VERTEX PROPERTY EXPIRATION

Vertex property expiration has the following impact.

- If a vertex has only one tag, once a property of the vertex expires, the vertex expires.
- If a vertex has multiple tags, once a property of the vertex expires, properties bound to the same tag with the expired property also expire, but the vertex does not expire and other properties of it remain untouched.

EDGE PROPERTY EXPIRATION

Since an edge can have only one edge type, once an edge property expires, the edge expires.

DATA DELETION

The expired data are still stored on the disk, but queries will filter them out.

NebulaGraph automatically deletes the expired data and reclaims the disk space during the next compaction.

#### Q Note

If TTL is disabled, the corresponding data deleted after the last compaction can be queried again.

### **Remove a timeout**

To disable TTL and remove the timeout on a property, you can use the following approaches.

• Drop the property with the timeout.

nebula> ALTER TAG t1 DROP (a);

• Set ttl\_col to an empty string.

nebula> ALTER TAG t1 TTL\_COL = "";

• Set ttl\_duration to 0. This operation keeps the TTL options and prevents the property from expiring and the property schema from being modified.

nebula> ALTER TAG t1 TTL\_DURATION = 0;

Last update: April 8, 2024

### 4.6.7 WHERE

The  $\ensuremath{\mathsf{WHERE}}$  clause filters the output by conditions.

The  $\ensuremath{\mathsf{WHERE}}$  clause usually works in the following queries:

- $\bullet$  Native nGQL: such as 60 and LOOKUP .
- OpenCypher syntax: such as MATCH and WITH.

### OpenCypher compatibility

Filtering on edge rank is a native nGQL feature. To retrieve the rank value in openCypher statements, use the rank() function, such as MATCH (:player)-[e:follow]->() RETURN rank(e); .

### Basic usage

Note
In the following examples, \$\$ and \$^ are reference operators. For more information, see Operators.

DEFINE CONDITIONS WITH BOOLEAN OPERATORS

Use the boolean operators NOT, AND, OR, and XOR to define conditions in WHERE clauses. For the precedence of the operators, see Precedence.

nebula> MATCH (v:player) \ WHERE v.player.nam XOR (v.player.age OR NOT (v.player.na RETURN v.player.na	e == "Tim Duncan" \ < 30 AND v.player.name == "Yao Ming") \ ame == "Yao Ming" OR v.player.name == "Tim Duncan") \ me, v.player.age;
v.player.name	v.player.age
"Danny Green"   "Tiago Splitter"   "David West" 	31.     34.     38.
nebula> GO FROM "player100 OVER follow \ WHERE properties(e OR properties(\$\$). AND properties(\$\$) YIELD properties(\$	" \ dge).degree > 90 \ age != 33 \ .name != "Tony Parker" \ \$);
properties(\$\$) +	+
{age: 41, name: "Manu Gi	nobili"}

#### FILTER ON PROPERTIES

Use vertex or edge properties to define conditions in WHERE clauses.

### • Filter on a vertex property:

#### • Filter on an edge property:

nebula> MATCH (v:playe WHERE e.start RETURN DISTING	er)-[e]->() \ _year < 2000 \ CT v.player.name, v.player.age;
v.player.name	v.player.age
"Tony Parker"   "Tim Duncan"   "Grant Hill" 	36     42     46
nebula> GO FROM "playe	er100" OVER follow \

FILTER ON DYNAMICALLY-CALCULATED PROPERTIES

nebula> MATCH (v:player) \ WHERE v[toLower("AGE")] < 21 \ RETURN v.player.name, v.player.	age;
++	
v.name v.age	
++	
"Luka Doncic"   20   ++	

#### FILTER ON EXISTING PROPERTIES

nebula> MATCH (v:player) WHERE exists(v.p RETURN v.player.	\ layer.age) \ name, v.player.age;
+	++
v.player.name	v.player.age
+	++
"Danny Green"	31
"Tiago Splitter"	34
"David West"	38

#### FILTER ON EDGE RANK

In nGQL, if a group of edges has the same source vertex, destination vertex, and properties, the only thing that distinguishes them is the rank. Use rank conditions in WHERE clauses to filter such edges.

# The following example creates test data. nebula> CREATE SPACE IF NOT EXISTS test (vid\_type=FIXED\_STRING(30)); nebula> USE test; nebula> CREATE EDGE IF NOT EXISTS e1(p1 int); nebula> CREATE TAG IF NOT EXISTS person(p1 int); nebula> INSERT VERTEX person(p1) VALUES "1":(1); nebula> INSERT VERTEX person(p1) VALUES "2":(2); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@0:(10); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@1:(11); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@2:(12); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@2:(12); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@3:(13); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@4:(14); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@5:(15); nebula> INSERT EDGE e1(p1) VALUES "1"->"2"@6:(16); # The following example use rank to filter edges and retrieves edges with a rank greater than 2. nebula> G0 FROM "1"  $\backslash$ OVER e1 \ WHERE rank(edge) > 2 \ YIELD src(edge), dst(edge), rank(edge) AS Rank, properties(edge).p1 | \ ORDER BY \$-.Rank DESC; | src(EDGE) | dst(EDGE) | Rank | properties(EDGE).pl "1" "2" 6 16 "1" "2" 5 15 "2" "2" "1" "1" | 4 | 3 14 | 13 # Filter edges by rank. Find follow edges with rank equal to 0. nebula> MATCH (v)-[e:follow]->() \ WHERE rank(e)==0 \ RETURN \*; l v l e ("player142" :player{age: 29, name: "Klay Thompson"}) [:follow "player142"->"player117" @0 {degree: 90}] ("player132" :player{age: 34, name: "Ktay finampson ("player139" :player{age: 34, name: "Marc Gasol"}) ("player108" :player{age: 36, name: "Boris Diaw"}) [:follow "player139"->"player138" @0 {degree: 30}] [:follow "player139"->"player138" @0 {degree: 30}] [:follow "player108"->"player100" @0 {degree: 80}] [:follow "player108"->"player101" @0 {degree: 80}] ("player108" :player{age: 36, name: "Boris Diaw"}) FILTER ON PATTERN nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(t) \ WHERE (v)-[e]->(t:team) \ RETURN (v)-->(); | (v) - ->() = (v) - ->() [ [<("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end\_year: 2016, start\_year: 1997}]->("team204" :team{name: "Spurs"})>, <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})>, <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})>] | +--nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(t)  $\ \$  WHERE NOT (v)-[e]->(t:team)  $\ \$ RETURN (v)-->(); | (v) - ->() = (v) - ->() [<("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end\_year: 2016, start\_year: 1997}]->("team204" :team{name: "Spurs"})>, <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"})>, <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})>] | | [<("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end\_year: 2016, start\_year: 1997}]->("team204" :team{name: "Spurs"})>, <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player101" :player{age: 36, name: "Tony Parker"}>>, <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})>] |

#### Filter on strings

+----

Use STARTS WITH, ENDS WITH, or CONTAINS in WHERE clauses to match a specific part of a string. String matching is case-sensitive.

#### STARTS WITH

STARTS WITH will match the beginning of a string.

The following example uses STARTS WITH "T" to retrieve the information of players whose name starts with T .

nebula> MATCH (v:pla WHERE v.play RETURN v.pla	ayer) \ /er.name STARTS WITH "T" ayer.name, v.player.age;
+	++
v.player.name	v.player.age
+	++
"Tony Parker"	36
"Tiago Splitter"	34
"Tim Duncan"	42
"Tracy McGrady"	39
+	+

If you use STARTS WITH "t" in the preceding statement, an empty set is returned because no name in the dataset starts with the lowercase t.

```
nebula> MATCH (v:player) \
WHERE v.player.name STARTS WITH "t" \
        RETURN v.player.name, v.player.age;
| v.player.name | v.player.age |
+----+----
Empty set (time spent 5080/6474 us)
```

#### ENDS WITH

ENDS WITH will match the ending of a string.

The following example uses ENDS WITH "r" to retrieve the information of players whose name ends with r.

```
nebula> MATCH (v:player) \
           WHERE v.player.name ENDS WITH "r" \
RETURN v.player.name, v.player.age;
v.player.name v.player.age
  "Tony Parker" | 36
"Tiago Splitter" | 34
"Vince Carter" | 42
```

#### CONTAINS

CONTAINS will match a certain part of a string.

The following example uses CONTAINS "Pa" to match the information of players whose name contains Pa.

nebula> MATCH (v: WHERE v.p RETURN v.	player) \ olayer.name CONTAINS "Pa" \ player.name, v.player.age;
+	+
v.player.name   +	v.player.age
"Paul George"	28
"Tony Parker"	36
"Paul Gasol"	38
"Chris Paul"	33
+	++

NEGATIVE STRING MATCHING

You can use the boolean operator NOT to negate a string matching condition.

nebula> MATCH (v:player)	\
WHERE NOT v.play	er.name ENDS WITH "R"
RETURN v.player.	name, v.player.age;
+	++
v.player.name	v.player.age
+	++
"Danny Green"	31
"Tiago Splitter"	34
"David West"	38
"Russell Westbrook"	30

### Filter on lists

MATCH VALUES IN A LIST

Use the  ${\,\rm I\!N}\,$  operator to check if a value is in a specific list.

nebula> MATCH (v:player) ∖ WHERE v.player.age RETURN v.player.nar	IN range(20,25) ∖ me, v.player.age;
v.plaver.name	++   v.plaver.age
+	++
"Ben Simmons"	22
"Giannis Antetokounmpo"	24
"Kyle Anderson"	25
"Joel Embiid"	25
"Kristaps Porzingis"	23
"Luka Doncic"	20
+	++
nebula> LOOKUP ON player \ WHERE player.age IM YIELD properties(ve	N [25,28] \ ertex).name, properties(vertex).age;
+	++
properties(VERTEX).name	properties(VERTEX).age
+	++
"Kyle Anderson"	25
"Damian Lillard"	28
Joel Embiid"	25
Paul George"	28
"Ricky Rubio"	28
+	++

MATCH VALUES NOT IN A LIST

Use  $\tt NOT$  before  $\tt IN$  to rule out the values in a list.

```
nebula> MATCH (v:player) \

WHERE v.player.age NOT IN range(20,25) \

RETURN v.player.name AS Name, v.player.age AS Age \

ORDER BY Age;

------+

Name | Age |

-----+

| "Kyrie Irving" | 26 |

"Cory Joseph" | 27 |

"Damian Lillard" | 28 |

"Paul George" | 28 |

-----+

----+

...
```

Last update: January 31, 2024

### 4.6.8 YIELD

YIELD defines the output of an nGQL query.

YIELD can lead a clause or a statement:

• A YIELD clause works in nGQL statements such as 60, FETCH, or LOOKUP and must be defined to return the result.

• A YIELD statement works in a composite query or independently.

### OpenCypher compatibility

This topic applies to native nGQL only. For the openCypher syntax, use RETURN.

YIELD has different functions in openCypher and nGQL.

• In openCypher, YIELD is used in the CALL[...YIELD] clause to specify the output of the procedure call.

#### O Note

NGQL does not support  ${\tt CALL[...YIELD]}$  yet.

• In nGQL, YIELD works like RETURN in openCypher.

## Note

In the following examples, \$\$ and \$- are property references. For more information, see Reference to properties.

#### **YIELD clauses**

#### SYNTAX

YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...];

Parameter	Description
DISTINCT	Aggregates the output and makes the statement return a distinct result set.
col	A field to be returned. If no alias is set, cot will be a column name in the output.
alias	An alias for col. It is set after the keyword AS and will be a column name in the output.

USE A YIELD CLAUSE IN A STATEMENT

### • Use YIELD with GO:

| "Manu Ginobili" | 41 |

#### • Use YIELD with FETCH:

### • Use YIELD with LOOKUP:

nebula>	LOOKUF YIELD	'ON player properties	WH (ve	<pre>ERE player.name == "Tony Parker" \ rtex).name, properties(vertex).age</pre>
+			+	+
prope	rties(\	/ERTEX).nam	e	<pre>properties(VERTEX).age  </pre>
+	Parker	."	+	36
+			+	+

YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...]

#### **YIELD statements**

#### SYNTAX

[WHERE <conditions>];</conditions>	
Parameter	Description
DISTINCT	Aggregates the output and makes the statement return a distinct result set.
col	A field to be returned. If no alias is set, cot will be a column name in the output.
alias	An alias for $\operatorname{col}$ . It is set after the keyword AS and will be a column name in the output.
conditions	Conditions set in a $\ensuremath{WHERE}$ clause to filter the output. For more information, see $\ensuremath{WHERE}$ .

USE A YIELD STATEMENT IN A COMPOSITE QUERY

In a composite query, a VIELD statement accepts, filters, and modifies the result set of the preceding statement, and then outputs it.

The following query finds the players that "player100" follows and calculates their average age.

```
nebula> G0 FROM "player100" OVER follow \
    YIELD dst(edge) AS ID \
    | FETCH PROP ON player $-.ID \
    YIELD properties(vertex).age AS Age \
    | YIELD AVG($-.Age) as Avg_age, count(*)as Num_friends;
+-----+
    Avg_age | Num_friends |
+-----+
    38.5 | 2 |
```

The following query finds the players that "player101" follows with the follow degrees greater than 90.

The following query finds the vertices in the player that are older than 30 and younger than 32, and returns the de-duplicate results.

```
nebula> LOOKUP ON player \
WHERE player.age < 32 and player.age >30 \
YIELD DISTINCT properties(vertex).age as v;
+------+
| v |
+------+
```

USE A STANDALONE YIELD STATEMENT

A YIELD statement can calculate a valid expression and output the result.

```
nebula> YIELD rand32(1, 6);
| rand32(1,6) |
+----
| 3
+-----
nebula> YIELD "Hel" + "\tlo" AS string1, ", World!" AS string2;
| "Hel lo" | ", World!" |
nebula> YIELD hash("Tim") % 100;
| (hash(Tim)%100) |
| 42
+----
nebula> YIELD \
CASE 2+3 \
     WHEN 4 THEN 0 \
WHEN 5 THEN 1 \
ELSE -1 \
     END \
AS result;
+-----
| result |
| 1
+----+
nebula> YIELD 1- -1;
| (1--(1)) |
| 2
```

Last update: January 17, 2024
### 4.6.9 WITH

The WITH clause can retrieve the output from a query part, process it, and pass it to the next query part as the input.

#### OpenCypher compatibility

This topic applies to openCypher syntax only.

## Note

WITH has a similar function with the Pipe symbol in native nGQL, but they work in different ways. DO NOT use pipe symbols in the openCypher syntax or use WITH in native nGQL statements.

#### Combine statements and form a composite query

Use a WITH clause to combine statements and transfer the output of a statement as the input of another statement.

EXAMPLE 1

The following statement:

- 1. Matches a path.
- 2. Outputs all the vertices on the path to a list with the nodes() function.
- 3. Unwinds the list into rows.
- 4. Removes duplicated vertices and returns a set of distinct vertices.

EXAMPLE 2

The following statement:

- 1. Matches the vertex with the VID player100.
- 2. Outputs all the tags of the vertex into a list with the labels() function.
- 3. Unwinds the list into rows.
- 4. Returns the output.

```
nebula> MATCH (v) \

WHERE id(v)=="player100" \

WITH labels(v) AS tags_unf \

UNWIND tags_unf AS tags_f \

RETURN tags_f;

+-----+

+ tags_f |

+-----+

| "player" |

+-----+
```

#### Filter composite queries

WITH can work as a filter in the middle of a composite query.

```
nebula> MATCH (v:player)-->(v2:player) \
WITH DISTINCT v2 AS v2, v2.player.age AS Age \
ORDER BY Age \
WHERE Age<25 \
RETURN v2.player.name AS Name, Age;
+----+
Name | Age |
----+
| Name | Age |
----+
| "Luka Doncic" | 20 |
"Ben Simmons" | 22 |
"Kristaps Porzingis" | 23 |
----+
```

#### Process the output before using collect()

Use a WITH clause to sort and limit the output before using collect() to transform the output into a list.



#### Use with RETURN

Set an alias using a wITH clause, and then output the result through a  ${\tt RETURN}$  clause.



Last update: November 3, 2023

## 4.6.10 UNWIND

UNWIND transform a list into a sequence of rows.

UNWIND can be used as an individual statement or as a clause within a statement.

#### **UNWIND statement**

SYNTAX

UNWIND <list> AS <alias> <RETURN clause>;

#### EXAMPLES

• To transform a list.

```
nebula> UNWIND [1,2,3] AS n RETURN n;
+---+
| n |
+---+
| 1 |
| 2 |
| 3 |
```

#### UNWIND clause

SYNTAX

• The UNWIND clause in native nGQL statements.

## Note

To use a UNWIND clause in a native nGQL statement, use it after the | operator and use the \$- prefix for variables. If you use a statement or clause after the UNWIND clause, use the | operator and use the \$- prefix for variables.

<statement> | UNWIND \$-.<var> AS <alias> <|> <clause>;

• The UNWIND clause in openCypher statements.

<statement> UNWIND <list> AS <alias> <RETURN clause> ;

#### EXAMPLES

• To transform a list of duplicates into a unique set of rows using WITH DISTINCT in a UNWIND clause.

#### Q Note

WITH DISTINCT is not available in native nGQL statements.

// Transform the list `[1,1,2,2,3,3]` into a unique set of rows, sort the rows, and then transform the rows into a list of unique values.

```
nebula> WITH [1,1,2,2,3,3] AS n \
    UNWIND n AS r \
    WITH DISTINCT r AS r \
    ORDER BY r \
    RETURN collect(r);
+-----+
    collect(r) |
+-----+
```

| [1, 2, 3] |

#### • To use an UNWIND clause in a MATCH statement.

// Get a list of the vertices in the matched path, transform the list into a unique set of rows, and then transform the rows into a list.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--(v2) \
WITH nodes(p) AS n \
UWWIND n AS r \
WITH nodes(p) AS n \
UWWIND n AS r \
KETURN collect(r);

("learn204" :team{name: "Spurs"}), ("player102" :player[age: 36, name: "Tony Parker"}),
[("tearn204" :team{name: "Spurs"}), ("player102" :player[age: 33, name: "LaMarcus Aldridge"}),
[("player125" :player[age: 41, name: "Manu Ginobili"}), ("player104" :player[age: 32, name: "Marco Belinelli"}),
[("player144" :player[age: 42, name: "Dinu Ginobili"}), ("player107" :player[age: 31, name: "Danny Green"}),
[("player131" :player[age: 29, name: "Dejounte Murray"}), ("player107" :player[age: 32, name: "Koro Baynes"}),
[("player109" :player[age: 34, name: "Tiago Splitter"}), ("player108" :player[age: 36, name: "Boris Diaw"})]
```

• To use an UNWIND clause in a GO statement.

 $/\!/$  Query the vertices in a list for the corresponding edges with a specified statement.

• To use an UNWIND clause in a LOOKUP statement.

// Find all the properties of players whose age is greater than 46, get a list of unique properties, and then transform the list into rows.

• To use an UNWIND clause in a FETCH statement.

// Query player101 for all tags related to player101, get a list of the tags and then transform the list into rows.

```
nebula> CREATE TAG hero(like string, height int);
INSERT VERTEX hero(like, height) VALUES "player101":("deep", 182);
FETCH PROP ON * "player101" \
YIELD tags(vertex) as t | UNWIND $-.t as a | YIELD $-.a AS a;
a |
+------+
| "hero" |
| "player" |
+------+
```

## . To use an UNWIND clause in a GET SUBGRAPH statement.

// Get the subgraph including outgoing and incoming serve edges within 0-2 hops from/to player100, and transform the result into rows.

nebula> GET SUBGRAPH 2 STEPS FROM "player100" BOTH serve \
TIELD EUges as e   UNWIND \$e as a   TIELD \$a As a;
a   ++
[:serve "nlaver100"->"team204" 60 {}]
[:serve "player101"->"team204" @0 {}]
[:serve "player101"->"team204" @0 {}]
$[:serve "player102" + team204" (0 {}]]$
[:serve "player105"->"team204" @0 {}]
[:serve player105 -> team204 @0 {}]
[:serve player100 -> team204 @0 {}]
[:Serve player107 -> leam204 @0 {}]
[:serve player108"->"team204" @0 {}]
[:serve "player109"->"team204" @0 {}]
[:serve "player110"->"team204" @0 {}]
[:serve "player111"->"team204" @0 {}]
[:serve "player112"->"team204" @0 {}]
[:serve "player113"->"team204" @0 {}]
[:serve "player114"->"team204" @0 {}]
[:serve "player125"->"team204" @0 {}]
[:serve "player138"->"team204" @0 {}]
[:serve "player104"->"team204" @20132015 {}]
[:serve "player104"->"team204" @20182019 {}]

#### • To use an UNWIND clause in a FIND PATH statement.

// Find all the vertices in the shortest path from player101 to team204 along the serve edge, and transform the result into rows.

## 4.7 Variables and composite queries

#### 4.7.1 Composite queries (clause structure)

Composite queries put data from different queries together. They then use filters, group-bys, or sorting before returning the combined return results.

Nebula Graph supports three methods to run composite queries (or sub-queries):

- (openCypher) Clauses are chained together, and they feed intermediate result sets between each other.
- (Native nGQL) More than one query can be batched together, separated by semicolons (;). The result of the last query is returned as the result of the batch.
- (Native nGQL) Queries can be piped together by using the pipe (||). The result of the previous query can be used as the input of the next query.

#### OpenCypher compatibility

In a composite query, **do not** put together openCypher and native nGQL clauses in one statement. For example, this statement is undefined: MATCH ... | 60 ... | YIELD ....

- If you are in the openCypher way (MATCH, RETURN, WITH, etc), do not introduce any pipe or semicolons to combine the sub-clauses.
- If you are in the native nGQL way (FETCH, GO, LOOKUP, etc), you must use pipe or semicolons to combine the sub-clauses.

#### Composite queries are not transactional queries (as in SQL/Cypher)

For example, a query is composed of three sub-queries:  $A \ B \ C$ ,  $A \ B \ C$  or A; B; C. In that A is a read operation, B is a computation operation, and C is a write operation. If any part fails in the execution, the whole result will be undefined. There is no rollback. What is written depends on the query executor.

#### Q Note

OpenCypher has no requirement of transaction.

#### Examples

· OpenCypher compatibility statement

```
# Connect multiple queries with clauses.
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
WITH nodes(p) AS n \
```

UNWIND n AS n1 \ RETURN DISTINCT n1;

#### • Native nGQL (Semicolon queries)

# Only return edges. nebula> SHOW TAGS; SHOW EDGES;

# Insert multiple vertices. nebula> INSERT VERTEX player(name, age) VALUES "player100":("Tim Duncan", 42); \ INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36); \ INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);

#### • Native nGQL (Pipe queries)

```
# Connect multiple queries with pipes.
nebula> G0 FROM "player100" OVER follow YIELD dst(edge) AS id | \
        G0 FROM $-.id OVER serve YIELD properties($$).name AS Team, \
        properties($^).name AS Player;
+-----+
| Team | Player |
+-----+
| "Spurs" | "Tony Parker" |
| "Hornets" | "Tony Parker" |
| "Spurs" | "Manu Ginobili" |
```

#### 4.7.2 User-defined variables

User-defined variables allow passing the result of one statement to another.

#### OpenCypher compatibility

In openCypher, when you refer to the vertex, edge, or path of a variable, you need to name it first. For example:

The user-defined variable in the preceding query is  $\overline{v}$ .

## Caution

In a pattern of a MATCH statement, you cannot use the same edge variable repeatedly. For example, e cannot be written in the pattern  $p=(v1)-[e^{*}2..2]->(v2)-[e^{*}2..2]->(v3)$ .

#### Native nGQL

User-defined variables are written as \$var\_name. The var\_name consists of letters, numbers, or underline characters. Any other characters are not permitted.

The user-defined variables are valid only at the current execution (namely, in this composite query). When the execution ends, the user-defined variables will be automatically expired. The user-defined variables in one statement **CANNOT** be used in any other clients, executions, or sessions.

You can use user-defined variables in composite queries. Details about composite queries, see Composite queries.

Note		
User-defined variables are case-sensitive.		

• To define a user-defined variable in a compound statement, end the statement with a semicolon (;). For details, please refer to the nGQL Style Guide.

#### Example

```
nebula> $var = G0 FROM "player100" OVER follow YIELD dst(edge) AS id; \
    G0 FROM $var.id OVER serve YIELD properties($$).name AS Team, \
    properties($^\).name AS Player;
+-----+
| Team | Player |
+----+
| "Spurs" | "Tony Parker" |
    "Hornets" | "Tony Parker" |
    "Spurs" | "Manu Ginobili" |
+----+
```

#### Set operations and scope of user-defined variables

When assigning variables within a compound statement involving set operations, it is important to enclose the scope of the variable assignment in parentheses. In the example below, the source of the *svar* assignment is the results of the output of two INTERSECT statements.

```
$var = ( \
    G0 FROM "player100" OVER follow \
    YIELD dst(edge) AS id \
```

INTERSECT \ GO FROM "player100" OVER follow \ YIELD dst(edge) AS id \ ); \ GO FROM \$var.id OVER follow YIELD follow.degree AS degree

## 4.7.3 Reference to properties

nGQL provides property references to allow you to refer to the properties of the source vertex, the destination vertex, and the edge in the 60 statement, and to refer to the output results of the statement in composite queries. This topic describes how to use these property references in nGQL.

Q Note			

This function applies to native  $n\mbox{GQL}$  only.

#### Property references for vertexes

Parameter	Description
Ş٨	Used to get the property of the source vertex.
\$\$	Used to get the property of the destination vertex.

PROPERTY REFERENCE SYNTAX

\$^.<tag\_name>.<prop\_name> # Source vertex property reference
\$\$.<tag\_name>.<prop\_name> # Destination vertex property reference

- tag\_name : The tag name of the vertex.
- prop\_name : The property name within the tag.

#### Property references for edges

Parameter	Description
_src	The source vertex ID of the edge
_dst	The destination vertex ID of the edge
_type	The internal encoding of edge types that uses sign to indicate direction. Positive numbers represent forward edges, while negative numbers represent backward edges.
_rank	The rank value for the edge

PROPERTY REFERENCE SYNTAX

nGQL allows you to reference edge properties, including user-defined edge properties and four built-in edge properties.

<edge\_type>.<prop\_name> # User-defined edge property reference
<edge\_type>.\_src|\_dst|\_type|\_rank # Built-in edge property reference

• edge\_type : The edge type.

• prop\_name : The property name within the edge type.

#### Property references for composite queries

Parameter	Description
\$-	Used to get the output results of the statement before the pipe in the composite query. For more information, see Pipe.

#### Examples

USE PROPERTY REFERENCES FOR VERTEXES

The following query returns the name property of the player tag on the source vertex and the age property of the player tag on the destination vertex.

```
nebula> GO FROM "player100" OVER follow YIELD $^.player.name AS startName, $$.player.age AS endAge;
```

```
| startName | endAge |
+-----+
| "Tim Duncan" | 36 |
| "Tim Duncan" | 41 |
+----++
```

## L jacy version compatibility

Starting from NebulaGraph 2.6.0, Schema-related functions are supported. The preceding example can be rewritten as follows in NebulaGraph 3.6.0 to produce the same results:

GO FROM "player100" OVER follow YIELD properties(\$^).name AS startName, properties(\$\$).age AS endAge;

NebulaGraph 3.6.0 is compatible with both new and old syntax.

USE PROPERTY REFERENCES FOR EDGES

The following query returns the degree property of the edge type follow.

nebula> GO FROM "player100" OVER follow YIELD follow.degree; +-----+ | follow.degree | +------+ | 95 | +-----+

The following query returns the source vertex, the destination vertex, the edge type, and the edge rank value of the edge type follow.

nebula> G0 FROM "player100" OVER follow YIELD follow.\_src, follow.\_dst, follow.\_type, follow.\_rank;

	followsrc	followdst	followtype	followrank	
   	"player100"   "player100"	"player101" "player125"	17   17	0   0	
1					÷.

# L Jacy version compatibility

Starting from NebulaGraph 2.6.0, Schema-related functions are supported. The preceding example can be rewritten as follows in NebulaGraph 3.6.0 to produce the same results:

GO FROM "player100" OVER follow YIELD properties(edge).degree; GO FROM "player100" OVER follow YIELD src(edge), dst(edge), type(edge), rank(edge);

NebulaGraph 3.6.0 is compatible with both new and old syntax.

USE PROPERTY REFERENCES FOR COMPOSITE QUERIES

The following composite query performs the following actions:

- 1. Uses the property reference \$-.id to get the results of the statement GO FROM "player100" OVER follow YIELD dst(edge) AS id, which returns the destination vertex ID of the follow edge type.
- 2. Uses the properties (<sup>()</sup> function to get the name property of the player tag on the source vertex of the serve edge type.
- 3. Uses the properties(\$\$) function to get the name property of the team tag on the destination vertex of the serve edge type.

nebula> GO FROM "p YIELD dst( GO FROM \$-	layer100" OVER follow \ edge) AS id   \ .id OVER serve \
YIELD prop	erties(\$^).name AS Player, properties(\$\$).name AS Team
+	++
Player	Team
+	++
"Tony Parker"	"Spurs"
"Tony Parker"	"Hornets"
"Manu Ginobili"	"Spurs"
+	++

Last update: January 31, 2024

## 4.8 Space statements

### 4.8.1 CREATE SPACE

Graph spaces are used to store data in a physically isolated way in NebulaGraph, which is similar to the database concept in MySQL. The CREATE SPACE statement can create a new graph space or clone the schema of an existing graph space.

#### Prerequisites

Only the God role can use the CREATE SPACE statement. For more information, see AUTHENTICATION.

#### Syntax

CREATE GRAPH SPACES

```
CREATE SPACE [IF NOT EXISTS] <graph_space_name> (
  [partition_num = <partition_number>,]
[replica_factor = <replica_number>,]
vid_type = {FIXED_STRING(<N>) | INT[64]}
  [COMMENT = '<comment>']
  Parameter
                           Description
  TE NOT EXTSTS
                           Detects if the related graph space exists. If it does not exist, a new one will be created. The graph space
                           existence detection here only compares the graph space name (excluding properties).
  <graph_space_name>
                           1. Uniquely identifies a graph space in a NebulaGraph instance.
                           2. Space names cannot be modified after they are set.
                           3. By default, the name only supports 1-4 byte UTF-8 encoded characters, including English letters (case
                           sensitive), numbers, Chinese characters, etc. However, it cannot include special characters other than the
                           underscore (_), and cannot start with a number.
                           4. To use special characters, reserved keywords, or start with a number, quote the entire name with
                           backticks (`) and do not include periods ( . ) within the pair of backticks (`). For more information, see
                           Keywords and reserved words.
                           Note:
                           1. If you name a space in Chinese and encounter a SyntaxError, you need to quote the Chinese characters
                           with backticks (`).
                           2. To include a backtick (`) in a space name, use a backslash to escape the backtick, such as \`; to include
                           a backslash, the backslash itself also needs to be escaped, such as \ .
  partition num
                           Specifies the number of partitions in each replica. The suggested value is 20 times (2 times for HDD) the
                           number of the hard disks in the cluster. For example, if you have three hard disks in the cluster, we
                           recommend that you set 60 partitions. The default value is 100.
  replica_factor
                           Specifies the number of replicas in the cluster. The suggested number is 3 in a production environment
                           and 1 in a test environment. The replica number must be an odd number for the need of quorum-based
                           voting. The default value is 1.
  vid_type
                           A required parameter. Specifies the VID type in a graph space. Available values are FIXED_STRING(N) and
                           INT64. INT equals to INT64.
                           `FIXED_STRING(<N>) specifies the VID as a string, while INT64 specifies it as an integer. N represents the
                           maximum length of the VIDs. If you set a VID that is longer than N bytes, NebulaGraph throws an error.
                           Note, for UTF-8 chars, the length may vary in different cases, i.e. a UTF-8 Chinese char is 3 byte, this
                           means 11 Chinese chars(length-33) will exeed a FIXED_STRING(32) vid defination.
  COMMENT
                           The remarks of the graph space. The maximum length is 256 bytes. By default, there is no comments on a
                           space.
```

## Caution

- If the replica number is set to one, you will not be able to load balance or scale out the NebulaGraph Storage Service with the SUBMIT JOB BALANCE statement.
- Restrictions on VID type change and VID length:
- For NebulaGraph v1.x, the type of VIDs can only be INT64, and the String type is not allowed. For NebulaGraph v2.x, both INT64 and FIXED\_STRING(<N>) VID types are allowed. You must specify the VID type when creating a graph space, and use the same VID type in INSERT statements, otherwise, an error message Wrong vertex id type: 1001 occurs.
- The length of the VID should not be longer than N characters. If it exceeds N, NebulaGraph throws The VID must be a 64-bit integer or a string fitting space vertex id length limit.

• If the Host not enough! error appears, the immediate cause is that the number of online storage hosts is less than the value of replica\_factor specified when creating a graph space. In this case, you can use the SHOW HOSTS command to see if the following situations occur:

- For the case where there is only one storage host in a cluster, the value of replica\_factor can only be specified to 1. Or create a graph space after storage hosts are scaled out.
- A new storage host is found, but ADD HOSTS is not executed to activate it. In this case, run SHOW HOSTS to locate the new storage host information and then run ADD HOSTS to activate it. A graph space can be created after there are enough storage hosts.
- For offline storage hosts after running SHOW HOSTS, troubleshooting is needed.

# L jacy version compatibility

For NebulaGraph v2.x before v2.5.0, vid\_type is optional and defaults to FIXED\_STRING(8).

#### Q Note

graph\_space\_name, partition\_num, replica\_factor, vid\_type, and comment cannot be modified once set. To modify them, drop the current working graph space with DROP SPACE and create a new one with CREATE SPACE.

#### CLONE GRAPH SPACES

CREATE SPACE [IF NOT EXISTS] <new\_graph\_space\_name> AS <old\_graph\_space\_name>;

Parameter	Description
IF NOT EXISTS	Detects if the new graph space exists. If it does not exist, the new one will be created. The graph space existence detection here only compares the graph space name (excluding properties).
<new_graph_space_name></new_graph_space_name>	The name of the graph space that is newly created. By default, the space name only supports 1-4 byte UTF-8 encoded characters, including English letters (case sensitive), numbers, Chinese characters, etc. But special characters can only use underscore, and cannot start with a number. To use special characters, reserved keywords, or start with a number, quote the entire name with backticks (`) and cannot use periods ( . ). For more information, see Keywords and reserved words. When a new graph space is created, the schema of the old graph space <ol> <li>old_graph_space_name&gt; will be</li> <li>cloned, including its parameters (the number of partitions and replicas, etc.), Tag, Edge type and native indexes. Note: <ol> <li>If you name a space in Chinese and encounter a SyntaxError, you need to quote the Chinese</li> <li>characters with backticks (`).</li> <li>To include a backtick (`) in a space name, use a backslash to escape the backtick, such as \`; to</li> </ol></li></ol>
<old_graph_space_name></old_graph_space_name>	The name of the graph space that already exists.

## Examples

# The following example creates a graph space with a specified VID type and the maximum length. Other fields still use the default values. nebula> CREATE SPACE IF NOT EXISTS my_space_1 (vid_type=FIXED_STRING(30));
# The following example creates a graph space with a specified partition number, replica number, and VID type. nebula> CREATE SPACE IF NOT EXISTS my_space_2 (partition_num=15, replica_factor=1, vid_type=FIXED_STRING(30));
# The following example creates a graph space with a specified partition number, replica number, and VID type, and adds a comment on it. nebula> CREATE SPACE IF NOT EXISTS my_space_3 (partition_num=15, replica_factor=1, vid_type=FIXED_STRING(30)) comment="Test the graph space";
<pre># Clone a graph space. nebula&gt; CREATE SPACE IF NOT EXISTS my_space_4 as my_space_3; nebula&gt; SHOW CREATE SPACE my_space_4;</pre>
Space   Create Space
"my_space_4"   "CREATE SPACE `my_space_4` (partition_num = 15, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type = FIXED_STRING(30)) comment = '测试图空间'"   ++

## Implementation of the operation

Caution	
Trying to use a newly created graph space may fail because the creation is implemented asynchronously. To make sure the follow-up	
operations work as expected, Wait for two heartbeat cycles, i.e., 20 seconds. To change the heartbeat interval, modify the	
heartbeat_interval_secs parameter in the configuration files for all services. If the heartbeat interval is too short (i.e., less than 5	
seconds) disconnection between neers may happen because of the misjudgment of machines in the distributed system	

## Check partition distribution

On some large clusters, the partition distribution is possibly unbalanced because of the different startup times. You can run the following command to do a check of the machine distribution.

nebula> SHOW HOSTS;

+	+	+		+	-++
Host	Port   Status	Leader count	Leader distribution	Partition distribution	Version
+	++	++		+	-++
"storaged0"	9779   "ONLINE"	8	"basketballplayer:3, test:5"	"basketballplayer:10, test:10"	"3.6.0"
storaged1"	9779   "ONLINE"	9	"basketballplayer:4, test:5"	<pre>"basketballplayer:10, test:10"</pre>	3.6.0"
storaged2"	9779   "ONLINE"	3	"basketballplayer:3"	"basketballplayer:10, test:10"	"3.6.0"
+	+	+	+	+	-++

## To balance the request loads, use the following command.

nebula> BALANCE LEADER; nebula> SHOW HOSTS; +	+++	+	
Host   Port   HTTP port   Status	Leader count   Leader distribution	Partition distribution   Ve	ersion
"storaged0"   9779   "ONLINE"   7   "storaged1"   9779   "ONLINE"   7   "storaged2"   9779   "ONLINE"   6 +	"basketballplayer:3, test:4"   "basket   "basketballplayer:4, test:3"   "basket   "basketballplayer:3, test:3"   "basket	ballplayer:10, test:10"   "3.6.0"   ballplayer:10, test:10"   "3.6.0"   ballplayer:10, test:10"   "3.6.0"	

Last update: December 28, 2023

## 4.8.2 USE

USE specifies a graph space as the current working graph space for subsequent queries.

#### Prerequisites

Running the USE statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

USE <graph\_space\_name>;

## Examples

# The following example creates two sample spaces. nebula> CREATE SPACE IF NOT EXISTS space1 (vid\_type=FIXED\_STRING(30)); nebula> CREATE SPACE IF NOT EXISTS space2 (vid\_type=FIXED\_STRING(30));

# The following example specifies space1 as the current working graph space. nebula> USE space1;

# The following example specifies space2 as the current working graph space. Hereafter, you cannot read any data from space1, because these vertices and edges being traversed have no relevance with space1. nebula> USE space2;

## Caution

You cannot use two graph spaces in one statement.

Different from Fabric Cypher, graph spaces in NebulaGraph are fully isolated from each other. Making a graph space as the working graph space prevents you from accessing other spaces. The only way to traverse in a new graph space is to switch by the USE statement. In Fabric Cypher, you can use two graph spaces in one statement (using the USE + CALL syntax). But in NebulaGraph, you can only use one graph space in one statement.

Last update: November 3, 2023

## 4.8.3 SHOW SPACES

SHOW SPACES lists all the graph spaces in the NebulaGraph examples.

## Syntax

SHOW SPACES;

## Example

nebula> SHOW SPACES; +-----+ | Name | +-----+ | "cba" | "basketballplayer" | +-----+

To create graph spaces, see CREATE SPACE.

## 4.8.4 DESCRIBE SPACE

 $\ensuremath{\texttt{DESCRIBE}}$  space returns the information about the specified graph space.

## Syntax

You can use DESC instead of DESCRIBE for short.

DESC[RIBE] SPACE <graph\_space\_name>;

The DESCRIBE SPACE statement is different from the SHOW SPACES statement. For details about SHOW SPACES, see SHOW SPACES.

## Example

nebula> DESCRIBE SPACE ba	sketballplayer;					
++	-+	-+	+	+	+	++
ID   Name	Partition Number	Replica Factor	Charset	Collate	Vid Type	Comment
1   "basketballplayer"	10	1	"utf8"	"utf8_bin"	"FIXED_STRING(32)"	

## 4.8.5 CLEAR SPACE

CLEAR SPACE deletes the vertices and edges in a graph space, but does not delete the graph space itself and the schema information.

## Note

It is recommended to execute SUBMIT JOB COMPACT immediately after executing the CLEAR SPACE operation improve the query performance. Note that the COMPACT operation may affect query performance, and it is recommended to perform this operation during low business hours (e.g., early morning).

## Permission requirements

Only the God role has the permission to run CLEAR SPACE.

#### Caution

- Once cleared, the data **CANNOT be recovered**. Use CLEAR SPACE with caution.
- CLEAR SPACE is not an atomic operation. If an error occurs, re-run CLEAR SPACE to avoid data remaining.
- The larger the amount of data in the graph space, the longer it takes to clear it. If the execution fails due to client connection timeout, increase the value of the storage\_client\_timeout\_ms parameter in the Graph Service configuration.
- During the execution of CLEAR SPACE, writing data into the graph space is not automatically prohibited. Such write operations can result in incomplete data clearing, and the residual data can be damaged.

( )
MAL-
Note

The NebulaGraph Community Edition does not support blocking data writing while allowing CLEAR SPACE .

#### Syntax

CLEAR SPACE [IF EXISTS] <space\_name>;

Parameter/ Option	Description
IF EXISTS	Check whether the graph space to be cleared exists. If it exists, continue to clear it. If it does not exist, the execution finishes, and a message indicating that the execution succeeded is displayed. If IF EXISTS is not set and the graph space does not exist, the CLEAR SPACE statement fails to execute, and an error occurs.
space_name	The name of the space to be cleared.

#### Example:

CLEAR SPACE basketballplayer;

#### Data reserved

CLEAR SPACE does not delete the following data in a graph space:

- Tag information.
- Edge type information.
- The metadata of native indexes and full-text indexes.

The following example shows what CLEAR SPACE deletes and reserves.

# Enter the graph space basketballplayer. nebula [(none)]> use basketballplayer; Execution succeeded # List tags and Edge types. nebula[basketballplayer]> SHOW TAGS; | Name | "player" "team" Got 2 rows nebula[basketballplayer]> SHOW EDGES; | Name | "follow" | "serve" Got 2 rows # Submit a job to make statistics of the graph space. nebula[basketballplayer]> SUBMIT JOB STATS; | New Job Id | | 4 Got 1 rows # Check the statistics. nebula[basketballplayer]> SHOW STATS; Count | Type Name "Tag" "player" | 51 "Tag" "team" | 30 "Edge" "Edge" "Space" "follow" 81 "serve" 152 "vertices" | 81 "Space" "edges" 233 Got 6 rows # List tag indexes. nebula[basketballplayer]> SHOW TAG INDEXES; ----+ Index Name | By Tag | Columns "player\_index\_0" | "player" | [] "player\_index\_1" | "player" | ["name"] ----+ Got 2 rows # ----- Dividing line for CLEAR SPACE -----# Run CLEAR SPACE to clear the graph space basketballplayer. nebula[basketballplayer]> CLEAR SPACE basketballplayer; Execution succeeded # Update the statistics. nebula[basketballplayer]> SUBMIT JOB STATS; | New Job Id | | 5 Got 1 rows # Check the statistics. The tags and edge types still exist, but all the vertices and edges are gone. nebula[basketballplayer]> SHOW STATS; -+-| Type | Name | Count "Tag" "Tag" | "player" | "team" | 0 0 "Edge" "Edge" "Space" "follow" 0 "serve" 0 "vertices" 0 "Space" "edges" 0 Got 6 rows

# Try to list the tag indexes. They still exist. nebula[basketballplayer]> SHOW TAG INDEXES;

Index Name	By Tag	Columns
++	+	+
"player_index_0"	"player"	[] [
"player index 1"	"player"	["name"]

+-----+ Got 2 rows (time spent 523/978 us)

Last update: November 15, 2023

### 4.8.6 DROP SPACE

DROP SPACE deletes the specified graph space and everything in it.

## Note

DROP SPACE can only delete the specified logic graph space while retain all the data on the hard disk by modifying the value of auto\_remove\_invalid\_space to false in the Storage service configuration file. For more information, see Storage configuration.

#### Arning

After you execute DROP SPACE, even if the snapshot contains data of the graph space, the data of the graph space cannot be recovered.

#### Prerequisites

Only the God role can use the DROP SPACE statement. For more information, see AUTHENTICATION.

#### Syntax

DROP SPACE [IF EXISTS] <graph\_space\_name>;

You can use the IF EXISTS keywords when dropping spaces. These keywords automatically detect if the related graph space exists. If it exists, it will be deleted. Otherwise, no graph space will be deleted.

## L Jacy version compatibility

In NebulaGraph versions earlier than 3.1.0, the DROP SPACE statement does not remove all the files and directories from the disk by default.

## **D**anger

BE CAUTIOUS about running the DROP SPACE statement.

### FAQ

Q: Why is my disk space not freed after executing the 'DROP SPACE' statement and deleting a graph space?

A: For NebulaGraph version earlier than 3.1.0, DROP SPACE can only delete the specified logic graph space and does not delete the files and directories on the disk. To delete the files and directories on the disk, manually delete the corresponding file path. The file path is located in <nebula\_graph\_install\_path>/data/storage/nebula/<space\_id>. The <space\_id> can be viewed via DESCRIBE SPACE {space\_name}.

Last update: December 14, 2023

## 4.9 Tag statements

## 4.9.1 CREATE TAG

 $\ensuremath{\mathsf{CREATE}}$  TAG creates a tag with the given name in a graph space.

#### OpenCypher compatibility

Tags in nGQL are similar to labels in openCypher. But they are also quite different. For example, the ways to create them are different.

- In openCypher, labels are created together with vertices in CREATE statements.
- In nGQL, tags are created separately using CREATE TAG statements. Tags in nGQL are more like tables in MySQL.

### Prerequisites

Running the CREATE TAG statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

### Syntax

To create a tag in a specific graph space, you must specify the current working space with the USE statement.

CREATE TAG [IF NOT EXISTS] <tag\_name>

<prop\_name> <data\_type> [NULL | NOT NULL] [DEFAULT <default\_value>] [COMMENT '<comment>'] [{, <prop\_name> <data\_type> [NULL | NOT NULL] [DEFAULT <default\_value>] [COMMENT '<comment>']} ...]

[TTL\_DURATION = <ttl\_duration>]

```
[TTL_COL = <prop_name>]
[COMMENT = '<comment>'];
```

Parameter	Description
IF NOT EXISTS	Detects if the tag that you want to create exists. If it does not exist, a new one will be created. The tag existence detection here only compares the tag names (excluding properties).
<tag_name></tag_name>	<ol> <li>Each tag name in the graph space must be unique.</li> <li>Tag names cannot be modified after they are set.</li> <li>By default, the name only supports 1-4 byte UTF-8 encoded characters, including English letters (case sensitive), numbers, Chinese characters, etc. However, it cannot include special characters other than the underscore (_), and cannot start with a number.</li> <li>To use special characters, reserved keywords, or start with a number, quote the entire name with backticks (`) and do not include periods ( . ) within the pair of backticks (`). For more information, see Keywords and reserved words.</li> <li>Note:         <ol> <li>If you name a tag in Chinese and encounter a Syntaxtrror, you need to quote the Chinese characters with backticks (`).</li> <li>To include a backtick (`) in a tag name, use a backslash to escape the backtick, such as \`; to include a backslash itself also needs to be escaped, such as \ .</li> </ol> </li> </ol>
<prop_name></prop_name>	The name of the property. It must be unique for each tag. The rules for permitted property names are the same as those for tag names.
<data_type></data_type>	Shows the data type of each property. For a full description of the property data types, see Data types and Boolean.
NULL   NOT	Specifies if the property supports $$ NULL $ $ NOT NULL . The default value is $$ NULL .
DEFAULT	Specifies a default value for a property. The default value can be a literal value or an expression supported by NebulaGraph. If no value is specified, the default value is used when inserting a new vertex.
COMMENT	The remarks of a certain property or the tag itself. The maximum length is 256 bytes. By default, there will be no comments on a tag.
TTL_DURATION	Specifies the life cycle for the property. The property that exceeds the specified TTL expires. The expiration threshold is the TTL_COL value plus the TTL_DURATION. The default value of TTL_DURATION is 0. It means the data never expires.
TTL_COL	Specifies the property to set a timeout on. The data type of the property must be int or timestamp. A tag can only specify one field as $TTL_COL$ . For more information on TTL, see $TTL$ options.

EXAMPLES

```
nebula> CREATE TAG IF NOT EXISTS player(name string, age int);
# The following example creates a tag with no properties.
nebula> CREATE TAG IF NOT EXISTS no_property();
# The following example creates a tag with a default value.
nebula> CREATE TAG IF NOT EXISTS player_with_default(name string, age int DEFAULT 20);
# In the following example, the TTL of the create_time field is set to be 100 seconds.
nebula> CREATE TAG IF NOT EXISTS monon(name string, age int, \
    married bool, salary double, create_time timestamp) \
    TTL_DURATION = 100, TTL_COL = "create_time";
```

#### Implementation of the operation

Trying to use a newly created tag may fail because the creation of the tag is implemented asynchronously. To make sure the follow-up operations work as expected, Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the heartbeat\_interval\_secs parameter in the configuration files for all services.

Last update: December 28, 2023

### 4.9.2 DROP TAG

DROP TAG drops a tag with the given name in the current working graph space.

A vertex can have one or more tags.

- If a vertex has only one tag, the vertex **CANNOT** be accessed after you drop it. The vertex will be dropped in the next compaction. But its edges are available, this operation will result in dangling edges.
- If a vertex has multiple tags, the vertex is still accessible after you drop one of them. But all the properties defined by this dropped tag **CANNOT** be accessed.

This operation only deletes the Schema data. All the files or directories in the disk will not be deleted directly until the next compaction.

# **P**\_mpatibility

In NebulaGraph 3.6.0, inserting vertex without tag is not supported by default. If you want to use the vertex without tags, add --graph\_use\_vertex\_key=true to the configuration files ( nebula-graphd.conf ) of all Graph services in the cluster, and add --use\_vertex\_key=true to the configuration files ( nebula-storaged.conf ) of all Storage services in the cluster.

#### Prerequisites

- Running the DROP TAG statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.
- Before you drop a tag, make sure that the tag does not have any indexes. Otherwise, the conflict error ( [ERROR (-1005)]: Conflict! ) will be returned when you run the DROP TAG statement. To drop an index, see DROP INDEX.

#### Syntax

DROP TAG [IF EXISTS] <tag\_name>;

- IF NOT EXISTS : Detects if the tag that you want to drop exists. Only when it exists will it be dropped.
- tag\_name : Specifies the tag name that you want to drop. You can drop only one tag in one statement.

#### Example

nebula> CREATE TAG IF NOT EXISTS test(p1 string, p2 int); nebula> DROP TAG test;

Last update: November 3, 2023

## 4.9.3 ALTER TAG

ALTER TAG alters the structure of a tag with the given name in a graph space. You can add or drop properties, and change the data type of an existing property. You can also set a TTL (Time-To-Live) on a property, or change its TTL duration.

#### Notes

- Running the ALTER TAG statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.
- Before you alter properties for a tag, make sure that the properties are not indexed. If the properties contain any indexes, the conflict error [ERROR (-1005)]: Conflict! will occur when you ALTER TAG. For more information on dropping an index, see DROP INDEX.
- The property name must be unique in a tag. If you add a property with the same name as an existing property or a dropped property, the operation fails.

### Syntax

```
ALTER TAG <tag_name>

<alter_definition> [[, alter_definition] ...]

[ttl_definition [, ttl_definition] ...]

[COMMENT '<comment>'];

alter_definition:

| ADD (prop_name data_type [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'])

| DROP (prop_name)

| CHANGE (prop_name data_type [NULL | NOT NULL] [DEFAULT <default_value>] [COMMENT '<comment>'])

ttl_definition:

TTL_DURATION = ttl_duration, TTL_COL = prop_name
```

- tag\_name: Specifies the tag name that you want to alter. You can alter only one tag in one statement. Before you alter a tag, make sure that the tag exists in the current working graph space. If the tag does not exist, an error will occur when you alter it.
- Multiple ADD, DROP, and CHANGE clauses are permitted in a single ALTER TAG statement, separated by commas.
- When a property value is set to NOT NULL using ADD or CHANGE, a default value must be specified for the property, that is, the value of DEFAULT must be specified.
- When using CHANGE to modify the data type of a property:
- Only the length of a FIXED\_STRING or an INT can be increased. The length of a STRING or an INT cannot be decreased.
- Only the data type conversions from FIXED\_STRING to STRING and from FLOAT to DOUBLE are allowed.

#### Examples

```
nebula> CREATE TAG IF NOT EXISTS t1 (p1 string, p2 int);
nebula> ALTER TAG t1 ADD (p3 int32, fixed_string(10));
nebula> ALTER TAG t1 TTL_DURATION = 2, TTL_COL = "p2";
nebula> ALTER TAG t1 COMMENT = 'test1';
nebula> ALTER TAG t1 ADD (p5 double NOT NULL DEFAULT 0.4 COMMENT 'p5') COMMENT='test2';
// Change the data type of p3 in the TAG t1 from INT32 to INT64, and that of p4 from FIXED_STRING(10) to STRING.
nebula> ALTER TAG t1 CHANGE (p3 int64, p4 string);
[ERROR(-1005)]: Unsupported!
```

#### Implementation of the operation

Trying to use a newly altered tag may fail because the alteration of the tag is implemented asynchronously. To make sure the follow-up operations work as expected, Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the heartbeat\_interval\_secs parameter in the configuration files for all services.

## 4.9.4 SHOW TAGS

The  $\ensuremath{\operatorname{SHOW}}$  TAGS statement shows the name of all tags in the current graph space.

You do not need any privileges for the graph space to run the SHOW TAGS statement. But the returned results are different based on role privileges.

#### Syntax

SHOW TAGS;

## Examples

nebula>	SHOW	TAGS;
+	+	
Name		
+	+	
"playe	r"	
"team"		
+	+	

## 4.9.5 DESCRIBE TAG

DESCRIBE TAG returns the information about a tag with the given name in a graph space, such as field names, data type, and so on.

## Prerequisite

Running the DESCRIBE TAG statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

DESC[RIBE] TAG <tag\_name>;

You can use DESC instead of DESCRIBE for short.

#### Example

nebula> DESCRIBE TAG player;	
++	
Field   Type   Null   Default   Comment	
++	
"name"   "string"   "YES"	
"age"   "int64"   "YES"	
++	

## 4.9.6 DELETE TAG

DELETE TAG deletes a tag with the given name on a specified vertex.

#### Prerequisites

Running the DELETE TAG statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

DELETE TAG <tag\_name\_list> FROM <VID\_list>;

- tag\_name\_list : The names of the tags you want to delete. Multiple tags are separated with commas (,). \* means all tags.
- VID: The VIDs of the vertices from which you want to delete the tags. Multiple VIDs are separated with commas (,).

#### Example

# **P**mpatibility

• In openCypher, you can use the statement REMOVE v:LABEL to delete the tag LABEL of the vertex v .

• DELETE TAG and DROP TAG have the same semantics but different syntax. In nGQL, use DELETE TAG.

Last update: November 3, 2023

#### 4.9.7 Add and delete tags

OpenCypher has the features of SET label and REMOVE label to speed up the process of querying or labeling.

NebulaGraph achieves the same operations by creating and inserting tags to an existing vertex, which can quickly query vertices based on the tag name. Users can also run DELETE TAG to delete some vertices that are no longer needed.

#### Examples

For example, in the basketballplayer data set, some basketball players are also team shareholders. Users can create an index for the shareholder tag shareholder for quick search. If the player is no longer a shareholder, users can delete the shareholder tag of the corresponding player by DELETE TAG.

```
//This example creates the shareholder tag and index.
nebula> CREATE TAG IF NOT EXISTS shareholder();
nebula> CREATE TAG INDEX IF NOT EXISTS shareholder_tag on shareholder();
//This example adds a tag on the vertex.
nebula> INSERT VERTEX shareholder() VALUES "player100":();
nebula> INSERT VERTEX shareholder() VALUES "player101":();
//This example queries all the shareholders.
nebula> MATCH (v:shareholder) RETURN v;
+----
V V
| ("player100" :player{age: 42, name: "Tim Duncan"} :shareholder{})
  ("player101" :player{age: 36, name: "Tony Parker"} :shareholder{})
nebula> LOOKUP ON shareholder YIELD id(vertex);
| id(VERTEX)
  "player100"
  "player101"
//In this example, the "player100" is no longer a shareholder.
nebula> DELETE TAG shareholder FROM "player100";
nebula> LOOKUP ON shareholder YIELD id(vertex);
| id(VERTEX)
| "player101"
```

## Note

If the index is created after inserting the test data, use the REBUILD TAG INDEX <index\_name\_list>; statement to rebuild the index.

## 4.10 Edge type statements

#### 4.10.1 CREATE EDGE

CREATE EDGE creates an edge type with the given name in a graph space.

#### OpenCypher compatibility

Edge types in nGQL are similar to relationship types in openCypher. But they are also quite different. For example, the ways to create them are different.

- In openCypher, relationship types are created together with vertices in CREATE statements.
- In nGQL, edge types are created separately using CREATE EDGE statements. Edge types in nGQL are more like tables in MySQL.

## Prerequisites

Running the CREATE EDGE statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

To create an edge type in a specific graph space, you must specify the current working space with the USE statement.

CREATE EDGE [IF NOT EXISTS] <edge\_type\_name>

<full | NOT NULL] [DEFAULT <default\_value>] [COMMENT '<comment>']
[{, <prop\_name> <data\_type> [NULL | NOT NULL] [DEFAULT <default\_value>] [COMMENT '<comment>']} ...]

[TTL\_DURATION = <ttl\_duration>]

## [TTL\_COL = <prop\_name>] [COMMENT = '<comment>'];

Parameter	Description
IF NOT EXISTS	Detects if the edge type that you want to create exists. If it does not exist, a new one will be created. The edge type existence detection here only compares the edge type names (excluding properties).
<edge_type_name></edge_type_name>	<ol> <li>The edge type name must be unique in a graph space.</li> <li>Once the edge type name is set, it can not be altered.</li> <li>By default, the name only supports 1-4 byte UTF-8 encoded characters, including English letters (case sensitive), numbers, Chinese characters, etc. However, it cannot include special characters other than the underscore (_), and cannot start with a number.</li> <li>To use special characters, reserved keywords, or start with a number, quote the entire name with backticks (`) and do not include periods ( .) within the pair of backticks (`). For more information, see Keywords and reserved words.</li> <li>Note:         <ol> <li>If you name an edge type in Chinese and encounter a SyntaxError, you need to quote the Chinese characters with backticks (`) in an edge type name, use a backslash to escape the backtick, such as \`; to include a backslash, the backslash itself also needs to be escaped, such as \.</li> </ol> </li> </ol>
<prop_name></prop_name>	The name of the property. It must be unique for each edge type. The rules for permitted property names are the same as those for edge type names.
<data_type></data_type>	Shows the data type of each property. For a full description of the property data types, see Data types and Boolean.
NULL   NOT NULL	Specifies if the property supports NULL $\mid$ NOT NULL . The default value is NULL . DEFAULT must be specified if NOT NULL is set.
DEFAULT	Specifies a default value for a property. The default value can be a literal value or an expression supported by NebulaGraph. If no value is specified, the default value is used when inserting a new edge.
COMMENT	The remarks of a certain property or the edge type itself. The maximum length is 256 bytes. By default, there will be no comments on an edge type.
TTL_DURATION	Specifies the life cycle for the property. The property that exceeds the specified TTL expires. The expiration threshold is the TTL_COL value plus the TTL_DURATION. The default value of TTL_DURATION is 0. It means the data never expires.
TTL_COL	Specifies the property to set a timeout on. The data type of the property must be int or timestamp. An edge type can only specify one field as ITL_COL. For more information on TTL, see TTL options.

EXAMPLES

nebula> CREATE EDGE IF NOT EXISTS follow(degree int);

# The following example creates an edge type with no properties. nebula> CREATE EDGE IF NOT EXISTS no\_property();

# The following example creates an edge type with a default value. nebula> CREATE EDGE IF NOT EXISTS follow\_with\_default(degree int DEFAULT 20);

# In the following example, the TTL of the p2 field is set to be 100 seconds. nebula> CREATE EDGE IF NOT EXISTS el(p1 string, p2 int, p3 timestamp) \ TTL\_DURATION = 100, TTL\_COL = "p2";

Last update: December 28, 2023

## 4.10.2 DROP EDGE

DROP EDGE drops an edge type with the given name in a graph space.

An edge can have only one edge type. After you drop it, the edge **CANNOT** be accessed. The edge will be deleted in the next compaction.

This operation only deletes the Schema data. All the files or directories in the disk will not be deleted directly until the next compaction.

#### Prerequisites

- Running the DROP EDGE statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.
- Before you drop an edge type, make sure that the edge type does not have any indexes. Otherwise, the conflict error ( [ERROR (-1005)]: Conflict! ) will be returned. To drop an index, see DROP INDEX.

#### Syntax

DROP EDGE [IF EXISTS] <edge\_type\_name>

- IF NOT EXISTS : Detects if the edge type that you want to drop exists. Only when it exists will it be dropped.
- edge\_type\_name : Specifies the edge type name that you want to drop. You can drop only one edge type in one statement.

#### Example

nebula> CREATE EDGE IF NOT EXISTS el(p1 string, p2 int); nebula> DROP EDGE el;
## 4.10.3 ALTER EDGE

ALTER EDGE alters the structure of an edge type with the given name in a graph space. You can add or drop properties, and change the data type of an existing property. You can also set a TTL (Time-To-Live) on a property, or change its TTL duration.

#### Notes

- Running the ALTER EDGE statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.
- Before you alter properties for an edge type, make sure that the properties are not indexed. If the properties contain any indexes, the conflict error [ERROR (-1005)]: Conflict! will occur when you ALTER EDGE. For more information on dropping an index, see DROP INDEX.
- The property name must be unique in an edge type. If you add a property with the same name as an existing property or a dropped property, the operation fails.
- Only the length of a FIXED\_STRING or an INT can be increased.
- Only the data type conversions from FIXED\_STRING to STRING and from FLOAT to DOUBLE are allowed.

#### Syntax

- edge\_type\_name : Specifies the edge type name that you want to alter. You can alter only one edge type in one statement. Before
  you alter an edge type, make sure that the edge type exists in the graph space. If the edge type does not exist, an error occurs
  when you alter it.
- Multiple ADD, DROP, and CHANGE clauses are permitted in a single ALTER EDGE statement, separated by commas.
- When a property value is set to NOT NULL using ADD or CHANGE, a default value must be specified for the property, that is, the value of DEFAULT must be specified.

#### Example

```
nebula> CREATE EDGE IF NOT EXISTS e1(p1 string, p2 int);
nebula> ALTER EDGE e1 ADD (p3 int, p4 string);
nebula> ALTER EDGE e1 TTL_DURATION = 2, TTL_COL = "p2";
nebula> ALTER EDGE e1 COMMENT = 'edge1';
```

#### Implementation of the operation

Trying to use a newly altered edge type may fail because the alteration of the edge type is implemented asynchronously. To make sure the follow-up operations work as expected, Wait for two heartbeat cycles, i.e., 20 seconds.

To change the heartbeat interval, modify the heartbeat\_interval\_secs parameter in the configuration files for all services.

```
Last update: October 25, 2023
```

## 4.10.4 SHOW EDGES

 $\ensuremath{\texttt{SHOW}}\xspace$  shows all edge types in the current graph space.

You do not need any privileges for the graph space to run the SHOW EDGES statement. But the returned results are different based on role privileges.

#### Syntax

SHOW EDGES;

## Example

nebula> SHOW EDGES;	
++	
Name	
++	
"follow"	
serve"	
++	

## 4.10.5 DESCRIBE EDGE

DESCRIBE EDGE returns the information about an edge type with the given name in a graph space, such as field names, data type, and so on.

#### Prerequisites

Running the DESCRIBE EDGE statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

DESC[RIBE] EDGE <edge\_type\_name>

You can use DESC instead of DESCRIBE for short.

## Example

nebula> DESCRIBE EDGE follow;
++
Field   Type   Null   Default   Comment
"degree"   "int64"   "YES"

## 4.11 Vertex statements

## 4.11.1 INSERT VERTEX

The INSERT VERTEX statement inserts one or more vertices into a graph space in NebulaGraph.

#### Prerequisites

Running the INSERT VERTEX statement requires some privileges for the graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

```
INSERT VERTEX [IF NOT EXISTS] [tag_props, [tag_props] ...]
VALUES VID: ([prop_value_list])
tag_props:
   tag_name ([prop_name_list])
prop_name_list:
   [prop_name [, prop_name] ...]
prop_value_list:
   [prop_value [, prop_value] ...]
```

• IF NOT EXISTS detects if the VID that you want to insert exists. If it does not exist, a new one will be inserted.



- IF NOT EXISTS only compares the names of the VID and the tag (excluding properties).
- IF NOT EXISTS will read to check whether the data exists, which will have a significant impact on performance.
- tag\_name denotes the tag (vertex type), which must be created before INSERT VERTEX. For more information, see CREATE TAG.

## Caution

NebulaGraph 3.6.0 supports inserting vertices without tags.

# **U**mpatibility

- prop\_name\_list contains the names of the properties on the tag.
- **VID** is the vertex ID. In NebulaGraph 2.0, string and integer VID types are supported. The VID type is set when a graph space is created. For more information, see CREATE SPACE.
- prop\_value\_list must provide the property values according to the prop\_name\_list. When the NOT NULL constraint is set for a given property, an error is returned if no property is given. When the default value for a property is NULL, you can omit to specify the property value. For details, see CREATE TAG.

## Caution

INSERT VERTEX and CREATE have different semantics.

- The semantics of INSERT VERTEX is closer to that of INSERT in NoSQL (key-value), or UPSERT ( UPDATE or INSERT ) in SQL.
- When two INSERT statements (neither uses IF NOT EXISTS) with the same VID and TAG are operated at the same time, the latter INSERT will overwrite the former.
- When two INSERT statements with the same VID but different TAGS are operated at the same time, the operation of different tags will not overwrite each other.

#### Examples are as follows.

#### Examples

```
# Insert a vertex without tag.
nebula> INSERT VERTEX VALUES "1":();
# The following examples create tag tl with no property and inserts vertex "10" with no property.
nebula> CREATE TAG IF NOT EXISTS t1();
nebula> INSERT VERTEX t1() VALUES "10":();
nebula> CREATE TAG IF NOT EXISTS t2 (name string, age int);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n1", 12);
```

# In the following example, the insertion fails because "a13" is not int. nebula> INSERT VERTEX t2 (name, age) VALUES "12":("n1", "a13");

# The following example inserts two vertices at one time. nebula> INSERT VERTEX t2 (name, age) VALUES "13":("n3", 12), "14":("n4", 8);

nebula> CREATE TAG IF NOT EXISTS t3(p1 int); nebula> CREATE TAG IF NOT EXISTS t4(p2 string);

# The following example inserts vertex "21" with two tags. nebula> INSERT VERTEX t3 (p1), t4(p2) VALUES "21": (321, "hello");

A vertex can be inserted/written with new values multiple times. Only the last written values can be read.

<pre># The following examples insert vertex "11" with new values for multiple times. nebula&gt; INSERT VERTEX t2 (name, age) VALUES "11":("n2", 13); nebula&gt; INSERT VERTEX t2 (name, age) VALUES "11":("n3", 14); nebula&gt; INSERT VERTEX t2 (name, age) VALUES "11":("n4", 15); nebula&gt; FETCH PROP ON t2 "11" YIELD properties(vertex); +</pre>
<pre>nebula&gt; CREATE TAG IF NOT EXISTS t5(p1 fixed_string(5) NOT NULL, p2 int, p3 int DEFAULT NULL); nebula&gt; INSERT VERTEX t5(p1, p2, p3) VALUES "001":("Abe", 2, 3);</pre>
# In the following example, the insertion fails because the value of p1 cannot be NULL. nebula> INSERT VERTEX t5(p1, p2, p3) VALUES "002":(NULL, 4, 5); [ERROR (-1009)]: SemanticError: No schema found for `t5'
<pre># In the following example, the value of p3 is the default NULL. nebula&gt; INSERT VERTEX t5(p1, p2) VALUES "003":("cd", 5); nebula&gt; FETCH PROP ON t5 "003" YIELD properties(vertex);</pre>
++
properties(VERTEX)
++
{p1: "cd", p2: 5, p3:NULL}
++
<pre># In the following example, the allowed maximum length of p1 is 5. nebula&gt; INSERT VERTEX t5(p1, p2) VALUES "004":("shalalalal", 4); nebula&gt; FETCH PROP on t5 "004" YIELD properties(vertex); +</pre>
++

If you insert a vertex that already exists with  $\mbox{ IF NOT EXISTS}$  , there will be no modification.

<sup>+----+</sup> | {p1: "shala", p2: 4, p3: \_\_NULL\_\_} | +-----+

# The following example inserts vertex "1".
nebula> INSERT VERTEX t2 (name, age) VALUES "1":("n2", 13);
# Modify vertex "1" with IF NOT EXISTS. But there will be no modification as vertex "1" already exists.
nebula> INSERT VERTEX IF NOT EXISTS t2 (name, age) VALUES "1":("n3", 14);
nebula> FETCH PROP ON t2 "1" YIELD properties(vertex);
+------++
| properties(VERTEX) |
+------++

| {age: 13, name: "n2"} |

Last update: November 3, 2023

#### 4.11.2 DELETE VERTEX

By default, the DELETE VERTEX statement deletes vertices but the incoming and outgoing edges of the vertices.

# **P**\_mpatibility

• NebulaGraph 2.x deletes vertices and their incoming and outgoing edges.

• NebulaGraph 3.6.0 only deletes the vertices, and does not delete the related outgoing and incoming edges of the vertices. At this time, there will be dangling edges by default.

The DELETE VERTEX statement deletes one vertex or multiple vertices at a time. You can use DELETE VERTEX together with pipes. For more information about pipe, see Pipe operator.

• DELETE VERTEX deletes vertices directly.

DELETE TAG deletes a tag with the given name on a specified vertex.

#### Syntax

DELETE VERTEX <vid> [, <vid> ...] [WITH EDGE];

• WITH EDGE: deletes vertices and the related incoming and outgoing edges of the vertices.

#### Examples

This query deletes the vertex whose ID is "team1".

# Delete the vertex whose VID is `team1` but the related incoming and outgoing edges are not deleted. nebula> DELETE VERTEX "team1"; # Delete the vertex whose VID is `team1` and the related incoming and outgoing edges.

nebula> DELETE VERTEX "team1" WITH EDGE;

This query shows that you can use DELETE VERTEX together with pipe to delete vertices.

nebula> 60 FROM "player100" OVER serve WHERE properties(edge).start\_year == "2021" YIELD dst(edge) AS id | DELETE VERTEX \$-.id;

#### Process of deleting vertices

Once NebulaGraph deletes the vertices, all edges (incoming and outgoing edges) of the target vertex will become dangling edges. When NebulaGraph deletes the vertices WITH EDGE, NebulaGraph traverses the incoming and outgoing edges related to the vertices and deletes them all. Then NebulaGraph deletes the vertices.

## Caution

• Atomic deletion is not supported during the entire process for now. Please retry when a failure occurs to avoid partial deletion, which will cause pendent edges.

• Deleting a supernode takes a lot of time. To avoid connection timeout before the deletion is complete, you can modify the parameter --storage\_client\_timeout\_ms in nebula-graphd.conf to extend the timeout period.

```
Last update: October 25, 2023
```

## 4.11.3 UPDATE VERTEX

The UPDATE VERTEX statement updates properties on tags of a vertex.

In NebulaGraph, UPDATE VERTEX supports compare-and-set (CAS).

## Note

An UPDATE VERTEX statement can only update properties on  $\mathbf{ONE}\ \mathbf{TAG}$  of a vertex.

#### Syntax

```
UPDATE VERTEX ON <tag_name> <vid>
SET <update_prop>
[WHEN <condition>]
[YIELD <output>]
```

Parameter	Required	Description	Example
ON <tag_name></tag_name>	Yes	Specifies the tag of the vertex. The properties to be updated must be on this tag.	ON player
<vid></vid>	Yes	Specifies the ID of the vertex to be updated.	"player100"
SET <update_prop></update_prop>	Yes	Specifies the properties to be updated and how they will be updated.	SET age = age +1
WHEN <condition></condition>	No	Specifies the filter conditions. If $<\!\!\text{condition}\!\!>$ evaluates to $false$ , the SET clause will not take effect.	WHEN name == "Tim"
YIELD <output></output>	No	Specifies the output format of the statement.	YIELD name AS Name

#### Example

```
// This query checks the properties of vertex "player101".
nebula> FETCH PROP ON player "player101" YIELD properties(vertex);
+-------+
| properties(VERTEX) |
+------+
| (age: 36, name: "Tony Parker"} |
+------+
// This query updates the age property and returns name and the new age.
nebula> UPDATE VERTEX ON player "player101" \
SFIT age = age + 2 \
WHEN name == "Tony Parker" \
YIELD name AS Name, age AS Age;
+------++--++
| Name | Age |
+------++--++
| "Tony Parker" | 38 |
+------++--+++
```

## 4.11.4 UPSERT VERTEX

The UPSERT statement is a combination of UPDATE and INSERT. You can use UPSERT VERTEX to update the properties of a vertex if it exists or insert a new vertex if it does not exist.



The performance of UPSERT is much lower than that of INSERT because UPSERT is a read-modify-write serialization operation at the partition level.

Banger

Don't use UPSERT for scenarios with highly concurrent writes. You can use UPDATE or INSERT instead.

#### Syntax

UPSERT VERTEX ON <tag> <vid> SET <update\_prop> [WHEN <condition>] [YIELD <output>]

Parameter	Required	Description	Example
ON <tag></tag>	Yes	Specifies the tag of the vertex. The properties to be updated must be on this tag.	ON player
<vid></vid>	Yes	Specifies the ID of the vertex to be updated or inserted.	"player100"
SET <update_prop></update_prop>	Yes	Specifies the properties to be updated and how they will be updated.	SET age = age +1
WHEN <condition></condition>	No	Specifies the filter conditions.	WHEN name == "Tim"
YIELD <output></output>	No	Specifies the output format of the statement.	YIELD name AS Name

#### Insert a vertex if it does not exist

If a vertex does not exist, it is created no matter the conditions in the WHEN clause are met or not, and the SET clause always takes effect. The property values of the new vertex depend on:

- How the SET clause is defined.
- Whether the property has a default value.

For example, if:

- The vertex to be inserted will have properties name and age based on the tag player.
- The SET clause specifies that age = 30.

Then the property values in different cases are listed as follows:

Are WHEN conditions met	If properties have default values	Value of name	Value of age
Yes	Yes	The default value	30
Yes	No	NULL	30
No	Yes	The default value	30
No	No	NULL	30

Here are some examples:

```
// This query checks if the following three vertices exist. The result "Empty set" indicates that the vertices do not exist.
nebula> FETCH PROP ON * "player666", "player667", "player668" YIELD properties(vertex);
| properties(VERTEX) |
+-----+
Empty set
nebula> UPSERT VERTEX ON player "player666" \
         SET age = 30 \
WHEN name == "Joe" \
         YIELD name AS Name, age AS Age;
+----
| Name | Age
| __NULL__ | 30
nebula> UPSERT VERTEX ON player "player666" \
         SET age = 31 \
WHEN name == "Joe" \
YIELD name AS Name, age AS Age;
| Name | Age |
|__NULL__ | 30 |
nebula> UPSERT VERTEX ON player "player667" \setminus
         SET age = 31 \
YIELD name AS Name, age AS Age;
+----
Name
           Age
| __NULL__ | 31 |
nebula> UPSERT VERTEX ON player "player668" \
SET name = "Amber", age = age + 1 \
YIELD name AS Name, age AS Age;
     | Name | Age
| "Amber" | __NULL__ |
```

In the last query of the preceding examples, since age has no default value, when the vertex is created, age is NULL, and age = age + 1 does not take effect. But if age has a default value, age = age + 1 will take effect. For example:

```
nebula> CREATE TAG IF NOT EXISTS player_with_default(name string, age int DEFAULT 20);
Execution succeeded
nebula> UPSERT VERTEX ON player_with_default "player101" \
    SET age = age + 1 \
    YIELD name AS Name, age AS Age;
+------+
| Name | Age |
+-----+
| ___NULL__ | 21 |
```

#### Update a vertex if it exists

If the vertex exists and the  $\ensuremath{\,{\tiny WHEN}}$  conditions are met, the vertex is updated.

```
nebula> FETCH PROP ON player "player101" YIELD properties(vertex);
+-----+
```

| properties(VERTEX) | +-----+ | {age: 36, name: "Tony Parker"} | +-----+ nebula> UPSERT VERTEX ON player "player101" \ SET age = age + 2 \ WHEN name == "Tony Parker" \ YIELD name AS Name, age AS Age; +------+ | Name | Age | +------+ | "Tony Parker" | 38 | +-----+

If the vertex exists and the  $\ensuremath{\,{\tiny WHEN}}$  conditions are not met, the update does not take effect.

nebula> FETCH PROP ON player "player101" YIELD properties(vertex);
+-----+
| properties(VERTEX) | +----+
| {age: 38, name: "Tony Parker"} |
+----+
nebula> UPSERT VERTEX ON player "player101" \
 SET age = age + 2 \
 WHEN name == "Someone else" \
 YIELD name AS Name, age AS Age;
+----++
| Name | Age |
+-----++
| "Tony Parker" | 38 |
+-----++

## 4.12 Edge statements

#### 4.12.1 INSERT EDGE

The INSERT EDGE statement inserts an edge or multiple edges into a graph space from a source vertex (given by src\_vid) to a destination vertex (given by dst\_vid) with a specific rank in NebulaGraph.

When inserting an edge that already exists, INSERT EDGE overrides the edge.

#### Syntax

```
INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) VALUES
<src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> )
[, <src_vid> -> <dst_vid>[@<rank>] : ( <prop_value_list> ), ...];
<prop_name_list> ::=
    [ <prop_name> [, <prop_name> ] ...]
<prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value_list> ::=
    [ <prop_value> [, <prop_value> ] ...]
```

• IF NOT EXISTS detects if the edge that you want to insert exists. If it does not exist, a new one will be inserted.

#### Q Note

- IF NOT EXISTS only detects whether exist and does not detect whether the property values overlap.
- IF NOT EXISTS will read to check whether the data exists, which will have a significant impact on performance.
- <cedge\_type> denotes the edge type, which must be created before INSERT EDGE. Only one edge type can be specified in this
  statement.

• <prop\_name\_list> is the property name list in the given <edge\_type>.

- src\_vid is the VID of the source vertex. It specifies the start of an edge.
- dst\_vid is the VID of the destination vertex. It specifies the end of an edge.
- rank is optional. It specifies the edge rank of the same edge type. The data type is int. If not specified, the default value is 0. You can insert many edges with the same edge type, source vertex, and destination vertex by using different rank values.

# **F**enCypher compatibility

OpenCypher has no such concept as rank.

<prop\_value\_List> must provide the value list according to <prop\_name\_List>. If the property values do not match the data type in the edge type, an error is returned. When the NOT NULL constraint is set for a given property, an error is returned if no property is given. When the default value for a property is NULL, you can omit to specify the property value. For details, see CREATE EDGE.

#### Examples

```
# The following example creates edge type el with no property and inserts an edge from vertex "10" to vertex "11" with no property.
nebula> CREATE EDGE IF NOT EXISTS e1();
nebula> INSERT EDGE e1 () VALUES "10"->"11":();
```

# The following example inserts an edge from vertex "10" to vertex "11" with no property. The edge rank is 1. nebula> INSERT EDGE e1 () VALUES "10"->"11"@1:(); nebula> CREATE EDGE IF NOT EXISTS e2 (name string, age int); nebula> INSERT EDGE e2 (name, age) VALUES "l1"->"13":("n1", 1); # The following example creates edge type e2 with two properties. nebula> INSERT EDGE e2 (name, age) VALUES \ "12"->"13":("n1", 1), "13"->"14":("n2", 2);

# In the following example, the insertion fails because "a13" is not int. nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", "a13");

An edge can be inserted/written with property values multiple times. Only the last written values can be read.

The following examples insert edge e2 with the new values for multiple times. nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 12); nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 13); nebula> INSERT EDGE e2 (name, age) VALUES "11"->"13":("n1", 14); nebula> FETCH PROP ON e2 "11"->"13" VIELD edge AS e; +-----+ | e | | +----++ | [:e2 "11"->"13" (@0 {age: 14, name: "n1"}] |

If you insert an edge that already exists with IF NOT EXISTS, there will be no modification.

#### Q Note

• NebulaGraph 3.6.0 allows dangling edges. Therefore, you can write the edge before the source vertex or the destination vertex exists. At this time, you can get the (not written) vertex VID through <edgetype>.\_src or <edgetype>.\_dst (which is not recommended).

• Atomic operation is not guaranteed during the entire process for now. If it fails, please try again. Otherwise, partial writing will occur. At this time, the behavior of reading the data is undefined. For example, if multiple machines are involved in the write operation, only one of the forward and reverse edges of a single edge is written successfully, or only part of the edge is written successfully when multiple edges are inserted. In this case, an error will be returned, so please execute the command again.

· Concurrently writing the same edge will cause an edge conflict error, so please try again later.

• The inserting speed of an edge is about half that of a vertex. Because in the storaged process, the insertion of an edge involves two tasks, while the insertion of a vertex involves only one task.

Last update: November 3, 2023

## 4.12.2 DELETE EDGE

The DELETE EDGE statement deletes one edge or multiple edges at a time. You can use DELETE EDGE together with pipe operators. For more information, see PIPE OPERATORS.

To delete all the outgoing edges for a vertex, please delete the vertex. For more information, see DELETE VERTEX.

#### Syntax

DELETE EDGE <edge\_type> <src\_vid> -> <dst\_vid>[@<rank>] [, <src\_vid> -> <dst\_vid>[@<rank>] ...]

## Caution

If no rank is specified, NebulaGraph only deletes the edge with rank 0. Delete edges with all ranks, as shown in the following example.

#### Examples

```
nebula> DELETE EDGE serve "player100" -> "team204"@0;
```

The following example shows that you can use DELETE EDGE together with pipe operators to delete edges that meet the conditions.

```
nebula> G0 FROM "player100" OVER follow \
WHERE dst(edge) == "player101" \
YIELD src(edge) AS src, dst(edge) AS dst, rank(edge) AS rank \
| DELETE EDGE follow $-.src->$-.dst @ $-.rank;
```

## 4.12.3 UPDATE EDGE

The UPDATE EDGE statement updates properties on an edge.

In NebulaGraph, UPDATE EDGE supports compare-and-swap (CAS).

#### Syntax

UPDATE EDGE ON <edge\_type> <src\_vid> -> <dst\_vid> [@<rank>] SET <update\_prop> [WHEN <condition>] [YIELD <output>]

Parameter	Required	Description	Example
ON <edge_type></edge_type>	Yes	Specifies the edge type. The properties to be updated must be on this edge type.	ON serve
<src_vid></src_vid>	Yes	Specifies the source vertex ID of the edge.	"player100"
<dst_vid></dst_vid>	Yes	Specifies the destination vertex ID of the edge.	"team204"
<rank></rank>	No	Specifies the rank of the edge. The data type is $\mbox{ int}.$	10
SET <update_prop></update_prop>	Yes	Specifies the properties to be updated and how they will be updated.	<pre>SET start_year = start_year +1</pre>
WHEN <condition></condition>	No	Specifies the filter conditions. If <condition> evaluates to false, the SET clause does not take effect.</condition>	WHEN end_year < 2010
YIELD <output></output>	No	Specifies the output format of the statement.	YIELD start_year AS Start_Year

#### Example

The following example checks the properties of the edge with the GO statement.

The following example updates the start\_year property and returns the end\_year and the new start\_year.

```
nebula> UPDATE EDGE on serve "player100" -> "team204"@0 \
    SET start_year = start_year + 1 \
    WHEN end_year > 2010 \
    YIELD start_year, end_year;
+------+
| start_year | end_year |
+----++
| 1998 | 2016 |
+-----++
```

## 4.12.4 UPSERT EDGE

The UPSERT statement is a combination of UPDATE and INSERT. You can use UPSERT EDGE to update the properties of an edge if it exists or insert a new edge if it does not exist.

The performance of UPSERT is much lower than that of INSERT because UPSERT is a read-modify-write serialization operation at the partition level.



Syntax

UPSERT EDGE ON <edge\_type>

<pre><src_vid> -&gt; <dst_vid> [@rank_ SET <update_prop> [WHEN <condition>] [YIELD <properties>]</properties></condition></update_prop></dst_vid></src_vid></pre>	J		
Parameter	Required	Description	Example
ON <edge_type></edge_type>	Yes	Specifies the edge type. The properties to be updated must be on this edge type.	ON serve
<src_vid></src_vid>	Yes	Specifies the source vertex ID of the edge.	"player100"
<dst_vid></dst_vid>	Yes	Specifies the destination vertex ID of the edge.	"team204"
<rank></rank>	No	Specifies the rank of the edge.	10
SET <update_prop></update_prop>	Yes	Specifies the properties to be updated and how they will be updated.	<pre>SET start_year =   start_year +1</pre>
WHEN <condition></condition>	No	Specifies the filter conditions.	WHEN end_year < 2010
YIELD <output></output>	No	Specifies the output format of the statement.	YIELD start_year AS Start_Year

#### Insert an edge if it does not exist

If an edge does not exist, it is created no matter the conditions in the WHEN clause are met or not, and the SET clause takes effect. The property values of the new edge depend on:

- How the SET clause is defined.
- Whether the property has a default value.

For example, if:

- The edge to be inserted will have properties start\_year and end\_year based on the edge type serve.
- The SET clause specifies that end\_year = 2021.

Then the property values in different cases are listed as follows:

Are WHEN conditions met	If properties have default values	Value of start_year	Value of end_year
Yes	Yes	The default value	2021
Yes	No	NULL	2021
No	Yes	The default value	2021
No	No	NULL	2021

Here are some examples:

<pre>// This example checks if the following three vertices have nebula&gt; G0 FROM "player666", "player667", "player668" \</pre>	any outgoing serve edge. The result "Empty set" indicates that such an edge does not exist. end_year;
properties(EDGE).start_year   properties(EDGE).end_year	
++	
Empty set	
<pre>nebula&gt; UPSERT EDGE on serve \     "player666" -&gt; "team200"@0 \     SET end_year = 2021 \     WHEN end_year = 2010 \     YIELD start_year, end_year; ++   start_year   end_year   ++  NULL   2021   +++</pre>	
<pre>nebula&gt; UPSERT EDGE on serve \     "player666" -&gt; "team200"@0 \     SET end_year = 2022 \     WHEN end_year == 2010 \     YIELD start_year, end_year; ++   start_year   end_year   +++  NULL   2021   +++</pre>	
<pre>nebula&gt; UPSERT EDGE on serve \     "player667" -&gt; "team200"@0 \     SET end_year = 2022 \     YIELD start_year, end_year; ++   start_year   end_year   +++</pre>	
NULL   2022   ++	
<pre>nebula&gt; UPSERT EDGE on serve \</pre>	

In the last query of the preceding example, since end\_year has no default value, when the edge is created, end\_year is NULL, and end\_year = end\_year + 1 does not take effect. But if end\_year has a default value, end\_year = end\_year + 1 will take effect. For example:

nebula> CREATE EDGE IF NOT EXISTS serve\_with\_default(start\_year int, end\_year int DEFAULT 2010); Execution succeeded nebula> UPSERT EDGE on serve\_with\_default \ "player668" -> "team200" \ SET end\_year = end\_year + 1 \ YIELD start\_year, end\_year; +------+ | start\_year | end\_year | +------+ | \_\_NULL\_\_ | 2011 |

#### Update an edge if it exists

If the edge exists and the WHEN conditions are met, the edge is updated.

If the edge exists and the WHEN conditions are not met, the update does not take effect.

## 4.13 Native index statements

#### 4.13.1 Index overview

Indexes are built to fast process graph queries. Nebula Graph supports two kinds of indexes: native indexes and full-text indexes. This topic introduces the index types and helps choose the right index.

#### Usage Instructions

- Indexes can improve query performance but may reduce write performance.
- An index is a prerequisite for locating data when executing a LOOKUP statement. If there is no index, an error will be reported when executing the LOOKUP statement.
- When using an index, NebulaGraph will automatically select the most optimal index.
- Indexes with high selectivity, that is, when the ratio of the number of records with unique values in the index column to the total number of records is high (for example, the ratio for ID numbers is 1), can significantly improve query performance. For indexes with low selectivity (such as country), query performance might not experience a substantial improvement.

#### Native indexes

Native indexes allow querying data based on a given property. Features are as follows.

- There are two kinds of native indexes: tag index and edge type index.
- Native indexes must be updated manually. You can use the REBUILD INDEX statement to update native indexes.
- Native indexes support indexing multiple properties on a tag or an edge type (composite indexes), but do not support indexing across multiple tags or edge types.

OPERATIONS ON NATIVE INDEXES

- CREATE INDEX
- SHOW CREATE INDEX
- SHOW INDEXES
- DESCRIBE INDEX
- REBUILD INDEX
- SHOW INDEX STATUS
- DROP INDEX
- LOOKUP
- MATCH
- · Geography index

## Full-text indexes

Full-text indexes are used to do prefix, wildcard, regexp, and fuzzy search on a string property. Features are as follows.

- · Full-text indexes allow indexing just one property.
- Full-text indexes do not support logical operations such as AND , OR , and NOT .

#### Q Note

To do complete string matches, use native indexes.

## Null values

Indexes do not support indexing null values.

## **Range queries**

In addition to querying single results from native indexes, you can also do range queries. Not all the native indexes support range queries. You can only do range searches for numeric, date, and time type properties.

Last update: November 3, 2023

## 4.13.2 CREATE INDEX

#### Prerequisites

Before you create an index, make sure that the relative tag or edge type is created. For how to create tags or edge types, see CREATE TAG and CREATE EDGE.

For how to create full-text indexes, see Deploy full-text index.

#### Must-read for using indexes

The concept and using restrictions of indexes are comparatively complex. Before you use indexes, you must read the following sections carefully.

You can use CREATE INDEX to add native indexes for the existing tags, edge types, or properties. They are usually called as tag indexes, edge type indexes, and property indexes.

- Tag indexes and edge type indexes apply to queries related to the tag and the edge type, but do not apply to queries that are based on certain properties on the tag. For example, you can use LOOKUP to retrieve all the vertices with the tag player.
- Property indexes apply to property-based queries. For example, you can use the age property to retrieve the VID of all vertices that meet age == 19.

If a property index  $i_TA$  is created for the property A of the tag T and  $i_T$  for the tag T, the indexes can be replaced as follows (the same for edge type indexes):

- The query engine can use  $i_TA$  to replace  $i_T$ .
- In the MATCH and LOOKUP statements, i\_T may replace i\_TA for querying properties.

# L Jacy version compatibility

In previous releases, the tag or edge type index in the LOOKUP statement cannot replace the property index for property queries.

Although the same results can be obtained by using alternative indexes for queries, the query performance varies according to the selected index.

## Caution

Indexes can dramatically reduce the write performance. The performance can be greatly reduced. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.

Long indexes decrease the scan performance of the Storage Service and use more memory. We suggest that you set the indexing length the same as that of the longest string to be indexed. For variable-length string-type properties, the longest index length is 256 bytes; for fixed-length string-type properties, the longest index length is the length of the index itself.

#### Steps

If you must use indexes, we suggest that you:

- 1. Import the data into NebulaGraph.
- 2. Create indexes.
- 3. Rebuild indexes.
- 4. After the index is created and the data is imported, you can use LOOKUP or MATCH to retrieve the data. You do not need to specify which indexes to use in a query, NebulaGraph figures that out by itself.

## Note

If you create an index before importing the data, the importing speed will be extremely slow due to the reduction in the write performance.

Keep --disable\_auto\_compaction = false during daily incremental writing.

The newly created index will not take effect immediately. Trying to use a newly created index (such as LOOKUP or REBUILD INDEX) may fail and return can't find xxx in the space because the creation is implemented asynchronously. To make sure the follow-up operations work as expected, Wait for two heartbeat cycles, i.e., 20 seconds. To change the heartbeat interval, modify the heartbeat\_interval\_secs in the configuration files for all services.

## Danger

After creating a new index, or dropping the old index and creating a new one with the same name again, you must REBUILD INDEX. Otherwise, these data cannot be returned in the MATCH and LOOKUP statements.

#### Syntax

CREATE {TAG | EDGE} INDEX [IF NOT EXISTS] <index\_name> ON {<tag\_name> | <edg\_name>} ([<prop\_name\_list>]) [COMMENT '<comment>'];

Parameter	Description
TAG   EDGE	Specifies the index type that you want to create.
IF NOT EXISTS	Detects if the index that you want to create exists. If it does not exist, a new one will be created.
<index_name></index_name>	<ol> <li>The name of the index. It must be unique in a graph space. A recommended way of naming is         i_tagName_propName.</li> <li>By default, the name only supports 1-4 byte UTF-8 encoded characters, including English letters (case sensitive), numbers, Chinese characters, etc. However, it cannot include special characters other than the underscore (_), and cannot start with a number.</li> <li>To use special characters, reserved keywords, or start with a number, quote the entire name with backticks (`) and do not include periods (.) within the pair of backticks (`). For more information, see Keywords and reserved words.</li> <li>Note:         <ol> <li>If you name an index in Chinese and encounter a SyntaxError, you need to quote the Chinese characters with backticks (`).</li> </ol> </li> </ol>
	2. To include a backtick (`) in an index name, use a backslash to escape the backtick, such as \`; to include a backslash, the backslash itself also needs to be escaped, such as \ .
<tag_name>   <edge_name></edge_name></tag_name>	Specifies the name of the tag or edge associated with the index.
<prop_name_list></prop_name_list>	To index a <b>variable-length</b> string property, you must use prop_name(length) to specify the index length, and the maximum index length is 256. To index a tag or an edge type, ignore the prop_name_list.
COMMENT	The remarks of the index. The maximum length is 256 bytes. By default, there will be no comments on an index.

#### Create tag/edge type indexes

nebula> CREATE TAG INDEX IF NOT EXISTS player\_index on player();

nebula> CREATE EDGE INDEX IF NOT EXISTS follow\_index on follow();

After indexing a tag or an edge type, you can use the LOOKUP statement to retrieve the VID of all vertices with the tag, or the source vertex ID, destination vertex ID, and ranks of all edges with the edge type. For more information, see LOOKUP.

#### Create single-property indexes

```
nebula> CREATE TAG INDEX IF NOT EXISTS player_index_0 on player(name(10));
```

The preceding example creates an index for the name property on all vertices carrying the player tag. This example creates an index using the first 10 characters of the name property.

```
# To index a variable-length string property, you need to specify the index length.
nebula> CREATE TAG IF NOT EXISTS var_string(pl string);
nebula> CREATE TAG INDEX IF NOT EXISTS var ON var_string(pl(10));
# To index a fixed-length string property, you do not need to specify the index length.
nebula> CREATE TAG IF NOT EXISTS fix_string(pl FIXED_STRING(10));
nebula> CREATE TAG INDEX IF NOT EXISTS fix_oN fix_string(pl);
```

nebula> CREATE EDGE INDEX IF NOT EXISTS follow\_index\_0 on follow(degree);

#### Create composite property indexes

An index on multiple properties on a tag (or an edge type) is called a composite property index.

nebula> CREATE TAG INDEX IF NOT EXISTS player\_index\_1 on player(name(10), age);

## Caution

Creating composite property indexes across multiple tags or edge types is not supported.

#### Q Note

NebulaGraph follows the left matching principle to select indexes.

Last update: December 28, 2023

## 4.13.3 SHOW INDEXES

 $\ensuremath{\mathsf{SHOW}}$  INDEXES shows the defined tag or edge type indexes names in the current graph space.

## Syntax

SHOW {TAG | EDGE} INDEXES

## Examples

nebula> SHOW TAG INDEXES;			
Index Name	By Tag	Columns	
"fix"   "player_index_0"   "player_index_1"   "var" +	"fix_string"   "player"   "player"   "var_string"	["p1"]   ["name"]   ["name", "age"]   ["p1"]	
nebula> SHOW EDGE INDEXES; ++			

Ľ	Index Name	L	By Edge	1	Columns	L
L	"follow_index"		"follow"		[]	
+-		+-		-+-		+

# ₽\_gacy version compatibility

In NebulaGraph 2.x, the SHOW TAG/EDGE INDEXES statement only returns Names .

Last update: November 3, 2023

## 4.13.4 SHOW CREATE INDEX

SHOW CREATE INDEX shows the statement used when creating a tag or an edge type. It contains detailed information about the index, such as its associated properties.

#### Syntax

SHOW CREATE {TAG | EDGE} INDEX <index\_name>;

#### Examples

You can run SHOW TAG INDEXES to list all tag indexes, and then use SHOW CREATE TAG INDEX to show the information about the creation of the specified index.

nebula> SHOW TAG IND	EXES;	
++	+	+
Index Name	By Tag	Columns
++	+	+
"player_index_0"     "player_index_1"	"player"   "player"	[]   ["name"]
±		+

nebula> SHOW CREATE TAG INDEX player\_index\_1;

	T	
	Tag Index Name   Create Tag Index	
+	+	+
I	"player_index_1"   "CREATE TAG INDEX `player_index_1` (	ON `player` (
	`name`(20)	
	)"	
4		+

Edge indexes can be queried through a similar approach.

nebula> SHOW EDGE INDEXES;
++
Index Name   By Edge   Columns
++
"follow_index"   "follow"   []
++
nebula> SHOW CREATE EDGE INDEX follow_index;
+
Edge Index Name   Create Edge Index
+
"follow_index"   "CREATE EDGE INDEX `follow_index` ON `follow` (

+-----

## 4.13.5 DESCRIBE INDEX

DESCRIBE INDEX can get the information about the index with a given name, including the property name (Field) and the property type (Type) of the index.

#### Syntax

DESCRIBE {TAG | EDGE} INDEX <index\_name>;

## Examples

nebula> DE	SCRIBE TAG INDEX	player_index_0;
++   Field	Туре	+   +
"name"   ++	"fixed_string(3	0)"   +
nebula> DE	SCRIBE TAG INDEX	<pre>player_index_1;</pre>
nebula> DE ++   Field   ++	SCRIBE TAG INDEX Type	player_index_1; +   +

## 4.13.6 REBUILD INDEX

# **D**anger • If data is updated or inserted before the creation of the index, you must rebuild the indexes manually to make sure that the indexes contain the previously added data. Otherwise, you cannot use LOOKUP and MATCH to query the data based on the index. If the index is created before any data insertion, there is no need to rebuild the index.

When the rebuild of an index is incomplete, queries that rely on the index can use only part of the index and therefore cannot obtain accurate results.

You can use REBUILD INDEX to rebuild the created tag or edge type index. For details on how to create an index, see CREATE INDEX.

## Caution

The speed of rebuilding indexes can be optimized by modifying the rebuild\_index\_part\_rate\_limit and snapshot\_batch\_size parameters in the configuration file. In addition, greater parameter values may result in higher memory and network usage, see Storage Service configurations for details.

#### Syntax

```
REBUILD {TAG | EDGE} INDEX [<index name list>]:
<index_name_list>::=
   [index_name [, index_name] ...]
```

- Multiple indexes are permitted in a single REBUILD statement, separated by commas. When the index name is not specified, all tag or edge indexes are rebuilt.
- After the rebuilding is complete, you can use the SHOW {TAG | EDGE} INDEX STATUS command to check if the index is successfully rebuilt. For details on index status, see SHOW INDEX STATUS.

#### Examples

nebula> CREATE TAG nebula> CREATE TAG	IF NOT EXISTS person( INDEX IF NOT EXISTS s	(name string, single_person_	age int, gender string, en index ON person(name(10))	mail string); ;
# The following ex nebula> REBUILD TA ++   New Job Id   ++   31   ++	ample rebuilds an inde G INDEX single_person_	ex and returns index;	s the job ID.	
# The following ex nebula> SHOW TAG I	ample checks the index NDEX STATUS;	status.		
+	++	÷		
Name	Index Status			
"single_person_i	ndex"   "FINISHED"			
+	++	÷		
# You can also use nebula> SHOW JOB 3	e "SHOW JOB <job_id>" t 1;</job_id>	to check if th	ne rebuilding process is co	omplete.
++	+	•4	•	+
Job Id(TaskId)	Command(Dest)	Status	Start Time	Stop Time
31       0       1       2       "Total:3"	"REBUILD_TAG_INDEX" "storaged1" "storaged2" "storaged0" "Succeeded:3"	"FINISHED" "FINISHED" "FINISHED" "FINISHED" "FINISHED" "Fai Led:0"	2021-07-07T09:04:24.000 2021-07-07T09:04:24.000 2021-07-07T09:04:24.000 2021-07-07T09:04:24.000 2021-07-07T09:04:24.000 "In Progress:0"	2021-07-07T09:04:24.000 2021-07-07T09:04:28.000 2021-07-07T09:04:28.000 2021-07-07T09:04:28.000 2021-07-07T09:04:28.000

Error Code

"SUCCEEDED" "SUCCEEDED

"SUCCEEDED"

"SUCCEEDED"

NebulaGraph creates a job to rebuild the index. The job ID is displayed in the preceding return message. To check if the rebuilding process is complete, use the SHOW JOB <job\_id> statement. For more information, see SHOW JOB.

## 4.13.7 SHOW INDEX STATUS

 $\ensuremath{\mathsf{SHOW}}$  INDEX STATUS returns the name of the created tag or edge type index and its status of job.

The status of rebuilding indexes includes:

- QUEUE : The job is in a queue.
- RUNNING : The job is running.
- FINISHED : The job is finished.
- FAILED : The job has failed.
- STOPPED : The job has stopped.
- INVALID : The job is invalid.

## Note

For details on how to create an index, see CREATE INDEX.

#### Syntax

SHOW {TAG | EDGE} INDEX STATUS;

#### Example

nebula> SHOW TAG INDE)	K STATUS;
+	++
Name	Index Status
+	++
"player_index_0"	"FINISHED"
"player_index_1"	"FINISHED"
+	+ +

Last update: November 3, 2023

## 4.13.8 DROP INDEX

DROP INDEX removes an existing index from the current graph space.

## Prerequisite

Running the DROP INDEX statement requires some privileges of DROP TAG INDEX and DROP EDGE INDEX in the given graph space. Otherwise, NebulaGraph throws an error.

#### Syntax

DROP {TAG | EDGE} INDEX [IF EXISTS] <index\_name>;

IF EXISTS : Detects whether the index that you want to drop exists. If it exists, it will be dropped.

## Example

nebula> DROP TAG INDEX player\_index\_0;

## 4.14 Full-text index statements

#### 4.14.1 Full-text index restrictions

This topic introduces the restrictions for full-text indexes. Please read the restrictions very carefully before using the full-text indexes.

## Caution

The full-text index feature has been redone in version 3.6.0 and is not compatible with previous versions. If you want to continue to use wildcards, regulars, fuzzy matches, etc., there are 3 ways to do so as follows:

- Delete the original full-text index, rebuild the full-text index in the new way, and use the new query syntax.
- Delete the original full-text index and use the native index and string operators directly.

• Continue to use the previous version of NebulaGraph and its full-text index.

For now, full-text search has the following limitations:

- Currently, full-text search supports LOOKUP statements only.
- The full-text index name can contain only numbers, lowercase letters, and underscores.
- The names of full-text indexes within different graph spaces cannot be duplicated.
- The query returns 10 records by default. You can use the LIMIT clause to return more records, up to 10,000. You can modify the ElasticSearch parameters to adjust the maximum number of records returned.
- If there is a full-text index on the tag/edge type, the tag/edge type cannot be deleted or modified.
- The type of properties must be STRING or FIXED\_STRING.
- Full-text index can not be applied to search multiple tags/edge types.
- Full-text index can not search properties with value NULL.
- Altering Elasticsearch indexes is not supported at this time.
- Modifying the analyzer is not supported. You have to delete the index data and then specify the analyzer when you rebuild the index.
- Make sure that you start the Elasticsearch cluster and Nebula Graph at the same time. If not, the data writing on the Elasticsearch cluster can be incomplete.
- It may take a while for Elasticsearch to create indexes. If Nebula Graph warns no index is found, you can check the status of the indexing task.
- NebulaGraph clusters deployed with K8s do not have native support for the full-text search feature. However, you can manually deploy the feature yourself.

#### 4.14.2 Deploy full-text index

Nebula Graph full-text indexes are powered by Elasticsearch. This means that you can use Elasticsearch full-text query language to retrieve what you want. Full-text indexes are managed through built-in procedures. They can be created only for variable STRING and FIXED\_STRING properties when the listener cluster and the Elasticsearch cluster are deployed.

#### Precaution

Before you start using the full-text index, please make sure that you know the restrictions.

#### **Deploy Elasticsearch cluster**

To deploy an Elasticsearch cluster, see Kubernetes Elasticsearch deployment or Elasticsearch installation.

## Note

To support external network access to Elasticsearch, set network.host to 0.0.0.0 in config/elasticsearch.yml.

You can configure the Elasticsearch to meet your business needs. To customize the Elasticsearch, see Elasticsearch Document.

#### Sign in to the text search clients

When the Elasticsearch cluster is deployed, use the SIGN IN statement to sign in to the Elasticsearch clients. Multiple elastic\_ip:port pairs are separated with commas. You must use the IPs and the port number in the configuration file for the Elasticsearch.

SYNTAX

```
SIGN IN TEXT SERVICE (<elastic_ip:port>, {HTTP | HTTPS} [,"<username>", "<password>"]) [, (<elastic_ip:port>, ...)];
```

#### EXAMPLE

nebula> SIGN IN TEXT SERVICE (192.168.8.100:9200, HTTP);

#### O Note

Elasticsearch does not have a username or password by default. If you configured a username and password, you need to specify them in the SIGN IN statement.

## Caution

The Elasticsearch client can only be logged in once, and if there are changes, you need to SIGN OUT and then SIGN IN again, and the client takes effect globally, and multiple graph spaces share the same Elasticsearch client.

#### Show text search clients

The SHOW TEXT SEARCH CLIENTS statement can list the text search clients.

SYNTAX

SHOW TEXT SEARCH CLIENTS;

#### EXAMPLE

nebula> SHOW TEXT SEARCH CLIENTS;

++   "ELASTICSEARCH"   "192.168.8.100"   9200		Туре		Host		Port	
"ELASTICSEARCH"   "192.168.8.100"   9200	+		-+-		+	+	
	l	"ELASTICSEARCH"		"192.168.8.100"	I	9200	

## Sign out to the text search clients

The SIGN OUT TEXT SERVICE statement can sign out all the text search clients.

SYNTAX

SIGN OUT TEXT SERVICE;

EXAMPLE

nebula> SIGN OUT TEXT SERVICE;

Last update: November 3, 2023

#### 4.14.3 Deploy Raft Listener for NebulaGraph Storage service

Full-text index data is written to the Elasticsearch cluster asynchronously. The Raft Listener (Listener for short) is a separate process that fetches data from the Storage Service and writes them into the Elasticsearch cluster.

#### Prerequisites

- You have read and fully understood the restrictions for using full-text indexes.
- You have deployed a NebulaGraph cluster.
- You have deployed a Elasticsearch cluster.
- You have prepared one or multiple servers to run one or multiple raft listeners.

#### Precautions

- The Storage Service that you want to run as the Listener must have the same or later release with all the other Nebula Graph services in the cluster.
- For now, you can only add all Listeners to a graph space once and for all. Trying to add a new Listener to a graph space that already has a Listener will fail. To add all Listeners, set them in one statement.

#### **Deployment process**

STEP 1: INSTALL THE LISTENER SERVICE

The Listener service uses the same binary as the storaged service. However, the configuration files are different and the processes use different ports. You can install NebulaGraph on all servers that need to deploy a Listener, but only the storaged service can be used. For details, see Install NebulaGraph by RPM or DEB Package.

STEP 2: PREPARE THE CONFIGURATION FILE FOR THE LISTENER

In the etc directory, remove the suffix from nebula-storaged-listener.conf.default or nebula-storaged-listener.conf.production to nebulastoraged-listener.conf, and then modify the configuration content.

Name	Default value	Description
daemonize	true	When set to true, the process is a daemon process.
pid_file	pids/nebula-metad- listener.pid	The file that records the process ID.
meta_server_addrs	-	IP (or hostname) and ports of all Meta services. Multiple Meta services are separated by commas.
local_ip	-	The local IP (or hostname) of the Listener service. Use real IP addresses instead of domain names or loopback IP addresses such as 127.0.0.1.
port	-	The listening port of the RPC daemon of the Listener service.
heartbeat_interval_secs	10	The heartbeat interval of the Meta service. The unit is second (s).
listener_path	data/listener	The WAL directory of the Listener. Only one directory is allowed.
data_path	data	For compatibility reasons, this parameter can be ignored. Fill in the default value $\ensuremath{data}$ .
part_man_type	memory	The type of the part manager. Optional values are $\ensuremath{\operatorname{memory}}$ and $\ensuremath{\operatorname{meta}}$ .
rocksdb_batch_size	4096	The default reserved bytes for batch operations.
rocksdb_block_cache	4	The default block cache size of BlockBasedTable. The unit is Megabyte (MB).
engine_type	rocksdb	The type of the Storage engine, such as rocksdb , memory , etc.
part_type	simple	The type of the part, such as simple, consensus, etc.

Most configurations are the same as the configurations of Storage Service. This topic only introduces the differences.

STEP 3: START LISTENERS

To initiate the Listener, navigate to the installation path of the desired cluster and execute the following command:

./bin/nebula-storaged --flagfile etc/nebula-storaged-listener.conf

STEP 4: ADD LISTENERS TO NEBULAGRAPH

Connect to NebulaGraph and run USE <space> to enter the graph space that you want to create full-text indexes for. Then run the following statement to add a Listener into NebulaGraph.

ADD LISTENER ELASTICSEARCH <listener\_ip:port> [,<listener\_ip:port>, ...]

### 

You must use real IPs for a Listener.

Add all Listeners in one statement completely.

nebula> ADD LISTENER ELASTICSEARCH 192.168.8.100:9789,192.168.8.101:9789;

#### Show Listeners

Run the SHOW LISTENER statement to list all Listeners.

EXAMPLE

ne	ebula> S	SHC	W LISTENER;				
+.		+-		+-		+-	++
l	PartId	I	Туре	l	Host	l	Host Status
+. 	1		"ELASTICSEARCH"	+-	""192.168.8.100":9789"	+- 	"ONLINE"

```
| 2 | "ELASTICSEARCH" | ""192.168.8.100":9789" | "ONLINE" |
| 3 | "ELASTICSEARCH" | ""192.168.8.100":9789" | "ONLINE" |
```

#### **Remove Listeners**

Run the REMOVE LISTENER ELASTICSEARCH statement to remove all Listeners in a graph space.

#### EXAMPLE

nebula> REMOVE LISTENER ELASTICSEARCH;

Last update: November 22, 2023
### 4.14.4 Full-text indexes

Full-text indexes are used to do prefix, wildcard, regexp, and fuzzy search on a string property.

You can use the WHERE clause to specify the search strings in LOOKUP statements.

### Prerequisite

Before using the full-text index, make sure that you have deployed a Elasticsearch cluster and a Listener cluster. For more information, see Deploy Elasticsearch and Deploy Listener.

### Precaution

Before using the full-text index, make sure that you know the restrictions.

### **Full Text Queries**

Full-text queries enable you to search for parsed text fields, using a parser with strict syntax to return content based on the query string provided. For details, see Query string query.

### Syntax

CREATE FULL-TEXT INDEXES

CREATE FULLTEXT {TAG | EDGE} INDEX <index\_name> 0N {<tag\_name> | <edge\_name> [ <prop\_name> [., <prop\_name>]...) [ANALYZER="<analyzer\_name>"];

- Composite indexes with multiple properties are supported when creating full-text indexes.
- <analyzer\_name> is the name of the analyzer. The default value is standard. To use other analyzers (e.g. IK Analysis), you need to make sure that the corresponding analyzer is installed in Elasticsearch in advance.

SHOW FULL-TEXT INDEXES

SHOW FULLTEXT INDEXES;

REBUILD FULL-TEXT INDEXES

REBUILD FULLTEXT INDEX;

# Caution

When there is a large amount of data, rebuilding full-text index is slow, you can modify snapshot\_send\_files=false in the configuration file of Storage service(nebula-storaged.conf).

DROP FULL-TEXT INDEXES

DROP FULLTEXT INDEX <index\_name>;

USE QUERY OPTIONS

LOOKUP ON {<tap> | <edge\_type>} WHERE ES\_QUERY(<index\_name>, "<text>") YIELD <return\_list> [| LIWIT [<offset>,] <number\_rows>];

<return list> <prop\_name> [AS <prop\_alias>] [, <prop\_name> [AS <prop\_alias>] ...] [, id(vertex) [AS <prop\_alias>]] [, score() AS <score\_alias>]

- index\_name : The name of the full-text index.
- text : Search conditions. The where can only be followed by the ES\_QUERY, and all judgment conditions must be written in the text. For supported syntax, see Query string syntax.
- score(): The score calculated by doing N degree expansion for the eligible vertices. The default value is 1.0. The higher the score, the higher the degree of match. The return value is sorted by default from highest to lowest score. For details, see Search and Scoring in Lucene.

### Examples

// This example creates the graph space nebula> CREATE SPACE IF NOT EXISTS basketballplayer (partition\_num=3,replica\_factor=1, vid\_type=fixed\_string(30));

// This example signs in the text service. nebula> SIGN IN TEXT SERVICE (192.168.8.100:9200, HTTP);

// This example checks the text service status.
nebula> SHOW TEXT SEARCH CLIENTS:

nebucu	011011 12/11	02,		CLILING	· ,	
+		+			+	+
Туре			lost		- 1	Port
+		+			+	+
"ELAST	TICSEARCH"	[ ]	'192.	168.8.1	100"	9200
+		+			+	+

// This example switches the graph space. nebula> USE basketballplayer;

// This example adds the listener to the NebulaGraph cluster. nebula> ADD LISTENER ELASTICSEARCH 192.168.8.100:9789;

// This example checks the listener status. When the status is `Online`, the listener is ready.

nebu La>	SHOW	LISIENER;	

PartId   Type	Host	Host Status
1   "ELASTI   2   "ELASTI   3   "ELASTI	LCSEARCH"   ""192.168.8.1 LCSEARCH"   ""192.168.8.1 LCSEARCH"   ""192.168.8.1 LCSEARCH"   ""192.168.8.1	.00":9789"   "ONLINE"   .00":9789"   "ONLINE"   .00":9789"   "ONLINE"   .00":9789"   "ONLINE"

// This example creates the tag. nebula> CREATE TAG IF NOT EXISTS player(name string, city string);

// This example creates a single-attribute full-text index. nebula> CREATE FULITEXT TAG INDEX fulltext\_index\_1 ON player(name) ANALYZER="standard";

// This example creates a multi-attribute full-text indexe. nebula> CREATE FULLTEXT TAG INDEX fulltext\_index\_2 ON player(name,city) ANALYZER="standard";

// This example rebuilds the full-text index nebula> REBUILD FULLTEXT INDEX;

### // This example shows the full-text index. CHOW EILLITEVT THDEVE

	bucu onon roccreat			,						
+-		+-		+		-+		+		+
	Name		Schema	Type	Schema Name		Fields		Analyzer	
+-		+-		+-		-+-		+		+
L	"fulltext_index_1"	I	"Tag"	1	"player"	1	"name"		"standard	"
	"fulltext_index_2"		"Tag"		"player"		"name,	city"	"standard	"

// This example inserts the test data. nebula> INSERT VERTEX player(name, city) VALUES \ ula> INSERT VERTEX player(name, city) VALUES \
 "Russell Westbrook": ("Russell Westbrook", "Los Angeles"), \
 "Chris Paul": ("Chris Paul", "Houston"), \
 "Boris Diaw": ("Boris Diaw", "Houston"), \
 "Danug Green": ("Danug Green", "Philadelphia"), \
 "Tim Duncan": ("Tim Duncan", "New York"), \
 "James Harden": ("James Harden", "New York"), \
 "Jony Green": (Chrong Parker", "Chicago"), \
 "Aron Baynes": ("Kon Baynes", "Chicago"), \
 "Ban Simmons": ("Ben Simmons", "Phoenix"), \
 "Bake Griffin": ("Blake Griffin", "Phoenix");

// These examples run test queries.

nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_1,"Chris") YIELD id(vertex);

| id(VERTEX)

| "Chris Paul"

nebula> LOOKUP ON player WHERE ES OUERY(fulltext index 1."Harden") YIELD properties(vertex):

properties(VERTEX)		
+	+	
{_vid: "James Harden", city: "New York", name: "James Harden"}	1	
+	+	

### nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_1,"Da\*") YIELD properties(vertex);

### | properties(VERTEX)

### nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_1,"\*b\*") YIELD id(vertex);

nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_1,"\*b\*") YIELD id(vertex) | LIMIT 2,3;

+----+ | id(VERTEX) | +----+ | "Aron Baynes" | | "Ben Simmons" | | "Blake Griffin" |

nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_1,"\*b\*") YIELD id(vertex) | YIELD count(\*);

+----+ | count(\*) | +----+ | 5 |

nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_1,"\*b\*") YIELD id(vertex), score() AS score;

"Russell Westbrook"     1.0       "Boris Diaw"     1.0       "Aron Baynes"     1.0       "Ben Simmons"     1.0       "Blake Griffin"     1.0	   +.	id(VERTEX)	score
		"Russell Westbrook"   "Boris Diaw"   "Aron Baynes"   "Ben Simmons"   "Blake Griffin"	1.0   1.0   1.0   1.0   1.0   1.0

// For documents containing a word `b`, its score will be multiplied by a weighting factor of 4, while for documents containing a word `c`, the default weighting factor of 1 is used. nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index1,"\*b\*^4 OR \*c\*") YIELD id(vertex), score() AS score;

 +.	id(VERTEX)		score	
İ.	"Russell Westbrook"	i	4.0	İ
Ĺ	"Boris Diaw"	İ	4.0	İ
L	"Aron Baynes"		4.0	
Ĺ	"Ben Simmons"	İ	4.0	İ
L	"Blake Griffin"		4.0	
L	"Chris Paul"		1.0	
L	"Tim Duncan"		1.0	
+		+		+

// When using a multi-attribute full-text index query, the conditions are matched within all properties of the index. nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_2, "\*h\*") YIELD properties(vertex);

,
properties(VERTEX)
+
{_vid: "Chris Paul", city: "Houston", name: "Chris Paul"}
{_vid: "Boris Diaw", city: "Houston", name: "Boris Diaw"}
{_vid: "David West", city: "Philadelphia", name: "David West"}
{_vid: "James Harden", city: "New York", name: "James Harden"}
{_vid: "Tony Parker", city: "Chicago", name: "Tony Parker"}
{_vid: "Aron Baynes", city: "Chicago", name: "Aron Baynes"}
{_vid: "Ben Simmons", city: "Phoenix", name: "Ben Simmons"}
{_vid: "Blake Griffin", city: "Phoenix", name: "Blake Griffin"}
{_vid: "Danny Green", city: "Philadelphia", name: "Danny Green"}
+

// When using multi-attribute full-text index queries, you can specify different text for different properties for the query. nebula> LOOKUP ON player WHERE ES\_QUERY(fulltext\_index\_2,"name:\*b\* AND city:Houston") YIELD properties(vertex); +------+

### | properties(VERTEX)

- +-----+ | {\_vid: "Boris Diaw", city: "Houston", name: "Boris Diaw"} |
- +-----

// Delete single-attribute full-text index. nebula> DROP FULLTEXT INDEX fulltext\_index\_1;

# 4.15 Query tuning and terminating statements

### 4.15.1 EXPLAIN and PROFILE

EXPLAIN helps output the execution plan of an nGQL statement without executing the statement.

**PROFILE** executes the statement, then outputs the execution plan as well as the execution profile. You can optimize the queries for better performance according to the execution plan and profile.

### **Execution Plan**

The execution plan is determined by the execution planner in the NebulaGraph query engine.

The execution planner processes the parsed nGQL statements into actions. An action is the smallest unit that can be executed. A typical action fetches all neighbors of a given vertex, gets the properties of an edge, and filters vertices or edges based on the given conditions. Each action is assigned to an operator that performs the action.

For example, a SHOW TAGS statement is processed into two actions and assigned to a Start operator and a ShowTags operator, while a more complex GO statement may be processed into more than 10 actions and assigned to 10 operators.

### Syntax

• EXPLAIN

EXPLAIN [format= {"row" | "dot" | "tck"}] <your\_nGQL\_statement>;

• PROFILE

PROFILE [format= {"row" | "dot" | "tck"}] <your\_nGQL\_statement>;

### Output formats

The output of an EXPLAIN or a PROFILE statement has three formats, the default row format, the dot format, and the tck format. You can use the format option to modify the output format. Omitting the format option indicates using the default row format.

# The row format

The row format outputs the return message in a table as follows.

### • EXPLAIN

nebula> EXPLAIN Execution succe	format="row" SH eded (time spent	OW TAGS; 327/892 us)	
Execution Plan			
+   id   name	+   dependencies	+   profiling data	+
1   ShowTags	0	   	
0   Start	-+   _+	+   +	+

### • PROFILE

nebula> +	PROFILE	format	="row"	SHOW	TAGS;
Name +	 -+				
player +	 -+				
team +	-+				
Got 2 ro	ws (time	spent	2038/2	2728 i	ıs)

### Execution Plan

1       ShowTags       0       ver: 0, rows: 1, execTime: 42us, totalTime: 1177us       outputVar: [{"colNames":[],"name":"ShowTags_1","type":"DATASET"}]         1       inputVar:       inputVar:		+ 1	name	dependencies	profiling data	+
	1	+ L	ShowTags	0	ver: 0, rows: 1, execTime: 42us, totalTime: 1177us	
	 	 +   (	 Start	+	ver: 0, rows: 0, execTime: 1us, totalTime: 57us	inputvar: 

### The descriptions are as follows.

Parameter	Description
id	The ID of the operator.
name	The name of the operator.
dependencies	The ID of the operator that the current operator depends on.
profiling data	The content of the execution profile. ver is the version of the operator. rows shows the number of rows to be output by the operator. execTime shows the execution time of action. totalTime is the sum of the execution time, the system scheduling time, and the queueing time.
operator info	The detailed information of the operator.

### The dot format

You can use the format="dot" option to output the return message in the dot language, and then use Graphviz to generate a graph of the plan.

# Note

Graphviz is open source graph visualization software. Graphviz provides an online tool for previewing DOT language files and exporting them to other formats such as SVG or JSON. For more information, see Graphviz Online.

nebula> EXPLAIN format="dot" SHOW TAGS; Execution succeeded (time spent 161/665 us) Execution Plan

```
plan
digraph exec_plan {
    rankdir=LR;
    "ShowTags_0"[label="ShowTags_0]outputVar: \[\{\"colNames\":\[],\"name\":\"__ShowTags_0\",\"type\":\"DATASET\"\}\]\l|inputVar:\l", shape=Mrecord];
    "Start_2"->"ShowTags_0";
    "Start_2"[label="Start_2|outputVar: \[\{\"colNames\":\[],\"name\":\"__Start_2\",\"type\":\"DATASET\"\}\]\l|inputVar: \l", shape=Mrecord];
}
```

The Graphviz graph transformed from the above DOT statement is as follows.

Start_2	)	ShowTags_0
outputVar: [{"colNames":[],"name":"Start_2","type":"DATASET"}]	┝	outputVar: [{"colNames":[],"name":"ShowTags_0","type":"DATASET"}]
inputVar:	)	inputVar:

### The tck format

The tck format is similar to a table, but without borders and dividing lines between rows. You can use the results as test cases for unit testing. For information on tck format test cases, see TCK cases.

### • EXPLAIN

nebula> EXPLAIN format="tck" FETCH PROP ON player "player_1","player_2","player_3" YIELD properties(vertex).name as name, properties(vertex).age as age; Execution succeeded (time spent 261µs/613.718µs) Execution Plan (optimize time 28 us)   id   name   dependencies   profiling data   operator info     2   Project   1           1   GetVertices   0           0   Start
PROFILE
nebula> PROFILE format="tck" FETCH PROP ON player "player_1","player_2","player_3" YIELD properties(vertex).name as name, properties(vertex).age as age; name   age   "Piter Park"   24     "aaa"   24     "acc"   24   Got 3 rows (time spent 1.474ms/2.19677ms) Execution Plan (notimize time 41 us)
id   name     dependencies   profiling data   operator info
2 Project   1   {"rows":3,"version":0}
1   GetVertices   0   {"resp[0]":{"exec":"232(us)","host":"127.0.0.1:9779","total":"758(us)"},"rows":3,"total_rpc":"875(us)","version":0}
0   Start   {"rows":0,"version":0}
Wed, 22 Mar 2023 23:16:13 CSI

# 4.15.2 Kill queries

KILL QUERY can terminate the query being executed, and is often used to terminate slow queries.

Note		
------	--	--

Users with the God role can kill any query. Other roles can only kill their own queries.

### Syntax

KILL QUERY (session=<session\_id>, plan=<plan\_id>);

- session\_id : The ID of the session.
- plan\_id : The ID of the execution plan.

The ID of the session and the ID of the execution plan can uniquely determine a query. Both can be obtained through the SHOW QUERIES statement.

### Examples

This example executes KILL QUERY in one session to terminate the query in another session.

nebula> KILL QUERY(SESSION=1625553545984255,PLAN=163);

The query will be terminated and the following information will be returned.

[ERROR (-1005)]: ExecutionPlanId[1001] does not exist in current Session.

### 4.15.3 Kill sessions

The KILL SESSION command is to terminate running sessions.

### Note

• Only the NebulaGraph root user can terminate sessions.

• After executing the KILL SESSION command, all Graph services synchronize the latest session information after 2\* session\_reclaim\_interval\_secs seconds (120 seconds by default).

### Syntax

You can run the KILL SESSION command to terminate one or multiple sessions. The syntax is as follows:

• To terminate one session

KILL {SESSION|SESSIONS} <SessionId>

- {SESSION|SESSIONS} : SESSION or SESSIONS , both are supported.
- <sessionId> : Specifies the ID of one session. You can run the SHOW SESSIONS command to view the IDs of sessions.
- To terminate multiple sessions

SHOW SESSIONS | YIELD \$-.SessionId AS sid [WHERE <filter\_clause>] | KILL {SESSION|SESSIONS} \$-.sid

### Q Note

The KILL SESSION command supports the pipeline operation, combining the SHOW SESSIONS command with the KILL SESSION command to terminate multiple sessions.

- Optional, the WHERE clause is used to filter sessions. <filter\_expression> specifies a session filtering expression, for example, WHERE \$-.CreateTime < datetime("2022-12-14T18:00:00"). If the WHERE clause is not specified, all sessions are terminated.
- Filtering conditions in a WHERE clause include: SessionId, UserName, SpaceName, CreateTime, UpdateTime, GraphAddr, Timezone, and ClientIp. You can run the SHOW SESSIONS command to view descriptions of these conditions.
- {SESSION|SESSIONS} : SESSION or SESSIONS , both are supported.

# Caution

Please use filtering conditions with caution to avoid deleting sessions by mistake.

### Examples

• To terminate one session

<sup>• [</sup>WHERE <filter\_clause>] :

nebula> KILL SESSION 1672887983842984

- To terminate multiple sessions
- Terminate all sessions whose creation time is less than 2023-01-05T18:00:00.

nebula> SHOW SESSIONS | YIELD \$-.SessionId AS sid WHERE \$-.CreateTime < datetime("2023-01-05T18:00:00") | KILL SESSIONS \$-.sid

• Terminates the two sessions with the earliest creation times.

nebula> SHOW SESSIONS | YIELD \$-.SessionId AS sid, \$-.CreateTime as CreateTime | ORDER BY \$-.CreateTime ASC | LIMIT 2 | KILL SESSIONS \$-.sid

• Terminates all sessions created by the username session\_user1.

nebula> SHOW SESSIONS | YIELD \$-.SessionId as sid WHERE \$-.UserName == "session\_user1" | KILL SESSIONS \$-.sid

• Terminate all sessions.

```
nebula> SHOW SESSIONS | YIELD $-.SessionId as sid | KILL SESSION $-.sid
// Or
nebula> SHOW SESSIONS | KILL SESSIONS $-.SessionId
```

# Caution

When you terminate all sessions, the current session is terminated. Please use it with caution.

# 4.16 Job manager and the JOB statements

The long-term tasks run by the Storage Service are called jobs, such as COMPACT, FLUSH, and STATS. These jobs can be timeconsuming if the data amount in the graph space is large. The job manager helps you run, show, stop, and recover jobs.

# Note

All job management commands can be executed only after selecting a graph space.

### 4.16.1 SUBMIT JOB BALANCE LEADER

Starts a job to balance the distribution of all the storage leaders in all graph spaces. It returns the job ID.

### For example:

nebula> SUBM	IT	JOB	BALANCE	LEADER
+	-+			
New Job Id				
+	-+			
33				
+	-+			

### 4.16.2 SUBMIT JOB COMPACT

The SUBMIT JOB COMPACT statement triggers the long-term RocksDB compact operation in the current graph space.

For more information about compact configuration, see Storage Service configuration.

For example:



### 4.16.3 SUBMIT JOB FLUSH

The SUBMIT JOB FLUSH statement writes the RocksDB memfile in the memory to the hard disk in the current graph space.

For example:

nebula> SUBMIT JOB FLUSH; +-----+ | New Job Id | +-----+ | 96 | +-----+

### 4.16.4 SUBMIT JOB STATS

The SUBMIT JOB STATS statement starts a job that makes the statistics of the current graph space. Once this job succeeds, you can use the SHOW STATS statement to list the statistics. For more information, see SHOW STATS.

# Note

If the data stored in the graph space changes, in order to get the latest statistics, you have to run SUBMIT JOB STATS again.

### For example:

nebula> SUBMIT	JOB	STATS;
++		
New Job Id		
++		
9		
++		

### 4.16.5 SUBMIT JOB DOWNLOAD/INGEST

The SUBMIT JOB DOWNLOAD HDFS and SUBMIT JOB INGEST commands are used to import the SST file into NebulaGraph. For detail, see Import data from SST files.

The SUBMIT JOB DOWNLOAD HDFS command will download the SST file on the specified HDFS.

The SUBMIT JOB INGEST command will import the downloaded SST file into NebulaGraph.

### For example:

```
nebula> SUBMIT JOB DOWNLOAD HDFS "hdfs://192.168.10.100:9000/sst";
+-----+
| New Job Id |
+----++
| 10 |
+----++
| 10 |
New Job Id |
+----++
| 11 |
```

### 4.16.6 SHOW JOB

The Meta Service parses a SUBMIT JOB request into multiple tasks and assigns them to the nebula-storaged processes. The SHOW JOB <job\_id> statement shows the information about a specific job and all its tasks in the current graph space.

job\_id is returned when you run the SUBMIT JOB statement.

For example:

nebula> SHOW JOB 8;	++	+	++++
Job Id(TaskId)   Command(Dest)	Status   Start	Time   Stop Time	Error Code
8   "STATS"   0   "192 168 8 129"	"FINISHED"   2022-10	D-18T08:14:45.000000   2022-10-18T0	08:14:45.000000   "SUCCEEDED"   08:15:13.000000   "SUCCEEDED"
"Total:1"   "Succeeded:1"	"Failed:0"   "In Pro	ogress:0"   ""	""

The descriptions are as follows.

Parameter	Description
Job Id(TaskId)	The first row shows the job ID and the other rows show the task IDs and the last row shows the total number of job-related tasks.
Command(Dest)	The first row shows the command executed and the other rows show on which storaged processes the task is running. The last row shows the number of successful tasks related to the job.
Status	Shows the status of the job or task. The last row shows the number of failed tasks related to the job. For more information, see Job status.
Start Time	Shows a timestamp indicating the time when the job or task enters the RUNNING phase. The last row shows the number of ongoing tasks related to the job.
Stop Time	Shows a timestamp indicating the time when the job or task gets $\ensuremath{FINISHED}$ , $\ensuremath{FAILED}$ , or $\ensuremath{STOPPED}$ .
Error Code	The error code of job.

### Job status

The descriptions are as follows.

Status	Description
QUEUE	The job or task is waiting in a queue. The Start Time is empty in this phase.
RUNNING	The job or task is running. The Start Time shows the beginning time of this phase.
FINISHED	The job or task is successfully finished. The Stop Time shows the time when the job or task enters this phase.
FAILED	The job or task has failed. The Stop Time shows the time when the job or task enters this phase.
STOPPED	The job or task is stopped without running. The Stop Time shows the time when the job or task enters this phase.
REMOVED	The job or task is removed.

The description of switching the status is described as follows.

### 4.16.7 SHOW JOBS

The  $\ensuremath{\mathsf{SHOW}}$  JOBS statement lists all the unexpired jobs in the current graph space.

The default job expiration interval is one week. You can change it by modifying the job\_expired\_secs parameter of the Meta Service. For how to modify job\_expired\_secs, see Meta Service configuration.

For example:

nebula> SHOW JOBS; +	+	+	++
Job Id   Command	Status	Start Time	Stop Time
++	+	+	++
34   "STATS"	"FINISHED"	2021-11-01T03:32:27.000000	2021-11-01T03:32:27.000000
33   "FLUSH"	"FINISHED"	2021-11-01T03:32:15.000000	2021-11-01T03:32:15.000000
32   "COMPACT"	"FINISHED"	2021-11-01T03:32:06.000000	2021-11-01T03:32:06.000000
31   "REBUILD_TAG_INDEX"	"FINISHED"	2021-10-29T05:39:16.000000	2021-10-29T05:39:17.000000
10   "COMPACT"	"FINISHED"	2021-10-26T02:27:05.000000	2021-10-26T02:27:05.000000
++	+	+	++

### 4.16.8 STOP JOB

The STOP JOB <job\_id> statement stops jobs that are not finished in the current graph space.

### For example:

nebula>	STOP	JOB	22;
+			F
Resul	t		
+			F
"Job s	stoppe	ed"	
+			F

### 4.16.9 RECOVER JOB

The RECOVER JOB  $[<job_id>]$  statement re-executes the jobs that status is FAILED or STOPPED in the current graph space and returns the number of recovered jobs. If  $<job_id>$  is not specified, re-execution is performed from the earliest job and the number of jobs that have been recovered is returned.

For example:

nebula> RECOVER JOB; +-----+ | Recovered job num | +-----+ | 5 job recovered | +-----+

# 4.16.10 FAQ

### How to troubleshoot job problems?

The SUBMIT JOB operations use the HTTP port. Please check if the HTTP ports on the machines where the Storage Service is running are working well. You can use the following command to debug.

curl "http://{storaged-ip}:19779/admin?space={space\_name}&op=compact"

# 5. Deploy and install

# 5.1 Prepare resources for compiling, installing, and running NebulaGraph

This topic describes the requirements and suggestions for compiling and installing NebulaGraph, as well as how to estimate the resource you need to reserve for running a NebulaGraph cluster.

### 5.1.1 About storage devices

NebulaGraph is designed and implemented for NVMe SSD. All default parameters are optimized for the SSD devices and require extremely high IOPS and low latency.

- Due to the poor IOPS capability and long random seek latency, HDD is not recommended. Users may encounter many problems when using HDD.
- Do not use remote storage devices, such as NAS or SAN. Do not connect an external virtual hard disk based on HDFS or Ceph.
- RAID is not recommended because NebulaGraph provides a multi-replica mechanism. Configuring RAID would result in a waste of resources.
- Use local SSD devices, or AWS Provisioned IOPS SSD equivalence.

### 5.1.2 About CPU architecture

Starting with 3.0.2, you can run containerized NebulaGraph databases on Docker Desktop for ARM macOS or on ARM Linux servers.

### Caution

We do not recommend you deploy NebulaGraph on Docker Desktop for Windows due to its subpar performance. For details, see #12401.

### 5.1.3 Requirements for compiling the source code

### Hardware requirements for compiling NebulaGraph

Item	Requirement
CPU architecture	x86_64
Memory	4 GB
Disk	10 GB, SSD

### Supported operating systems for compiling NebulaGraph

For now, we can only compile NebulaGraph in the Linux system. We recommend that you use any Linux system with kernel version 4.15 or above.

# Note

To install NebulaGraph on Linux systems with kernel version lower than required, use RPM/DEB packages or TAR files.

# Software requirements for compiling NebulaGraph

You must have the correct version of the software listed below to compile NebulaGraph. If they are not as required or you are not sure, follow the steps in Prepare software for compiling NebulaGraph to get them ready.

Software	Version	Note
glibc	2.17 or above	You can run lddversion to check the glibc version.
make	Any stable version	-
m4	Any stable version	-
git	Any stable version	-
wget	Any stable version	-
unzip	Any stable version	-
XZ	Any stable version	-
readline-devel	Any stable version	-
ncurses-devel	Any stable version	-
zlib-devel	Any stable version	-
g++	8.5.0 or above	You can run gcc -v to check the gcc version.
cmake	3.14.0 or above	You can run cmakeversion to check the cmake version.
curl	Any stable version	-
redhat-lsb-core	Any stable version	-
libstdc++-static	Any stable version	Only needed in CentOS 8+, RedHat 8+, and Fedora systems.
libasan	Any stable version	Only needed in CentOS 8+, RedHat 8+, and Fedora systems.
bzip2	Any stable version	-

Other third-party software will be automatically downloaded and installed to the build directory at the configure (cmake) stage.

### Prepare software for compiling NebulaGraph

If part of the dependencies are missing or the versions does not meet the requirements, manually install them with the following steps. You can skip unnecessary dependencies or steps according to your needs.

- 1. Install dependencies.
- For CentOS, RedHat, and Fedora users, run the following commands.

```
$ yum update
$ yum install -y make \
                m4 \
                git 🔪
                wget
                unzip 🔪
                xz \
                readline-devel \
                 ncurses-devel \
                zlib-devel
                gcc \
                 gcc-c++ \
                cmake \
                curl \
                 redhat-lsb-core
                bzip2
  // For CentOS 8+, RedHat 8+, and Fedora, install libstdc++-static and libasan as well
$ yum install -y libstdc++-static libasan
```

• For Debian and Ubuntu users, run the following commands.

```
$ apt-get update
$ apt-get install -y make \
    git \
    wget \
    unzip \
    xz-utils \
    curl \
    lsb-core \
    build-essential \
    libreadline-dev \
    ncurses-dev \
    cmake \
    bzip2
```

2. Check if the GCC and cmake on your host are in the right version. See Software requirements for compiling NebulaGraph for the required versions.

\$ g++ --version \$ cmake --version

If your GCC and CMake are in the right versions, then you are all set and you can ignore the subsequent steps. If they are not, select and perform the needed steps as follows.

- 3. If the CMake version is incorrect, visit the CMake official website to install the required version.
- 4. If the G++ version is incorrect, visit the G++ official website or follow the instructions below to to install the required version.
- For CentOS users, run:

```
yum install centos-release-scl
yum install devtoolset-11
scl enable devtoolset-11 'bash
```

• For Ubuntu users, run:

```
add-apt-repository ppa:ubuntu-toolchain-r/test
apt install gcc-11 g++-11
```

### 5.1.4 Requirements and suggestions for installing NebulaGraph in test environments

### Hardware requirements for test environments

Item	Requirement
CPU architecture	x86_64
Number of CPU core	4
Memory	8 GB
Disk	100 GB, SSD

### Supported operating systems for test environments

For now, we can only install NebulaGraph in the Linux system. To install NebulaGraph in a test environment, we recommend that you use any Linux system with kernel version 3.9 or above.

### Suggested service architecture for test environments

Process	Suggested number
metad (the metadata service process)	1
storaged (the storage service process)	1 or more
graphd (the query engine service process)	1 or more

For example, for a single-machine test environment, you can deploy 1 metad, 1 storaged, and 1 graphd processes in the machine.

For a more common test environment, such as a cluster of 3 machines (named as A, B, and C), you can deploy NebulaGraph as follows:

Machine name	Number of metad	Number of storaged	Number of graphd
А	1	1	1
В	None	1	1
С	None	1	1

### 5.1.5 Requirements and suggestions for installing NebulaGraph in production environments

### Hardware requirements for production environments

Item	Requirement
CPU architecture	x86_64
Number of CPU core	48
Memory	256 GB
Disk	2 * 1.6 TB, NVMe SSD

### Supported operating systems for production environments

For now, we can only install NebulaGraph in the Linux system. To install NebulaGraph in a production environment, we recommend that you use any Linux system with kernel version 3.9 or above.

Users can adjust some of the kernel parameters to better accommodate the need for running NebulaGraph. For more information, see kernel configuration.

### Suggested service architecture for production environments

Danger
DO NOT deploy a single cluster across IDCs (The Enterprise Editon supports data synchronization between clusters across IDCs).

Process	Suggested number
metad (the metadata service process)	3
storaged (the storage service process)	3 or more
graphd (the query engine service process)	3 or more

Each metad process automatically creates and maintains a replica of the metadata. Usually, you need to deploy three metad processes and only three.

The number of storaged processes does not affect the number of graph space replicas.

Users can deploy multiple processes on a single machine. For example, on a cluster of 5 machines (named as A, B, C, D, and E), you can deploy NebulaGraph as follows:

l Number of graphd
1
1
1
1
1

### 5.1.6 Capacity requirements for running a NebulaGraph cluster

Users can estimate the memory, disk space, and partition number needed for a NebulaGraph cluster of 3 replicas as follows.

Resource	Unit	How to estimate	Description
Disk space for a cluster	Bytes	the_sum_of_edge_number_and_vertex_number * average_bytes_of_properties * 7.5 * 120%	For more information, see Edge partitioning and storage amplification.
Memory for a cluster	Bytes	<pre>[ the_sum_of_edge_number_and_vertex_number * 16 + the_number_of_RocksDB_instances * ( write_buffer_size * max_write_buffer_number ) + rocksdb_block_cache ] * 120%</pre>	write_buffer_size and max_write_buffer_number are RocksDB parameters. For more information, see MemTable. For details about rocksdb_block_cache, see Memory usage in RocksDB.
Number of partitions for a graph space	-	<pre>the_number_of_disks_in_the_cluster * disk_partition_num_multiplier</pre>	disk_partition_num_multiplier is an integer between 2 and 20 (both including). Its value depends on the disk performance. Use 20 for SSD and 2 for HDD.

• Question 1: Why do I need to multiply by 7.5 in the disk space estimation formula?

Answer: On one hand, the data in one single replica takes up about 2.5 times more space than that of the original data file (csv) according to test values. On the other hand, indexes take up additional space. Each indexed vertex or edge takes up 16 bytes of memory. The hard disk space occupied by the index can be empirically estimated as the total number of indexed vertices or edges \* 50 bytes.

• Question 2: Why do we multiply the disk space and memory by 120%?

Answer: The extra 20% is for buffer.

• Question 3: How to get the number of RocksDB instances?

Answer: Each graph space corresponds to one RocksDB instance and each directory in the --data\_path item in the etc/nebulastoraged.conf file corresponds to one RocksDB instance. That is, the number of RocksDB instances = the number of directories \* the number of graph spaces.

### Q Note

Users can decrease the memory size occupied by the bloom filter by adding --enable\_partitioned\_index\_filter=true in etc/nebulastoraged.conf. But it may decrease the read performance in some random-seek cases.

# Caution

Each RocksDB instance takes up about 70M of disk space even when no data has been written yet. One partition corresponds to one RocksDB instance, and when the partition setting is very large, for example, 100, the graph space takes up a lot of disk space after it is created.

Last update: March 26, 2024

# 5.2 Compile and install

### 5.2.1 Install NebulaGraph by compiling the source code

Installing NebulaGraph from the source code allows you to customize the compiling and installation settings and test the latest features.

### Prerequisites

• Users have to prepare correct resources described in Prepare resources for compiling, installing, and running NebulaGraph.

Q Note

Compilation of NebulaGraph offline is not currently supported.

• The host to be installed with NebulaGraph has access to the Internet.

### Installation steps

- 1. Use Git to clone the source code of NebulaGraph to the host.
- [Recommended] To install NebulaGraph 3.6.0, run the following command.

\$ git clone --branch release-3.6 https://github.com/vesoft-inc/nebula.git

• To install the latest developing release, run the following command to clone the source code from the master branch.

\$ git clone https://github.com/vesoft-inc/nebula.git

2. Go to the nebula/third-party directory, and run the install-third-party.sh script to install the third-party libraries.

```
$ cd nebula/third-party
$ ./install-third-party.sh
```

3. Go back to the nebula directory, create a directory named build, and enter the directory.

\$ cd .. \$ mkdir build && cd build

 ${\small 4. \ Generate \ Makefile \ with \ CMake.}$ 

### O Note

The installation path is /usr/local/nebula by default. To customize it, add the -DCMAKE\_INSTALL\_PREFIX=<installation\_path> CMake variable in the following command.

For more information about CMake variables, see CMake variables.

\$ cmake -DCMAKE\_INSTALL\_PREFIX=/usr/local/nebula -DENABLE\_TESTING=OFF -DCMAKE\_BUILD\_TYPE=Release ...

### 5. Compile NebulaGraph.

### Q Note

Check Prepare resources for compiling, installing, and running NebulaGraph.

To speed up the compiling, use the -j option to set a concurrent number  $\mathbb{N}$ . It should be  $((\min(\text{CPU core number}), \text{CPU core number}))$ .

**\$ make -j**{N} # E.g., make -j2

### 6. Install NebulaGraph.

\$ sudo make install

# Note

The configuration files in the etc/ directory (/usr/local/nebula/etc by default) are references. Users can create their own configuration files accordingly. If you want to use the scripts in the script directory to start, stop, restart, and kill the service, and check the service status, the configuration files have to be named as nebula-graph.conf, nebula-metad.conf, and nebula-storaged.conf.

### Update the master branch

The source code of the master branch changes frequently. If the corresponding NebulaGraph release is installed, update it in the following steps.

1. In the nebula directory, run git pull upstream master to update the source code.

2. In the nebula/build directory, run make -j{N} and make install again.

### Next to do

Manage NebulaGraph services

### **CMake variables**

USAGE OF CMAKE VARIABLES

\$ cmake -D<variable>=<value> ...

The following CMake variables can be used at the configure (cmake) stage to adjust the compiling settings.

CMAKE\_INSTALL\_PREFIX

CMAKE\_INSTALL\_PREFIX specifies the path where the service modules, scripts, configuration files are installed. The default path is /usr/ local/nebula.

ENABLE\_WERROR

ENABLE\_WERROR is ON by default and it makes all warnings into errors. You can set it to OFF if needed.

ENABLE\_TESTING

ENABLE\_TESTING is ON by default and unit tests are built with the NebulaGraph services. If you just need the service modules, set it to OFF.

### ENABLE\_ASAN

ENABLE\_ASAN is OFF by default and the building of ASan (AddressSanitizer), a memory error detector, is disabled. To enable it, set ENABLE\_ASAN to ON. This variable is intended for NebulaGraph developers.

### CMAKE\_BUILD\_TYPE

NebulaGraph supports the following building types of MAKE\_BUILD\_TYPE :

• Debug

The default value of CMAKE\_BUILD\_TYPE . It indicates building NebulaGraph with the debug info but not the optimization options.

• Release

It indicates building NebulaGraph with the optimization options but not the debug info.

• RelWithDebInfo

It indicates building NebulaGraph with the optimization options and the debug info.

• MinSizeRel

It indicates building NebulaGraph with the optimization options for controlling the code size but not the debug info.

### ENABLE\_INCLUDE\_WHAT\_YOU\_USE

ENABLE\_INCLUDE\_WHAT\_YOU\_USE is OFF by default. When set to ON and include-what-you-use is installed on the system, the system reports redundant headers contained in the project source code during makefile generation.

NEBULA\_USE\_LINKER

Specifies the program linker on the system. The available values are:

- bfd, the default value, indicates that ld.bfd is applied as the linker.
- IIId, indicates that ld.lld, if installed on the system, is applied as the linker.
- gold, indicates that ld.gold, if installed on the system, is applied as the linker.

### CMAKE\_C\_COMPILER/CMAKE\_CXX\_COMPILER

Usually, CMake locates and uses a C/C++ compiler installed in the host automatically. But if your compiler is not installed at the standard path, or if you want to use a different one, run the command as follows to specify the installation path of the target compiler:

\$ cmake -DCMAKE\_C\_COMPILER=<path\_to\_gcc/bin/gcc> -DCMAKE\_CXX\_COMPILER=<path\_to\_gcc/bin/g++> .. \$ cmake -DCMAKE\_C\_COMPILER=<path\_to\_clang/bin/clang> -DCMAKE\_CXX\_COMPILER=<path\_to\_clang/bin/clang++> .

### ENABLE\_CCACHE

ENABLE\_CCACHE is ON by default and Ccache (compiler cache) is used to speed up the compiling of NebulaGraph.

To disable ccache, setting ENABLE\_CCACHE to OFF is not enough. On some platforms, the ccache installation hooks up or precedes the compiler. In such a case, you have to set an environment variable export CCACHE\_DISABLE=true or add a line disable=true in ~/.ccache/ ccache.conf as well. For more information, see the ccache official documentation.

### NEBULA\_THIRDPARTY\_ROOT

NEBULA\_THIRDPARTY\_ROOT specifies the path where the third party software is installed. By default it is /opt/vesoft/third-party.

### Examine problems

If the compiling fails, we suggest you:

1. Check whether the operating system release meets the requirements and whether the memory and hard disk space are sufficient.

- 2. Check whether the third-party is installed correctly.
- 3. Use make -j1 to reduce the compiling concurrency.

Last update: November 14, 2023

### 5.2.2 Compile NebulaGraph using Docker

NebulaGraph's source code is written in C++. Compiling NebulaGraph requires certain dependencies which might conflict with host system dependencies, potentially causing compilation failures. Docker offers a solution to this. NebulaGraph provides a Docker image containing the complete compilation environment, ensuring an efficient build process and avoiding host OS conflicts. This guide outlines the steps to compile NebulaGraph using Docker.

### Prerequisites

Before you begin:

- 1. Docker: Ensure Docker is installed on your system.
- 2. Clone NebulaGraph's Source Code: Clone the repository locally using:

```
git clone --branch release-3.6 https://github.com/vesoft-inc/nebula.git
```

This clones the NebulaGraph source code to a subdirectory named nebula.

### Compilation steps

1. Pull the NebulaGraph compilation image.

```
docker pull vesoft/nebula-dev:ubuntu2004
```

Here, we use the official NebulaGraph compilation image, ubuntu2004. For different versions, see nebula-dev-docker.

2. Start the compilation container.

```
docker run -ti \
    --security-opt seccomp=unconfined \
    -v "$PWD":/home \
    -w /home \
    -name nebula_dev \
    vesoft/nebula-dev:ubuntu2004 \
    bash
```

- --security-opt seccomp=unconfined : Disables the seccomp security mechanism to avoid compilation errors.
- -v "\$PWD":/home : Mounts the local path of the NebulaGraph code to the container's /home directory.
- -w /home : Sets the container's working directory to /home . Any command run inside the container will use this directory as the current directory.
- --name nebula\_dev : Assigns a name to the container, making it easier to manage and operate.
- vesoft/nebula-dev:ubuntu2004: Uses the ubuntu2004 version of the vesoft/nebula-dev compilation image.
- bash : Executes the bash command inside the container, entering the container's interactive terminal.

After executing this command, you'll enter an interactive terminal inside the container. To re-enter the container, use docker exec -ti nebula\_dev bash .

# $_{\mbox{3.}}$ Compile NebulaGraph inside the container.

### a. Enter the NebulaGraph source code directory.

cd nebula

b. Create a build directory and enter it.

mkdir build && cd build

c. Use CMake to generate the Makefile.

cmake -DCMAKE\_CXX\_COMPILER=\$TOOLSET\_CLANG\_DIR/bin/g++ -DCMAKE\_C\_COMPILER=\$TOOLSET\_CLANG\_DIR/bin/gcc -DENABLE\_WERROR=OFF -DCMAKE\_BUILD\_TYPE=Debug -DENABLE\_TESTING=OFF .

For more on CMake, see CMake Parameters.

### d. Compile NebulaGraph.

```
# The -j parameter specifies the number of threads to use. 
# If you have a multi-core CPU, you can use more threads to speed up compilation. make -j2
```

Compilation might take some time based on your system performance.

### 4. Install the Executables and Libraries.

Post successful compilation, NebulaGraph's binaries and libraries are located in /home/nebula/build. Install them to /usr/locat/nebula:

make install

 $Once \ completed, \ NebulaGraph \ is \ compiled \ and \ installed \ in \ the \ host \ directory \ /usr/local/nebula \ .$ 

### Next Steps

- Start NebulaGraph Service
- Connect to NebulaGraph

# 5.3 Local single-node installation

# 5.3.1 Install NebulaGraph with RPM or DEB package

RPM and DEB are common package formats on Linux systems. This topic shows how to quickly install NebulaGraph with the RPM or DEB package.

# Note

The console is not complied or packaged with NebulaGraph server binaries. You can install nebula-console by yourself.

### Prerequisites

• The tool wget is installed.

### Step 1: Download the package from cloud service

# NebulaGraph is currently only supported for installation on Linux systems, and only CentOS 7.x, CentOS 8.x, Ubuntu 16.04, Ubuntu 18.04, and Ubuntu 20.04 operating systems are supported. • Download the released version. URL: //Centos 7 https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.el7.x86\_64.rpm

https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.el8.x86\_64.rpm

### //Ubuntu **1604**

 $https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1604.amd64.deb$ 

### //Ubuntu **1804**

 $https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1804.amd64.deb$ 

### //Ubuntu 2004

https://oss-cdn.nebula-graph.io/package/<release\_version>/nebula-graph-<release\_version>.ubuntu2004.amd64.deb

For example, download the release package 3.6.0 for Centos 7.5:

wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.el7.x86\_64.rpm
wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.el7.x86\_64.rpm.sha256sum.txt

### Download the release package 3.6.0 for Ubuntu 1804:

wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.ubuntu1804.amd64.deb wget https://oss-cdn.nebula-graph.io/package/3.6.0/nebula-graph-3.6.0.ubuntu1804.amd64.deb.sha256sum.txt

### • Download the nightly version.

# Danger

- Nightly versions are usually used to test new features. Do not use it in a production environment.
- Nightly versions may not be built successfully every night. And the names may change from day to day.

URL:



### For example, download the Centos 7.5 package developed and built in 2021.11.28:

wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.el7.x86\_64.rpm
wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.el7.x86\_64.rpm.sha256sum.txt

### For example, download the Ubuntu 1804 package developed and built in 2021.11.28:

wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.ubuntu1804.amd64.deb wget https://oss-cdn.nebula-graph.io/package/nightly/2021.11.28/nebula-graph-2021.11.28-nightly.ubuntu1804.amd64.deb.sha256sum.txt

### Step 2: Install NebulaGraph

• Use the following syntax to install with an RPM package.

\$ sudo rpm -ivh --prefix=<installation\_path> <package\_name>

The option --prefix indicates the installation path. The default path is /usr/local/nebula/.

For example, to install an RPM package in the default path for the 3.6.0 version, run the following command.

sudo rpm -ivh nebula-graph-3.6.0.el7.x86\_64.rpm

• Use the following syntax to install with a DEB package.

\$ sudo dpkg -i <package\_name>

# Note

 $Customizing \ the \ installation \ path \ is \ not \ supported \ when \ installing \ NebulaGraph \ with \ a \ DEB \ package. \ The \ default \ installation \ path \ is \ /usr/local/nebula/ \ .$ 

For example, to install a DEB package for the 3.6.0 version, run the following command.

sudo dpkg -i nebula-graph-3.6.0.ubuntu1804.amd64.deb

Q Note

The default installation path is  $\mbox{/usr/local/nebula/}$  .

### Next to do

- Start NebulaGraph
- Connect to NebulaGraph

### 5.3.2 Install NebulaGraph graph with the tar.gz file

You can install NebulaGraph by downloading the tar.gz file.

Note
• NebulaGraph provides installing with the tar.gz file starting from version 2.6.0.

• NebulaGraph is currently only supported for installation on Linux systems, and only CentOS 7.x, CentOS 8.x, Ubuntu 16.04, Ubuntu 18.04, and Ubuntu 20.04 operating systems are supported.

### Installation steps

1. Download the NebulaGraph tar.gz file using the following address.

Before downloading, you need to replace <release\_version> with the version you want to download.

//Centos 7

. https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.el7.x86\_64.tar.gz //Checksum

https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.el7.x86\_64.tar.gz.sha256sum.txt

//Centos 8

https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.el8.x86\_64.tar.gz //Checksum

https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.el8.x86\_64.tar.gz.sha256sum.txt

//Ubuntu 1604

https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1604.amd64.tar.gz.sha256sum.txt

//Ubuntu **1804** 

https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1804.amd64.tar.gz //Checksum

 $https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.ubuntu1804.amd64.tar.gz.sha256sum.txt$ 

# //Ubuntu 2004 https://oss-co //Checksum

 $https://oss-cdn.nebula-graph.com.cn/package/<release_version>/nebula-graph-<release_version>.ubuntu2004.amd64.tar.gz$ 

https://oss-cdn.nebula-graph.com.cn/package/<release\_version>/nebula-graph-<release\_version>.ubuntu2004.amd64.tar.gz.sha256sum.txt

For example, to download the NebulaGraph release-3.6 tar.gz file for CentoS 7.5, run the following command:

wget https://oss-cdn.nebula-graph.com.cn/package/3.6.0/nebula-graph-3.6.0.el7.x86\_64.tar.gz

### 2. Decompress the tar.gz file to the NebulaGraph installation directory.

tar -xvzf <tar.gz\_file\_name> -C <install\_path>

- tar.gz\_file\_name specifies the name of the tar.gz file.
- install\_path specifies the installation path.

For example:

tar -xvzf nebula-graph-3.6.0.el7.x86\_64.tar.gz -C /home/joe/nebula/install

3. Modify the name of the configuration file.

Enter the decompressed directory, rename the files nebula-graphd.conf.default, nebula-metad.conf.default, and nebula-storaged.conf.default in the subdirectory etc, and delete .default to apply the default configuration of NebulaGraph.

Note

To modify the configuration, see Configurations.

So far, you have installed NebulaGraph successfully.

# Next to do

Manage NebulaGraph services

### 5.3.3 Standalone NebulaGraph

Standalone NebulaGraph merges the Meta, Storage, and Graph services into a single process deployed on a single machine. This topic introduces scenarios, deployment steps, etc. of standalone NebulaGraph.

# Banger

Do not use standalone NebulaGraph in production environments.

### Background

The traditional NebulaGraph consists of three services, each service having executable binary files and the corresponding process. Processes communicate with each other by RPC. In standalone NebulaGraph, the three processes corresponding to the three services are combined into one process. For more information about NebulaGraph, see Architecture overview.

### Scenarios

Small data sizes and low availability requirements. For example, test environments that are limited by the number of machines, scenarios that are only used to verify functionality.

### Limitations

- Single service instance per machine.
- High availability and reliability not supported.

### **Resource requirements**

For information about the resource requirements for standalone NebulaGraph, see Software requirements for compiling NebulaGraph.

### Steps

Currently, you can only install standalone NebulaGraph with the source code. The steps are similar to those of the multi-process NebulaGraph. You only need to modify the step **Generate Makefile with CMake** by adding -DENABLE\_STANDALONE\_VERSION=on to the command. For example:

cmake -DCMAKE\_INSTALL\_PREFIX=/usr/local/nebula -DENABLE\_TESTING=OFF -DENABLE\_STANDALONE\_VERSION=on -DCMAKE\_BUILD\_TYPE=Release ..

For more information about installation details, see Install NebulaGraph by compiling the source code.

After installing standalone NebulaGraph, see the topic connect to Service to connect to NebulaGraph databases.

### Configuration file

The path to the configuration file for standalone NebulaGraph is /usr/local/nebula/etc by default.

You can run sudo cat nebula-standalone.conf.default to see the file content. The parameters and the corresponding descriptions in the file are generally the same as the configurations for multi-process NebulaGraph except for the following parameters.

Parameter	Predefined value	Description
meta_port	9559	The port number of the Meta service.
storage_port	9779	The port number of the Storage Service.
meta_data_path	data/meta	The path to Meta data.

You can run commands to check configurable parameters and the corresponding descriptions. For details, see Configurations.

# 5.4 Deploy a NebulaGraph cluster with RPM/DEB package on multiple servers

For now, NebulaGraph does not provide an official deployment tool. Users can deploy a NebulaGraph cluster with RPM or DEB package manually. This topic provides an example of deploying a NebulaGraph cluster on multiple servers (machines).

### 5.4.1 Deployment

Machine name	IP address	Number of graphd	Number of storaged	Number of metad
А	192.168.10.111	1	1	1
В	192.168.10.112	1	1	1
С	192.168.10.113	1	1	1
D	192.168.10.114	1	1	None
E	192.168.10.115	1	1	None

### 5.4.2 Prerequisites

- Prepare 5 machines for deploying the cluster.
- Use the NTP service to synchronize time in the cluster.

### 5.4.3 Manual deployment process

### Install NebulaGraph

Install NebulaGraph on each machine in the cluster. Available approaches of installation are as follows.

- Install NebulaGraph with RPM or DEB package
- Install NebulaGraph by compiling the source code

### Modify the configurations

To deploy NebulaGraph according to your requirements, you have to modify the configuration files.

All the configuration files for NebulaGraph, including nebula-graphd.conf, nebula-metad.conf, and nebula-storaged.conf, are stored in the etc directory in the installation path. You only need to modify the configuration for the corresponding service on the machines. The configurations that need to be modified for each machine are as follows.

Machine name	The configuration to be modified
А	nebula-graphd.conf, nebula-storaged.conf, nebula-metad.conf
В	nebula-graphd.conf, nebula-storaged.conf, nebula-metad.conf
С	nebula-graphd.conf, nebula-storaged.conf, nebula-metad.conf
D	nebula-graphd.conf , nebula-storaged.conf
Е	nebula-graphd.conf , nebula-storaged.conf

Users can refer to the content of the following configurations, which only show part of the cluster settings. The hidden content uses the default setting so that users can better understand the relationship between the servers in the NebulaGraph cluster.

# Q Note

The main configuration to be modified is meta\_server\_addrs. All configurations need to fill in the IP addresses and ports of all Meta services. At the same time, local\_ip needs to be modified as the network IP address of the machine itself. For detailed descriptions of the configuration parameters, see:

• Meta Service configurations

- Graph Service configurations
- Storage Service configurations

### • Deploy machine A

### nebula-graphd.conf

########## networking ########## # Comma separated Meta Server Addresses --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559 # Local IP used to identify the nebula-graphd process. # Change it to an address other than loopback if the service is distributed or # will be accessed remotely. Local IP used to ID accessed remotely. --local\_ip=<mark>192</mark>.168.10.111 # Network device to listen on --listen\_netdev=any # Port to listen on
--port=9669

### nebula-storaged.conf

# Comma separated Meta server addresses

- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559 # Local IP used to identify the nebula-storaged process.
- # Change it to an address other than loopback if the service is distributed or

# will be accessed remotely.
--local\_ip=192.168.10.111

- # Storage daemon listening port
  --port=9779

### nebula-metad.conf

# Comma separated Meta Server addresses

- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
- # Local IP used to identify the nebula-metad process. # Change it to an address other than loopback if the service is distributed or
- # will be accessed remotely.
  --local\_ip=192.168.10.111
- # Meta daemon listening port

--port=<mark>9559</mark>

## • Deploy machine B

### nebula-graphd.conf

- # Comma separated Meta Server Addresses --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559 # Local IP used to identify the nebula-graphd process. # Change it to an address other than loopback if the service is distributed or # will be accessed remotely. Local i=109.169 100 112

- --local\_ip=192.168.10.112
  # Network device to listen on
  --listen\_netdev=any
- # Port to listen on
  --port=9669

### nebula-storaged.conf

- # Comma separated Meta server addresses
- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
  # Local LP used to identify the nebula-storaged process.
  # Change it to an address other than loopback if the service is distributed or
  # will be accessed remotely.
  --local\_ip=192.168.10.112
  # Change address is the service is distributed or
  # will be accessed remotely.
  --local\_ip=192.168.10.112
  # Change address is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # will be accessed remotely.
  --local\_ip=192.168.10.112
  # Change address is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # will be accessed remotely.
  --local\_ip=192.168.10.112
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is distributed or
  # total doesnot listen in a service
  # Change is the service is the service
  # Change is the service is the service
  # Change is the service
  # Change is the service is

- # Storage daemon listening port --port=9779

### • nebula-metad.conf

- ########## networking ########## # Comma separated Meta Server addresses
- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559 # Local IP used to identify the nebula-metad process. # Change it to an address other than loopback if the service is distributed or

- # will be accessed remotely.
  --local\_ip=192.168.10.112
- # Meta daemon listening port
  --port=9559
## . Deploy machine C

#### nebula-graphd.conf

- ########## networking ##########
- # Comma separated Meta Server Addresses
  --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
- # Local IP used to identify the nebula-graphd process. # Change it to an address other than loopback if the service is distributed or # will be accessed remotely.

- --local\_ip=192.168.10.113 # Network device to listen on
- --listen\_netdev=any
- # Port to listen on
- --port=9669

#### nebula-storaged.conf

- ########## networking ##########
- # Comma separated Meta server addresses
- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
  # Local IP used to identify the nebula-storaged process.
  # Change it to an address other than loopback if the service is distributed or

- # will be accessed remotely.
  --local\_ip=192.168.10.113
- # Storage daemon listening port --port=9779

#### nebula-metad.conf

- # Comma separated Meta Server addresses
- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
- # Local IP used to identify the nebula-metad process. # Change it to an address other than loopback if the service is distributed or
- # will be accessed remotely.
  --local\_ip=192.168.10.113
- # Meta daemon listening port
- --port=9559

#### • Deploy machine D

#### nebula-graphd.conf

########## networking ##########

- # Comma separated Meta Server Addresses
- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559
- # Local IP used to identify the nebula-graphd process. # Change it to an address other than loopback if the service is distributed or
- # will be accessed remotely.
- --local\_ip=192.168.10.114 # Network device to listen on
- --listen\_netdev=any # Port to listen on
- --port=9669

#### nebula-storaged.conf

- --local\_ip=<mark>192</mark>.168.10.114
- # Storage daemon listening port --port=9779

## Deploy machine E

#### nebula-graphd.conf

	######### networking ##########
	# Comma separated Meta Server Addresses
	<pre>meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559 # Local IP used to identify the nebula-graphd process. # Change it to an address other than loopback if the service is distributed or # will be accessed remotelylocal_ip=192.168.10.115 # Network device to listen onlisten_netdev=any # Port to listen onport=9669</pre>
n	nebula-storaged.conf
	########## networking ######### # Comma separated Meta server addresses meta_server_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:9559

- --meta\_server\_addrs=192.168.10.111:9559,192.168.10.112:9559,192.168.10.113:955
  # Local IP used to identify the nebula-storaged process.
  # Change it to an address other than loopback if the service is distributed or
  # will be accessed remotely.
  --local\_ip=192.168.10.115

- # Storage daemon listening port
  --port=9779

#### Start the cluster

Start the corresponding service on **each machine**. Descriptions are as follows.

Machine name	The process to be started
А	graphd, storaged, metad
В	graphd, storaged, metad
С	graphd, storaged, metad
D	graphd, storaged
Е	graphd, storaged

The command to start the NebulaGraph services is as follows.

sudo /usr/local/nebula/scripts/nebula.service start <metad|graphd|storaged|all>

## Q Note

• Make sure all the processes of services on each machine are started. Otherwise, you will fail to start NebulaGraph.

• When the graphd process, the storaged process, and the metad process are all started, you can use all instead.

• /usr/tocat/nebuta is the default installation path for NebulaGraph. Use the actual path if you have customized the path. For more information about how to start and stop the services, see Manage NebulaGraph services.

#### Check the cluster status

Install the native CLI client NebulaGraph Console, then connect to any machine that has started the graphd process, run ADD HOSTS command to add storage hosts, and run SHOW HOSTS to check the cluster status. For example:

\$ ./nebula-console --addr 192.168.10.111 --port 9669 -u root -p nebula

2021/05/25 01:41:19 [INFO] connection pool is initialized successfully Welcome to NebulaGraph!

<sup>&</sup>gt; ADD HOSTS 192.168.10.111:9779, 192.168.10.112:9779, 192.168.10.113:9779, 192.168.10.114:9779, 192.168.10.115:9779;

<pre>&gt; SHOW HOSTS;</pre>							
+	++	Statuc	++	Loador distribution	+	lictribution	++
+	+4		+		+		++
"192.168.10.111"	9779	"ONLINE"	0	"No valid partition"	"No valid p	partition"	"3.6.0"
"192.168.10.112"	9779	"ONLINE"	0	"No valid partition"	"No valid p	partition"	"3.6.0"
"192.168.10.113"	9779	"ONLINE"	0	"No valid partition"	"No valid p	partition"	3.6.0"
"192.168.10.114"	9779	"ONLINE"	0	"No valid partition"	"No valid p	partition"	3.6.0
"192.168.10.115"	9779	"ONLINE"	0	"No valid partition"	"No valid p	partition"	3.6.0

Last update: October 25, 2023

## 5.5 Deploy NebulaGraph with Docker Compose

Using Docker Compose can quickly deploy NebulaGraph services based on the prepared configuration file. It is only recommended to use this method when testing functions of NebulaGraph.

#### 5.5.1 Prerequisites

• You have installed the following applications on your host.

Application	<b>Recommended version</b>	Official installation reference
Docker	Latest	Install Docker Engine
Docker Compose	Latest	Install Docker Compose
Git	Latest	Download Git

• If you are deploying NebulaGraph as a non-root user, grant the user with Docker-related privileges. For detailed instructions, see Manage Docker as a non-root user.

- You have started the Docker service on your host.
- If you have already deployed another version of NebulaGraph with Docker Compose on your host, to avoid compatibility issues, you need to delete the nebula-docker-compose/data directory.

#### 5.5.2 Deploy NebulaGraph

1. Clone the 3.6.0 branch of the nebula-docker-compose repository to your host with Git.

## Danger

The master branch contains the untested code for the latest NebulaGraph development release. **DO NOT** use this release in a production environment.

```
$ git clone -b release-3.6 https://github.com/vesoft-inc/nebula-docker-compose.git
```

# Note

The x.y version of Docker Compose aligns to the x.y version of NebulaGraph. For the NebulaGraph z version, Docker Compose does not publish the corresponding z version, but pulls the z version of the NebulaGraph image.

#### 2. Go to the nebula-docker-compose directory.

\$ cd nebula-docker-compose/

3. Run the following command to start all the NebulaGraph services.

#### Q Note

• Update the NebulaGraph images and NebulaGraph Console images first if they are out of date.

• The return result after executing the command varies depending on the installation directory.

<sup>[</sup>nebula-docker-compose]\$ docker-compose up -d Creating nebula-docker-compose\_metad0\_1 ... done Creating nebula-docker-compose\_metad2\_1 ... done Creating nebula-docker-compose\_graphd2\_1 ... done

Creating nebula-docker-compose\_graphd\_1 ... done Creating nebula-docker-compose\_graphdl\_1 ... done Creating nebula-docker-compose\_storaged0\_1 ... done Creating nebula-docker-compose\_storaged2\_1 ... done Creating nebula-docker-compose\_storaged1\_1 ... done

# Empatibility

Starting from NebulaGraph version 3.1.0, nebula-docker-compose automatically starts a NebulaGraph Console docker container and adds the storage host to the cluster (i.e. ADD HOSTS command).

#### Q Note

For more information of the preceding services, see NebulaGraph architecture.

#### 5.5.3 Connect to NebulaGraph

There are two ways to connect to NebulaGraph:

- Connected with Nebula Console outside the container. Because the external mapping port for the Graph service is also fixed as 9669 in the container's configuration file, you can connect directly through the default port. For details, see Connect to NebulaGraph.
- Log into the container installed NebulaGraph Console, then connect to the Graph service. This section describes this approach.
- 1. Run the following command to view the name of NebulaGraph Console docker container.

nebula-docker-compose_console_1         sh -c for i in `seq 1 60`; Up         Up         0.0.0.0:32847->155669/tcp,:::32847->155669/tcp, 19669/tcp,           nebula-docker-compose_graphdl_1         /usr/local/nebula/bin/nebu Up         Up (healthy)         0.0.0.0:32846->19670/tcp, :::32847->155669/tcp, 19669/tcp,           0.0.0.0:32846->19670/tcp,         0.0.0.0:32846->19670/tcp,         0.0.0.0:32846->19670/tcp,	\$ docker-compose ps Name	Command	State	Ports
	nebula-docker-compose_console_1 nebula-docker-compose_graphdl_1	sh -c for i in `seq 1 60`; /usr/local/nebula/bin/nebu	Up Up (healthy)	0.0.0.0:32847->15669/tcp,:::32847->15669/tcp, 19669/tcp, 0.0.0.0:32846->19670/tcp,:::32846->19670/tcp, 0.0.0.0:32849->5669/tcp,:::32849->5669/tcp, 9669/tcp

#### Q Note

nebula-docker-compose\_console\_1 and nebula-docker-compose\_graphd1\_1 are the container names of NebulaGraph Console and Graph Service respectively.

2. Run the following command to enter the NebulaGraph Console docker container.

docker exec -it nebula-docker-compose\_console\_1 /bin/sh
/ #

3. Connect to NebulaGraph with NebulaGraph Console.

```
/ # ./usr/local/bin/nebula-console -u <user_name> -p <password> --address=graphd --port=9669
```

#### Q Note

By default, the authentication is off, you can only log in with an existing username (the default is root ) and any password. To turn it on, see Enable authentication.

#### 4. Run the following commands to view the cluster state.

nebula> SHOW HO	)STS;					L
Host	Port	Status	Leader count	Leader distribution	Partition distribution	Version
"storaged0"   "storaged1"   "storaged2"	9779   9779   9779	"ONLINE" "ONLINE" "ONLINE"	0 0 0	"No valid partition" "No valid partition" "No valid partition"	"No valid partition"   "No valid partition"   "No valid partition"	"3.6.0"     "3.6.0"     "3.6.0"

Run exit twice to switch back to your terminal (shell).

#### 5.5.4 Check the NebulaGraph service status and ports

Run docker-compose ps to list all the services of NebulaGraph and their status and ports.

## Note

NebulaGraph provides services to the clients through port 9669 by default. To use other ports, modify the docker-compose.yaml file in the nebula-docker-compose directory and restart the NebulaGraph services.

\$ docker-compose ps			
nebula-docker-compose_console_1	sh -c sleep 3 &&	Up	
1	nebula-co		
nebula-docker-compose_graphd1_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49174->19669/tcp,:::49174->19669/tcp, 0.0.0.0:49171->19670/tcp,:::49171->19670/tcp, 0.0.0.0:49177->9669/
tcp,:::49177->9669/tcp			
nebula-docker-compose_graphd2_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49175->19669/tcp,:::49175->19669/tcp, 0.0.0.0:49172->19670/tcp,:::49172->19670/tcp, 0.0.0.0:49178->9669/
tcp,:::49178->9669/tcp			
nebula-docker-compose_graphd_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49180->19669/tcp,:::49180->19669/tcp, 0.0.0.0:49179->19670/tcp,:::49179->19670/tcp, 0.0.0.0:9669->9669/
tcp,:::9669->9669/tcp			
nebula-docker-compose_metad0_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49157->19559/tcp,:::49157->19559/tcp, 0.0.0.0:49154->19560/tcp,:::49154->19560/tcp, 0.0.0.0:49160->9559/
tcp,:::49160->9559/tcp, 9560/tcp			
nebula-docker-compose_metad1_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49156->19559/tcp,:::49156->19559/tcp, 0.0.0.0:49153->19560/tcp,:::49153->19560/tcp, 0.0.0.0:49159->9559/
tcp,:::49159->9559/tcp, 9560/tcp			
nebula-docker-compose_metad2_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49158->19559/tcp,:::49158->19559/tcp, 0.0.0.0:49155->19560/tcp,:::49155->19560/tcp, 0.0.0.0:49161->9559/
tcp,:::49161->9559/tcp, 9560/tcp			
nebula-docker-compose_storaged0_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49166->19779/tcp,:::49166->19779/tcp, 0.0.0.0:49163->19780/tcp,:::49163->19780/tcp, 9777/tcp, 9778/tcp,
0.0.0.0:49169->9779/tcp,:::49169->9	9779/tcp, <mark>9780</mark> /tcp		
nebula-docker-compose_storaged1_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49165->19779/tcp,:::49165->19779/tcp, 0.0.0.0:49162->19780/tcp,:::49162->19780/tcp, 9777/tcp, 9778/tcp,
0.0.0.0:49168->9779/tcp,:::49168->9	9779/tcp, <mark>9780</mark> /tcp		
nebula-docker-compose_storaged2_1	/usr/local/nebula/bin/nebu	Up	0.0.0.0:49167->19779/tcp,:::49167->19779/tcp, 0.0.0.0:49164->19780/tcp,:::49164->19780/tcp, 9777/tcp, 9778/tcp,
0.0.0.0:49170->9779/tcp,:::49170->9	9779/tcp, 9780/tcp		

If the service is abnormal, you can first confirm the abnormal container name (such as nebula-docker-compose\_graphd2\_1) and then log in to the container and troubleshoot.

\$ docker exec -it nebula-docker-compose\_graphd2\_1 bash

## 5.5.5 Check the service data and logs

All the data and logs of NebulaGraph are stored persistently in the nebula-docker-compose/data and nebula-docker-compose/logs directories.

The structure of the directories is as follows:



## 5.5.6 Modify configurations

The configuration file of Docker Compose is nebula-docker-compose/docker-compose.yaml. To make the new configuration take effect, modify the configuration in this file and restart the service.

The configurations in the docker-compose.yaml file overwrite the configurations in the configuration file (/usr/local/nebula/etc) of the containered NebulaGraph service. Therefore, you can modify the configurations in the docker-compose.yaml file to customize the configurations of the NebulaGraph service.

For more instructions, see Configurations.

#### 5.5.7 Restart NebulaGraph services

To restart all the NebulaGraph services, run the following command:

\$ docker-comp	iose restart	
Restarting ne	bula-docker-compose_console_1	 done
Restarting ne	bula-docker-compose_graphd_1	 done
Restarting ne	bula-docker-compose_graphd1_1	 done
Restarting ne	bula-docker-compose_graphd2_1	 done
Restarting ne	bula-docker-compose_storaged1_1	 done
Restarting ne	bula-docker-compose-storaged0_1	 done
Restarting ne	bula-docker-compose_storaged2_1	 done

Restarting nebula-docker-compose\_metad1\_1 ... done Restarting nebula-docker-compose\_metad2\_1 ... done Restarting nebula-docker-compose\_metad0\_1 ... done

To restart multiple services, such as graphd and storaged0, run the following command:

\$ docker-compose restart graphd storaged0
Restarting nebula-docker-compose\_graphd\_1 ... done
Restarting nebula-docker-compose\_storaged0\_1 ... done

#### 5.5.8 Stop and remove NebulaGraph services

You can stop and remove all the NebulaGraph services by running the following command:

**D**anger

This command stops and removes all the containers of the NebulaGraph services and the related network. If you define volumes in the docker-compose.yaml, the related data are retained.

The command docker-compose down -v removes all the local data. Try this command if you are using the nightly release and having some compatibility issues.

\$ docker-compose down

The following information indicates you have successfully stopped the NebulaGraph services:

... done Stopping nebula-docker-compose\_console\_1 ... done Stopping nebula-docker-compose\_graphd1\_1 Stopping nebula-docker-compose\_graphd\_1 ... done Stopping nebula-docker-compose\_graphd2\_1 done Stopping nebula-docker-compose storaged1 1 done Stopping nebula-docker-compose\_storaged0\_1 ... done Stopping nebula-docker-compose\_storaged2\_1 ... done Stopping nebula-docker-compose metad2 1 done Stopping nebula-docker-compose\_metad0\_1 done Stopping nebula-docker-compose\_metad1\_1 done Removing nebula-docker-compose\_console\_1 ... done Removing nebula-docker-compose\_graphd1\_1 done Removing nebula-docker-compose\_graphd\_1 Removing nebula-docker-compose\_graphd2\_1 done done Removing nebula-docker-compose\_storaged1\_1 done Removing nebula-docker-compose\_storaged0\_1 ... done Removing nebula-docker-compose\_storaged2\_1 ... done Removing nebula-docker-compose\_metad2\_1 done Removing nebula-docker-compose metad0 1 ... done Removing nebula-docker-compose\_metad1\_1 done Removing network nebula-docker-compose\_nebula-net

#### 5.5.9 FAQ

#### How to fix the docker mapping to external ports?

To set the ports of corresponding services as fixed mapping, modify the docker-compose.yaml in the nebula-docker-compose directory. For example:

```
graphd:
    image: vesoft/nebula-graphd:release-3.6
    ...
    ports:
        - 9669:9669
        - 19669
        - 19670
```

9669:9669 indicates the internal port 9669 is uniformly mapped to external ports, while 19669 indicates the internal port 19669 is randomly mapped to external ports.

#### How to upgrade or update the docker images of NebulaGraph services?

1. In the nebula-docker-compose/docker-compose.yaml file, change all the image values to the required image version.

- 2. In the nebula-docker-compose directory, run docker-compose pull to update the images of the Graph Service, Storage Service, Meta Service, and NebulaGraph Console.
- 3. Run docker-compose up -d to start the NebulaGraph services again.
- 4. After connecting to NebulaGraph with NebulaGraph Console, run SHOW HOSTS GRAPH, SHOW HOSTS STORAGE, or SHOW HOSTS META to check the version of the responding service respectively.

#### ERROR: toomanyrequests when docker-compose pull

You may meet the following error.

ERROR: toomanyrequests: You have reached your pull rate limit. You may increase the limit by authenticating and upgrading: https://www.docker.com/increaserate-limit.

You have met the rate limit of Docker Hub. Learn more on Understanding Docker Hub Rate Limiting.

#### How to update the NebulaGraph Console client?

The command docker-compose pull updates both the NebulaGraph services and the NebulaGraph Console.

#### How to activate storaged containers when they remain in offline status?

The activation script for storaged containers in Docker Compose may fail to run in rare cases. You can connect to NebulaGraph with NebulaGraph Console or NebulaGraph Studio and then manually run the ADD HOSTS command to activate them by adding the storaged containers to the cluster. An example of the command is as follows:

nebula> ADD HOSTS "storaged0":9779,"storaged1":9779,"storaged2":9779

#### 5.5.10 Related documents

- Install and deploy NebulaGraph with the source code
- Install NebulaGraph by RPM or DEB
- Connect to NebulaGraph

Last update: April 26, 2024

# 5.6 Deploy NebulaGraph with NebulaGraph Lite

Using NebulaGraph Lite can quickly deploy NebulaGraph and start experiencing NebulaGraph in just five minutes. It is ideal for ad-hoc development and learning NebulaGraph.

## 5.6.1 Benefits

- Quick installation of NebulaGraph Lite through the Python package management tool.
- NebulaGraph Lite supports the deployment of NebulaGraph with non-root permission.
- NebulaGraph Lite supports the deployment of NebulaGraph in containers or any Jupyter Notebook platform on Linux-based systems.

## 5.6.2 Steps

1. Run the following statement to install NebulaGraph Lite.

pip3 install nebulagraph-lite

- 2. Start NebulaGraph Lite. NebulaGraph Lite automatically deploys and starts a single-node NebulaGraph service, and imports a test dataset.
- Start from Jupyter Notebook

```
from nebulagraph_lite import nebulagraph_let as ng_let n = ng_let() n.start()
```

#### • Start from the command line

nebulagraph start

The following result is returned indicating that the startup and import of the test dataset was successful.



## 5.6.3 What's next

- $\bullet$  Connect to NebulaGraph with NebulaGraph Jupyter Extension.
- Connect to NebulaGraph with NebulaGraph Console.

Last update: April 15, 2024

# 5.7 Install NebulaGraph with ecosystem tools

You can install the NebulaGraph Community Edition with the following ecosystem tools:

• NebulaGraph Operator

#### 5.7.1 Installation details

• To install NebulaGraph with NebulaGraph Operator, see Install NebulaGraph clusters.

Last update: November 15, 2023

# 5.8 Manage NebulaGraph Service

NebulaGraph supports managing services with scripts.

## 5.8.1 Manage services with script

You can use the nebula.service script to start, stop, restart, terminate, and check the NebulaGraph services.

Note					
nebula.service is stored in the environment.	/usr/local/nebula/scripts	directory by default.	If you have customized	the path, use the actual pa	th in your

#### Syntax

```
$ sudo /usr/local/nebula/scripts/nebula.service
[-v] [-c <config_file_path>]
<start | stop | restart | kill | status>
<metad | graphd | storaged | all>
```

Parameter	Description
- V	Display detailed debugging information.
-c	Specify the configuration file path. The default path is $\mbox{/usr/local/nebula/etc/}$ .
start	Start the target services.
stop	Stop the target services.
restart	Restart the target services.
kill	Terminate the target services.
status	Check the status of the target services.
metad	Set the Meta Service as the target service.
graphd	Set the Graph Service as the target service.
storaged	Set the Storage Service as the target service.
all	Set all the NebulaGraph services as the target services.

#### 5.8.2 Start NebulaGraph

Run the following command to start NebulaGraph.

```
$ sudo /usr/local/nebula/scripts/nebula.service start all
[INF0] Starting nebula-metad...
[INF0] Done
[INF0] Starting nebula-graphd...
[INF0] Starting nebula-storaged...
[INF0] Done
```

## 5.8.3 Stop NebulaGraph

## **D**anger

Do not run kill -9 to forcibly terminate the processes. Otherwise, there is a low probability of data loss.

Run the following command to stop NebulaGraph.

```
$ sudo /usr/local/nebula/scripts/nebula.service stop all
[INF0] Stopping nebula-metad...
[INF0] Done
[INF0] Stopping nebula-graphd...
[INF0] Done
[INF0] Stopping nebula-storaged...
[INF0] Done
```

#### 5.8.4 Check the service status

Run the following command to check the service status of NebulaGraph.

\$ sudo /usr/local/nebula/scripts/nebula.service status all

• NebulaGraph is running normally if the following information is returned.

```
INFO] nebula-metad(33fd35e): Running as 29020, Listening on 9559
[INFO] nebula-graphd(33fd35e): Running as 29095, Listening on 9669
[WARN] nebula-storaged after v3.0.0 will not start service until it is added to cluster.
[WARN] See Manage Storage hosts:ADD HOSTS in https://docs.nebula-graph.io/
[INFO] nebula-storaged(33fd35e): Running as 29147, Listening on 9779
```

Q Note

After starting NebulaGraph, the port of the nebula-storaged process is shown in red. Because the nebula-storaged process waits for the nebula-metad to add the current Storage service during the startup process. The Storage works after it receives the ready signal. Starting from NebulaGraph 3.0.0, the Meta service cannot directly read or write data in the Storage service that you add in the configuration file. The configuration file only registers the Storage service to the Meta service. You must run the ADD HOSTS command to enable the Meta to read and write data in the Storage service. For more information, see Manage Storage hosts.

- If the returned result is similar to the following one, there is a problem. You may also go to the NebulaGraph community for help.
  - [INFO] nebula-metad: Running as 25600, Listening on 9559 [INFO] nebula-graphd: Exited [INFO] nebula-storaged: Running as 25646, Listening on 9779

The NebulaGraph services consist of the Meta Service, Graph Service, and Storage Service. The configuration files for all three services are stored in the /usr/local/nebula/etc/ directory by default. You can check the configuration files according to the returned result to troubleshoot problems.

#### 5.8.5 Next to do

Connect to NebulaGraph

Last update: October 25, 2023

## 5.9 Connect to NebulaGraph

This topic provides basic instruction on how to use the native CLI client NebulaGraph Console to connect to NebulaGraph.

## Caution

When connecting to NebulaGraph for the first time, you must register the Storage Service before querying data.

NebulaGraph supports multiple types of clients, including a CLI client, a GUI client, and clients developed in popular programming languages. For more information, see the client list.

#### 5.9.1 Prerequisites

- You have started NebulaGraph services.
- The machine on which you plan to run NebulaGraph Console has network access to the Graph Service of NebulaGraph.
- The NebulaGraph Console version is compatible with the NebulaGraph version.

#### Q Note

NebulaGraph Console and NebulaGraph of the same version number are the most compatible. There may be compatibility issues when connecting to NebulaGraph with a different version of NebulaGraph Console. The error message incompatible version between client and server is displayed when there is such an issue.

#### Steps

1. On the NebulaGraph Console releases page, select a NebulaGraph Console version and click Assets.

Note					
it is recommended to select the <b>latest</b> version.					

- 2. In the **Assets** area, find the correct binary file for the machine where you want to run NebulaGraph Console and download the file to the machine.
- 3. (Optional) Rename the binary file to nebula-console for convenience.

## Note

For Windows, rename the file to nebula-console.exe.

4. On the machine to run NebulaGraph Console, grant the execute permission of the nebula-console binary file to the user.

Note	
or Windows, skip this step.	

\$ chmod 111 nebula-console

5. In the command line interface, change the working directory to the one where the nebula-console binary file is stored.

- $_{6.}$  Run the following command to connect to NebulaGraph.
- For Linux or macOS:

```
$ ./nebula-console -addr <ip> -port <port> -u <username> -p <password>
[-t 120] [-e "nGQL_statement" | -f filename.nGQL]
```

#### • For Windows:

> nebula-console.exe -addr <ip> -port <port> -u <username> -p <password> [-t 120] [-e "nGQL\_statement" | -f filename.nGQL]

## Parameter descriptions are as follows:

Parameter	Description
-h/-help	Shows the help menu.
-addr/-address	Sets the IP (or hostname) of the Graph service. The default address is 127.0.0.1.
-P/-port	Sets the port number of the graphd service. The default port number is 9669.
-u/-user	Sets the username of your NebulaGraph account. Before enabling authentication, you can use any existing username. The default username is root.
-p/-password	Sets the password of your NebulaGraph account. Before enabling authentication, you can use any characters as the password.
-t/-timeout	Sets an integer-type timeout threshold of the connection. The unit is millisecond. The default value is 120.
-e/-eval	Sets a string-type nGQL statement. The nGQL statement is executed once the connection succeeds. The connection stops after the result is returned.
-f/-file	Sets the path of an nGQL file. The nGQL statements in the file are executed once the connection succeeds. The result will be returned and the connection stops then.
-enable_ssl	Enables SSL encryption when connecting to NebulaGraph.
-ssl_root_ca_path	Sets the storage path of the certification authority file.
-ssl_cert_path	Sets the storage path of the certificate file.
- ssl_private_key_path	Sets the storage path of the private key file.

For information on more parameters, see the project repository.

Last update: October 25, 2023

## 5.10 Manage Storage hosts

Starting from NebulaGraph 3.0.0, setting Storage hosts in the configuration files only registers the hosts on the Meta side, but does not add them into the cluster. You must run the ADD HOSTS statement to add the Storage hosts.

## Note

 $NebulaGraph\ Cloud\ clusters\ add\ Storage\ hosts\ automatically.\ Cloud\ users\ do\ not\ need\ to\ manually\ run\ \ ADD\ HoSTS\ .$ 

#### 5.10.1 Prerequisites

• You have connected to the NebulaGraph database.

#### 5.10.2 Add Storage hosts

Add the Storage hosts to a NebulaGraph cluster.

```
nebula> ADD HOSTS <ip>:<port> [,<ip>:<port> ...];
nebula> ADD HOSTS "<hostname>":<port> [,"<hostname>":<port> ...];
```

#### Q Note

• To make sure the follow-up operations work as expected, wait for two heartbeat cycles, i.e., 20 seconds, and then run SHOW HOSTS to check whether the host is online.

• Make sure that the IP address and port number are the same as those in the configuration file. For example, the default IP address and port number in standalone deployment are 127.0.0.1:9779.

• When using a domain name, enclose it in quotation marks, for example, ADD HOSTS "foo-bar":9779.

• Ensure that the storage host to be added is not used by any other cluster, otherwise, the storage adding operation will fail.

#### 5.10.3 Drop Storage hosts

Delete the Storage hosts from cluster.

## Note

You can not delete an in-use Storage host directly. Delete the associated graph space before deleting the Storage host.

nebula> DROP HOSTS <ip>:<port> [,<ip>::<port> ...]; nebula> DROP HOSTS "<hostname>":<port> [,"<hostname>":<port> ...];

#### 5.10.4 View Storage hosts

#### View the Storage hosts in the cluster.

nebula> SHOW HOSTS STORAGE;				
Host   Port   Status	Role	Git Info Sha   Version		
++++++	+	++		
"storaged0"   9779   "ONLINE"	"STORAGE"	"3ba41bd"  "3.6.0"		
"storaged1"   9779   "ONLINE"	"STORAGE"	"3ba41bd"   "3.6.0"		
"storaged2"   9779   "UNLINE"	"STURAGE"	"3Da41Dd"   "3.6.0"		

Last update: October 25, 2023

# 5.11 Upgrade NebulaGraph to 3.6.0

This topic describes how to upgrade NebulaGraph from version 2.x and 3.x to 3.6.0, taking upgrading from version 2.6.1 to 3.6.0 as an example.

#### 5.11.1 Applicable source versions

This topic applies to upgrading NebulaGraph from 2.5.0 and later 2.x, and 3.x versions to 3.6.0. It does not apply to historical versions earlier than 2.5.0, including the 1.x versions.

To upgrade NebulaGraph from historical versions to 3.6.0:

1. Upgrade it to the latest 2.5 version according to the docs of that version.

#### 2. Follow this topic to upgrade it to 3.6.0.

# Caution

To upgrade NebulaGraph from versions earlier than 2.0.0 (including the 1.x versions) to 3.6.0, you need to find the date\_time\_zonespec.csv in the share/resources directory of 3.6.0 files, and then copy it to the same directory in the NebulaGraph installation path.

#### 5.11.2 Limitations

- Rolling Upgrade is not supported. You must stop all the NebulaGraph services before the upgrade.
- There is no upgrade script. You have to manually upgrade each server in the cluster.
- This topic does not apply to scenarios where NebulaGraph is deployed with Docker, including Docker Swarm, Docker Compose, and K8s.
- You must upgrade the old NebulaGraph services on the same machines they are deployed. **DO NOT** change the IP addresses, configuration files of the machines, and **DO NOT** change the cluster topology.
- Known issues that could cause data loss are listed on GitHub known issues. The issues are all related to altering schema or default values.
- DO NOT use soft links to switch the data directories.
- You must have the sudo privileges to complete the steps in this topic.

## 5.11.3 Upgrade influences

• Client compatibility

After the upgrade, you will not be able to connect to NebulaGraph from old clients. You will need to upgrade all clients to a version compatible with NebulaGraph 3.6.0.

• Configuration changes

A few configuration parameters have been changed. For more information, see the release notes and configuration docs.

• nGQL compatibility

The nGQL syntax is partially incompatible:

- Disable the YIELD clause to return custom variables.
- The YIELD clause is required in the FETCH, GO, LOOKUP, FIND PATH and GET SUBGRAPH statements.
- It is required to specify a tag to query properties of a vertex in a MATCH statement. For example, from return v.name to return v.player.name.
- Full-text indexes

Before upgrading a NebulaGraph cluster with full-text indexes deployed, you must manually delete the full-text indexes in Elasticsearch, and then run the SIGN IN command to log into ES and recreate the indexes after the upgrade is complete. To manually delete the full-text indexes in Elasticsearch, you can use the curl command curl -XDELETE -u <es\_username>:<es\_password> '<es\_access\_ip>:<port>/<fullindex\_name>' , for example, curl -XDELETE -u elastic:elastic 'http://192.168.8.xxx:9200/nebula\_index\_2534' . If no username and password are set for Elasticsearch, you can omit the -u <es\_username>:<es\_password> part.

## Caution

There may be other undiscovered influences. Before the upgrade, we recommend that you read the release notes and user manual carefully, and keep an eye on the posts on the forum and issues on Github.

#### 5.11.4 Preparations before the upgrade

• Download the package of NebulaGraph 3.6.0 according to your operating system and system architecture. You need the binary files during the upgrade. Find the package on the download page.

## Note

You can also get the new binaries from the source code or the RPM/DEB package.

• Locate the data files based on the value of the data\_path parameters in the Storage and Meta configurations, and backup the data files. The default paths are nebula/data/storage and nebula/data/meta.

## **D**anger

The old data will not be automatically backed up during the upgrade. You must manually back up the data to avoid data loss.

- Backup the configuration files.
- Collect the statistics of all graph spaces before the upgrade. After the upgrade, you can collect again and compare the results to make sure that no data is lost. To collect the statistics:
- a. Run SUBMIT JOB STATS.
- b. Run SHOW JOBS and record the result.

#### 5.11.5 Upgrade steps

#### 1. Stop all NebulaGraph services.

<nebula\_install\_path>/scripts/nebula.service stop all

nebula\_install\_path indicates the installation path of NebulaGraph.

The storaged progress needs around 1 minute to flush data. You can run nebula.service status all to check if all services are stopped. For more information about starting and stopping services, see Manage services.

#### ) Note

If the services are not fully stopped in 20 minutes, stop upgrading and ask for help on the forum or Github.

## Caution

Starting from version 3.0.0, it is possible to insert vertices without tags. If you need to keep vertices without tags, add --graph\_use\_vertex\_key=true in the configuration file ( nebula-graphd.conf ) of all Graph services within the cluster; and add --use\_vertex\_key=true in the configuration file ( nebula-storaged.conf ) of all Storage services."

2. In the target path where you unpacked the package, use the binaries in the bin directory to replace the old binaries in the bin directory in the NebulaGraph installation path.

O Note

Update the binary of the corresponding service on each NebulaGraph server.

- 3. Modify the following parameters in all Graph configuration files to accommodate the value range of the new version. If the parameter values are within the specified range, skip this step.
- Set a value in [1,604800] for session\_idle\_timeout\_secs. The recommended value is 28800.
- Set a value in [1,604800] for client\_idle\_timeout\_secs. The recommended value is 28800.

The default values of these parameters in the 2.x versions are not within the range of the new version. If you do not change the default values, the upgrade will fail. For detailed parameter description, see Graph Service Configuration.

4. Start all Meta services.

<nebula\_install\_path>/scripts/nebula-metad.service start

Once started, the Meta services take several seconds to elect a leader.

To verify that Meta services are all started, you can start any Graph server, connect to it through NebulaGraph Console, and run SHOW HOSTS meta and SHOW META LEADER. If the status of Meta services are correctly returned, the services are successfully started.

#### Q Note

If the operation fails, stop the upgrade and ask for help on the forum or GitHub.

5. Start all the Graph and Storage services.

## Note

If the operation fails, stop the upgrade and ask for help on the forum or GitHub.

6. Connect to the new version of NebulaGraph to verify that services are available and data are complete. For how to connect, see Connect to NebulaGraph.

Currently, there is no official way to check whether the upgrade is successful. You can run the following reference statements to test the upgrade:

nebula> SHOW HOSTS; nebula> SHOW HOSTS storage; nebula> SHOW SPACES; nebula> USE <space\_name> nebula> SHOW PARTS; nebula> SUBWIT JOB STATS; nebula> SHOW STATS; nebula> SHOW STATS; nebula> SHOW STATS;

You can also test against new features in version 3.6.0.

#### 5.11.6 Upgrade failure and rollback

If the upgrade fails, stop all NebulaGraph services of the new version, recover the old configuration files and binaries, and start the services of the old version.

All NebulaGraph clients in use must be switched to the old version.

#### 5.11.7 FAQ

#### Can I write through the client during the upgrade?

A: No. You must stop all NebulaGraph services during the upgrade.

#### The Space 0 not found warning message during the upgrade process

When the Space 0 not found warning message appears during the upgrade process, you can ignore it. The space 0 is used to store meta information about the Storage service and does not contain user data, so it will not affect the upgrade.

#### How to upgrade if a machine has only the Graph Service, but not the Storage Service?

A: You only need to update the configuration files and binaries of the Graph Service.

#### How to resolve the error Permission denied?

A: Try again with the sudo privileges.

#### Is there any change in gflags?

A: Yes. For more information, see the release notes and configuration docs.

#### Is there a tool or solution for verifying data consistency after the upgrade?

A: No. But if you only want to check the number of vertices and edges, run SUBMIT JOB STATS and SHOW STATS after the upgrade, and compare the result with the result that you recorded before the upgrade.

#### How to solve the issue that Storage is OFFLINE and Leader count is 0?

A: Run the following statement to add the Storage hosts into the cluster manually.

ADD HOSTS <ip>:<port>[, <ip>:<port> ...];

#### For example:

ADD HOSTS 192.168.10.100:9779, 192.168.10.101:9779, 192.168.10.102:9779;

If the issue persists, ask for help on the forum or GitHub.

## Why the job type changed after the upgrade, but job ID remains the same?

A: SHOW JOBS depends on an internal ID to identify job types, but in NebulaGraph 2.5.0 the internal ID changed in this pull request, so this issue happens after upgrading from a version earlier than 2.5.0.

Last update: October 25, 2023

## 5.12 Uninstall NebulaGraph

This topic describes how to uninstall NebulaGraph.

## Caution

Before re-installing NebulaGraph on a machine, follow this topic to completely uninstall the old NebulaGraph, in case the remaining data interferes with the new services, including inconsistencies between Meta services.

#### 5.12.1 Prerequisite

The NebulaGraph services should be stopped before the uninstallation. For more information, see Manage NebulaGraph services.

#### 5.12.2 Step 1: Delete data files of the Storage and Meta Services

If you have modified the data\_path in the configuration files for the Meta Service and Storage Service, the directories where NebulaGraph stores data may not be in the installation path of NebulaGraph. Check the configuration files to confirm the data paths, and then manually delete the directories to clear all data.

## Note

For a NebulaGraph cluster, delete the data files of all Storage and Meta servers.

#### 1. Check the Storage Service disk settings. For example:

2. Check the Metad Service configurations and find the corresponding metadata directories.

3. Delete the data and the directories found in step 2.

## 5.12.3 Step 2: Delete the installation directories



The default installation path is /usr/local/nebula , which is specified by --prefix while installing NebulaGraph.

#### Uninstall NebulaGraph deployed with source code

Find the installation directories of NebulaGraph, and delete them all.

#### Uninstall NebulaGraph deployed with RPM packages

1. Run the following command to get the NebulaGraph version.

#### \$ rpm -qa | grep "nebula"

The return message is as follows.

nebula-graph-3.6.0-1.x86\_64

2. Run the following command to uninstall NebulaGraph.

sudo rpm -e <nebula\_version>

#### For example:

sudo rpm -e nebula-graph-3.6.0-1.x86\_64

3. Delete the installation directories.

#### Uninstall NebulaGraph deployed with DEB packages

1. Run the following command to get the NebulaGraph version.

\$ dpkg -l | grep "nebula"

The return message is as follows.

ii nebula-graph 3.6.0 amd64 NebulaGraph Package built using CMake

2. Run the following command to uninstall NebulaGraph.

sudo dpkg -r <nebula\_version>

#### For example:

sudo dpkg -r nebula-graph

3. Delete the installation directories.

## Uninstall NebulaGraph deployed with Docker Compose

1. In the nebula-docker-compose directory, run the following command to stop the NebulaGraph services.

docker-compose down -v

2. Delete the nebula-docker-compose directory.

Last update: November 3, 2023

# 6. Configure and log

## 6.1 Configurations

## 6.1.1 Configurations

NebulaGraph builds the configurations based on the gflags repository. Most configurations are flags. When the NebulaGraph service starts, it will get the configuration information from Configuration files by default. Configurations that are not in the file apply the default values.

Q Note

• Because there are many configurations and they may change as NebulaGraph develops, this topic will not introduce all configurations. To get detailed descriptions of configurations, follow the instructions below.

• It is not recommended to modify the configurations that are not introduced in this topic, unless you are familiar with the source code and fully understand the function of configurations.

# L<sub>Jacy</sub> version compatibility

In the topic of 1.x, we provide a method of using the CONFIGS command to modify the configurations in the cache. However, using this method in a production environment can easily cause inconsistencies of configurations between clusters and the local. Therefore, this method will no longer be introduced starting with version 2.x.

#### Get the configuration list and descriptions

Use the following command to get all the configuration information of the service corresponding to the binary file:

<br/>dinary> --help

#### For example:

```
# Get the help information from Meta
$ /usr/local/nebula/bin/nebula-metad --help
```

# Get the help information from Graph

\$ /usr/local/nebula/bin/nebula-graphd --help

# Get the help information from Storage
\$ /usr/local/nebula/bin/nebula-storaged --help

The above examples use the default storage path /usr/local/nebula/bin/. If you modify the installation path of NebulaGraph, use the actual path to query the configurations.

#### Get configurations

Use the curl command to get the value of the running configurations.

For example:

```
# Get the running configurations from Meta
curl 127.0.0.1:19559/flags
# Get the running configurations from Graph
curl 127.0.0.1:19669/flags
```

# Get the running configurations from Storage
curl 127.0.0.1:19779/flags

Utilizing the -s or `-silent option allows for the concealment of the progress bar and error messages. For example:

curl -s 127.0.0.1:19559/flags

#### Q Note

In an actual environment, use the real IP (or hostname) instead of 127.0.0.1 in the above example.

#### **Configuration files**

CONFIGURATION FILES FOR CLUSTERS INSTALLED FROM SOURCE, WITH AN RPM/DEB PACKAGE, OR A TAR PACKAGE

NebulaGraph provides two initial configuration files for each service, <service\_name>.conf.default and <service\_name>.conf.default and <service\_name>.conf.production. You can use them in different scenarios conveniently. For clusters installed from source and with a RPM/DEB package, the default path is /usr/local/nebula/etc/. For clusters installed with a TAR package, the path is <install\_path>/<tar\_package\_directory>/etc.

The configuration values in the initial configuration file are for reference only and can be adjusted according to actual needs. To use the initial configuration file, choose one of the above two files and delete the suffix .default or .production to make it valid.

#### Q Note

To ensure the availability of services, it is recommended that configurations for the same service be consistent, except for <code>local\_ip</code>. For example, three Storage servers are deployed in one NebulaGraph cluster. The configurations of the three Storage servers are recommended to be consistent, except for <code>local\_ip</code>.

The initial configuration files corresponding to each service are as follows.

NebulaGraph service	Initial configuration file	Description
Meta	${\tt nebula-metad.conf.default} \ {\tt and} \ {\tt nebula-metad.conf.production}$	Meta service configuration
Graph	${\tt nebula-graphd.conf.default} \ {\tt and} \ {\tt nebula-graphd.conf.production}$	Graph service configuration
Storage	nebula-storaged.conf.default ${\ and \ }$ nebula-storaged.conf.production	Storage service configuration

Each initial configuration file of all services contains local\_config. The default value is true, which means that the NebulaGraph service will get configurations from its configuration files and start it.

# Caution

It is not recommended to modify the value of local\_config to false. If modified, the NebulaGraph service will first read the cached configurations, which may cause configuration inconsistencies between clusters and cause unknown risks.

CONFIGURATION FILES FOR CLUSTERS INSTALLED WITH DOCKER COMPOSE

For clusters installed with Docker Compose, the configuration file's default installation path of the cluster is <install\_path>/nebuladocker-compose/docker-compose.yaml. The parameters in the command field of the file are the launch parameters for each service.

CONFIGURATION FILES FOR CLUSTERS INSTALLED WITH NEBULAGRAPH OPERATOR

For clusters installed with Kubectl through NebulaGraph Operator, the configuration file's path is the path of the cluster YAML file. You can modify the configuration of each service through the spec.{graphd|storaged|metad}.config parameter.

#### Q Note

The services cannot be configured for clusters installed with Helm.

#### Modify configurations

You can modify the configurations of NebulaGraph in the configuration file or use commands to dynamically modify configurations.

## Caution

Using both methods to modify the configuration can cause the configuration information to be managed inconsistently, which may result in confusion. It is recommended to only use the configuration file to manage the configuration, or to make the same modifications to the configuration file after dynamically updating the configuration through commands to ensure consistency.

MODIFYING CONFIGURATIONS IN THE CONFIGURATION FILE

By default, each NebulaGraph service gets configured from its configuration files. You can modify configurations and make them valid according to the following steps:

- For clusters installed from source, with a RPM/DEB, or a TAR package
- a. Use a text editor to modify the configuration files of the target service and save the modification.
- b. Choose an appropriate time to restart all NebulaGraph services to make the modifications valid.
- For clusters installed with Docker Compose
- a. In the <install\_path>/nebula-docker-compose/docker-compose.yaml file, modify the configurations of the target service.
- b. In the nebula-docker-compose directory, run the command docker-compose up -d to restart the service involving configuration modifications.
- For clusters installed with Kubectl

For details, see Customize configuration parameters for a NebulaGraph cluster.

DYNAMICALLY MODIFYING CONFIGURATIONS USING COMMAND

You can dynamically modify the configuration of NebulaGraph by using the curl command. For example, to modify the wal\_ttl parameter of the Storage service to 600, use the following command:

curl -X PUT -H "Content-Type: application/json" -d'{"wal\_ttl":"600"}' -s "http://192.168.15.6:19779/flags"

In this command, {"wal\_ttl":"600"} specifies the configuration parameter and its value to be modified, and 192.168.15.6:19779 specifies the IP address and HTTP port number of the Storage service.

## Caution

• The functionality of dynamically modifying configurations is only applicable to prototype verification and testing environments. It is not recommended to use this feature in production environments. This is because when the <code>local\_config</code> value is set to <code>true</code>, the dynamically modified configuration is not persisted, and the configuration will be restored to the initial configuration after the service is restarted.

Only **part of** the configuration parameters can be dynamically modified. For the specific list of parameters that can be modified, see the description of **Whether supports runtime dynamic modifications** in the respective service configuration.

Last update: November 22, 2023

#### 6.1.2 Meta Service configuration

NebulaGraph provides two initial configuration files for the Meta service, nebula-metad.conf.default and nebula-metad.conf.production. Users can use them in different scenarios conveniently. The default file path is /usr/local/nebula/etc/.

Caution	
It is not recommended to modify the value of local_config to false. If modified, the NebulaGraph service will first read the cach configurations, which may cause configuration inconsistencies between clusters and cause unknown risks.	ıed

• It is not recommended to modify the configurations that are not introduced in this topic, unless you are familiar with the source code and fully understand the function of configurations.

#### How to use the configuration files

To use the initial configuration file, choose one of the above two files and delete the suffix .default or .production from the initial configuration file for the Meta Service to apply the configurations defined in it.

#### About parameter values

If a parameter is not set in the configuration file, NebulaGraph uses the default value. Not all parameters are predefined. And the predefined parameters in the two initial configuration files are different. This topic uses the parameters in <code>nebula-metad.conf.default</code>.

## Caution

Some parameter values in the configuration file can be dynamically modified during runtime. We label these parameters as **Yes** that supports runtime dynamic modification in this article. When the <code>local\_config</code> value is set to <code>true</code>, the dynamically modified configuration is not persisted, and the configuration will be restored to the initial configuration after the service is restarted. For more information, see Modify configurations.

For all parameters and their current values, see Configurations.

#### **Basics configurations**

Name	Predefined value	Description	Whether supports runtime dynamic modifications
daemonize	true	When set to true, the process is a daemon process.	No
pid_file	pids/nebula- metad.pid	The file that records the process ID.	No
timezone_name	-	Specifies the NebulaGraph time zone. This parameter is not predefined in the initial configuration files. You can manually set it if you need it. The system default value is UTC+00:00:00. For the format of the parameter value, see Specifying the Time Zone with TZ. For example,timezone_name=UTC+08:00 represents the GMT+8 time zone.	No

#### Q Note

• While inserting property values of time types, NebulaGraph transforms time types (except TIMESTAMP) to the corresponding UTC according to the time zone specified by timezone\_name. The time-type values returned by nGQL queries are all UTC time.

• timezone\_name is only used to transform the data stored in NebulaGraph. Other time-related data of the NebulaGraph processes still uses the default time zone of the host, such as the log printing time.

## Logging configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
log_dir	Logs	The directory that stores the Meta Service log. It is recommended to put logs on a different hard disk from the data.	No
minloglevel	0	Specifies the minimum level of the log. That is, log messages at or above this level. Optional values are 0 (INFO), 1 (WARNING), 2 (ERROR), 3 (FATAL). It is recommended to set it to 0 during debugging and 1 in a production environment. If it is set to 4, NebulaGraph will not print any logs.	Yes
v	0	Specifies the detailed level of VLOG. That is, log all VLOG messages less or equal to the level. Optional values are 0, 1, 2, 3, 4, 5. The VLOG macro provided by glog allows users to define their own numeric logging levels and control verbose messages that are logged with the parameter v. For details, see Verbose Logging.	Yes
logbufsecs	0	Specifies the maximum time to buffer the logs. If there is a timeout, it will output the buffered log to the log file. In means real-time output. This configuration is measured in seconds.	No
redirect_stdout	true	When set to true, the process redirects the stdout and stderr to separate output files.	No
<pre>stdout_log_file</pre>	metad- stdout.log	Specifies the filename for the stdout log.	No
stderr_log_file	metad- stderr.log	Specifies the filename for the stderr log.	No
stderrthreshold	3	Specifies the minloglevel to be copied to the stderr log.	No
<pre>timestamp_in_logfile_name</pre>	true	Specifies if the log file name contains a timestamp. true indicates yes, false indicates no.	No

## Networking configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
meta_server_addrs	127.0.0.1:9559	Specifies the IPs (or hostnames) and ports of all Meta Services. Multiple addresses are separated with commas.	No
local_ip	127.0.0.1	Specifies the local IP (or hostname) for the Meta Service. The local IP address is used to identify the nebula-metad process. If it is a distributed cluster or requires remote access, modify it to the corresponding address.	No
port	9559	Specifies RPC daemon listening port of the Meta service. The neighboring +1 (9560) port is used for Raft communication between Meta services.	No
ws_ip	0.0.0	Specifies the IP address for the HTTP service.	No
ws_http_port	19559	Specifies the port for the HTTP service.	No
ws_storage_http_port	19779	Specifies the Storage service listening port used by the HTTP protocol. It must be consistent with the ws_http_port in the Storage service configuration file. This parameter only applies to standalone NebulaGraph.	No

# Caution

It is recommended to use a real IP when using IP address. Otherwise, 127.0.0.1/0.0.0 cannot be parsed correctly in some cases.

#### Storage configurations

Name	Predefined Value	Description	Whether supports runtime dynamic modifications
data_path	data/meta	The storage path for Meta data.	No

## Misc configurations

Name	Predefin Value	ed Description	Whether supports runtime dynamic modifications
default_parts_num	10	Specifies the default partition number when creating a new graph space.	No
default_replica_factor	1	Specifies the default replica number when creating a new graph space.	No
heartbeat_interval_secs	10	Specifies the default heartbeat interval. Make sure the heartbeat_interval_secs values for all services are the same, otherwise NebulaGrap <b>CANNOT</b> work normally. This configuration is measured in seconds.	Yes h
agent_heartbeat_interval_s	secs 60	Specifies the default heartbeat interval for the Agent service. This configuration influences the time it takes for the system to determine that the Agent service is offline. This configuration measured in seconds.	e No ne Lis
ocksDB options configura	tions		
Name	Predefined Value	Description	Whether supports runtime dynamic modifications
rocksdb_wal_sync	true	Enables or disables RocksDB WAL synchronization. Available values are true (enable) and false (disable).	No

Last update: April 7, 2024

#### 6.1.3 Graph Service configuration

NebulaGraph provides two initial configuration files for the Graph Service, nebula-graphd.conf.default and nebula-graphd.conf.production. Users can use them in different scenarios conveniently. The default file path is /usr/local/nebula/etc/.

Caution
It is not recommended to modify the value of local_config to false. If modified, the NebulaGraph service will first read the cached
configurations, which may cause configuration inconsistencies between clusters and cause unknown risks.

• It is not recommended to modify the configurations that are not introduced in this topic, unless you are familiar with the source code and fully understand the function of configurations.

#### How to use the configuration files

To use the initial configuration file, choose one of the above two files and delete the suffix .default or .production from the initial configuration file for the Meta Service to apply the configurations defined in it.

#### About parameter values

If a parameter is not set in the configuration file, NebulaGraph uses the default value. Not all parameters are predefined. And the predefined parameters in the two initial configuration files are different. This topic uses the parameters in <code>nebula-metad.conf.default</code>.

## Caution

Some parameter values in the configuration file can be dynamically modified during runtime. We label these parameters as **Yes** that supports runtime dynamic modification in this article. When the local\_config value is set to true, the dynamically modified configuration is not persisted, and the configuration will be restored to the initial configuration after the service is restarted. For more information, see Modify configurations.

For all parameters and their current values, see Configurations.

## **Basics configurations**

Name	Predefined value	Description	Whether supports runtime dynamic modifications
daemonize	true	When set to true, the process is a daemon process.	No
pid_file	pids/nebula- graphd.pid	The file that records the process ID.	No
enable_optimizer	true	When set to true, the optimizer is enabled.	No
timezone_name	-	Specifies the NebulaGraph time zone. This parameter is not predefined in the initial configuration files. The system default value is UTC+00:00:00 . For the format of the parameter value, see Specifying the Time Zone with TZ. For example,timezone_name=UTC+08:00 represents the GMT+8 time zone.	No
default_charset	utf8	Specifies the default charset when creating a new graph space.	No
default_collate	utf8_bin	Specifies the default collate when creating a new graph space.	No
local_config	true	When set to true, the process gets configurations from the configuration files.	No

# Note

• While inserting property values of time types, NebulaGraph transforms time types (except TIMESTAMP) to the corresponding UTC according to the time zone specified by timezone\_name. The time-type values returned by nGQL queries are all UTC time.

• timezone\_name is only used to transform the data stored in NebulaGraph. Other time-related data of the NebulaGraph processes still uses the default time zone of the host, such as the log printing time.

## Logging configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
log_dir	logs	The directory that stores the Graph service log. It is recommended to put logs on a different hard disk from the data.	No
minloglevel	0	Specifies the minimum level of the log. That is, log messages at or above this level. Optional values are 0 (INFO), 1 (WARNING), 2 (ERROR), 3 (FATAL). It is recommended to set it to 0 during debugging and 1 in a production environment. If it is set to 4, NebulaGraph will not print any logs.	Yes
v	0	Specifies the detailed level of VLOG. That is, log all VLOG messages less or equal to the level. Optional values are 0, 1, 2, 3, 4, 5. The VLOG macro provided by glog allows users to define their own numeric logging levels and control verbose messages that are logged with the parameter v. For details, see Verbose Logging.	Yes
logbufsecs	0	Specifies the maximum time to buffer the logs. If there is a timeout, it will output the buffered log to the log file. 0 means real-time output. This configuration is measured in seconds.	No
redirect_stdout	true	When set to true, the process redirects the stdout and stderr to separate output files.	No
<pre>stdout_log_file</pre>	graphd- stdout.log	Specifies the filename for the stdout log.	No
stderr_log_file	graphd- stderr.log	Specifies the filename for the stderr log.	No
stderrthreshold	3	Specifies the minloglevel to be copied to the stderr log.	No
timestamp_in_logfile_name	true	Specifies if the log file name contains a timestamp. true indicates yes, false indicates no.	No

## Query configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
accept_partial_success	false	When set to false, the process treats partial success as an error. This configuration only applies to read-only requests. Write requests always treat partial success as an error. A partial success query will prompt Got partial result.	Yes
session_reclaim_interval_secs	60	Specifies the interval that the Session information is sent to the Meta service. This configuration is measured in seconds.	Yes
<pre>max_allowed_query_size</pre>	4194304	Specifies the maximum length of queries. Unit: bytes. The default value is 4194304 , namely 4MB.	Yes

## Networking configurations
Name	Predefined value	Description	Whether supports runtime dynamic modifications
meta_server_addrs	127.0.0.1:9559	Specifies the IPs (or hostnames) and ports of all Meta Services. Multiple addresses are separated with commas.	No
local_ip	127.0.0.1	Specifies the local IP (or hostname) for the Graph Service. The local IP address is used to identify the nebula-graphd process. If it is a distributed cluster or requires remote access, modify it to the corresponding address.	No
listen_netdev	any	Specifies the listening network device.	No
port	9669	Specifies RPC daemon listening port of the Graph service.	No
reuse_port	false	When set to false, the SO_REUSEPORT is closed.	No
listen_backlog	1024	Specifies the maximum length of the connection queue for socket monitoring. This configuration must be modified together with the net.core.somaxconn.	No
client_idle_timeout_secs	28800	Specifies the time to expire an idle connection. The value ranges from 1 to 604800. The default is 8 hours. This configuration is measured in seconds.	No
<pre>session_idle_timeout_secs</pre>	28800	Specifies the time to expire an idle session. The value ranges from 1 to 604800. The default is 8 hours. This configuration is measured in seconds.	No
<pre>num_accept_threads</pre>	1	Specifies the number of threads that accept incoming connections.	No
num_netio_threads	0	Specifies the number of networking IO threads. 0 is the number of CPU cores.	No
num_max_connections	0	Max active connections for all networking threads. 0 means no limit. Max connections for each networking thread = num_max_connections / num_netio_threads	No
num_worker_threads	0	Specifies the number of threads that execute queries. 0 is the number of CPU cores.	No
ws_ip	0.0.0.0	Specifies the IP address for the HTTP service.	No
ws_http_port	19669	Specifies the port for the HTTP service.	No
heartbeat_interval_secs	10	Specifies the default heartbeat interval. Make sure the heartbeat_interval_secs values for all services are the same, otherwise NebulaGraph <b>CANNOT</b> work normally. This configuration is measured in seconds.	Yes
storage_client_timeout_ms	-	Specifies the RPC connection timeout threshold between the Graph Service and the Storage Service. This parameter is not predefined in the initial configuration files. You can manually set it if you need it. The system default value is 60000 ms.	No
<pre>slow_query_threshold_us</pre>	200000	When the execution time of a query exceeds the value, the query is called a slow query. Unit:	No

Name	Predefined value	Description	Whether supports runtime dynamic modifications
		Microsecond. <b>Note</b> : Even if the execution time of DML	
		statements exceeds this value, they will not be recorded as slow queries.	
ws_meta_http_port	19559	Specifies the Meta service listening port used by the HTTP protocol. It must be consistent with the ws_http_port in the Meta service configuration file.	No
Caution			

 $It is recommended to use a real IP when using IP address. Otherwise, \ 127.0.0.1/0.0.0. \ cannot be parsed correctly in some cases.$ 

# Authorization configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
enable_authorize	false	When set to false, the system authentication is not enabled. For more information, see Authentication.	No
auth_type	password	Specifies the login method. Available values are password, ldap, and cloud.	No

# Memory configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
system_memory_high_watermark_ratio	0.8	Specifies the trigger threshold of the high- level memory alarm mechanism. If the system memory usage is higher than this value, an alarm mechanism will be triggered, and NebulaGraph will stop querying. This parameter is not predefined in the initial configuration files.	Yes

# Metrics configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
enable_space_level_metrics	false	Enable or disable space-level metrics. Such metric names contain the name of the graph space that it monitors, for example, query_latency_us{space=basketballplayer}.avg.3600. You can view the supported metrics with the curl command. For more information, see Query NebulaGraph metrics.	No

# Session configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
max_sessions_per_ip_per_user	300	The maximum number of active sessions that can be created from a single IP adddress for a single user.	No

# Experimental configurations

O Note					
The switch of the experiment	The switch of the experimental feature is only available in the Community Edition.				
Name	Predefined value	Description	Whether supports runtime dynamic modifications		
enable_experimental_feature	false	Specifies the experimental feature. Optional values are true and false.	No		
enable_data_balance	true	Whether to enable the BALANCE DATA feature. Only works when enable_experimental_feature is true.	No		

## Memory tracker configurations

# Q Note

Memory Tracker is a memory management tool designed to monitor and limit memory usage. For large-scale queries, Memory Tracker can prevent Out Of Memory (OOM) issues. If you're using Memory Tracker in a containerized environment, you need to add the relevant configurations to the configuration file of the Graph service.

. Create the directory /sys/fs/cgroup/graphd/, and then add and configure the memory.max file under the directory.

2. Add the following configurations to etc/nebula-graphd.conf.

--containerized=true

- --containerized=true
  --cgroup\_v2\_controllers=/sys/fs/cgroup/graphd/cgroup.controllers
  --cgroup\_v2\_memory\_stat\_path=/sys/fs/cgroup/graphd/memory.stat
  --cgroup\_v2\_memory\_max\_path=/sys/fs/cgroup/graphd/memory.max
  --cgroup\_v2\_memory\_current\_path=/sys/fs/cgroup/graphd/memory.current

For more details, see Memory Tracker: Memory Management Practice in NebulaGraph Database.

Name	Predefined value	Description	Whether supports runtime dynamic modifications
<pre>memory_tracker_limit_ratio</pre>	0.8	<ul> <li>The value of this parameter can be set to (0, 1], 2, and 3.</li> <li>Caution: When setting this parameter, ensure that the value of system_memory_high_watermark_ratio is not set to 1, otherwise the value of this parameter will not take effect.</li> <li>(0, 1]: The percentage of available memory. Formula: Percentage of available memory = Available memory / (Total memory = Available memory is released.</li> <li>Note: For the hybrid deployment of a cluster with cloud-based and on-premises nodes, the value of</li> <li>memory_tracker_limit_ratio should be set to a Iower value. For example, when the graphd is expected to occupy only 50% of memory, the value can be set to less than 0.5.</li> <li>2: Dynamic Self Adaptive mode.</li> <li>MemoryTracker dynamically adjusts the available memory based on the system's current available memory.</li> <li>Note: This feature is experimental. As memory usage cannot be monitored in real time in dynamic adaptive mode, an OOM error may still occur to handle large memory allocations.</li> <li>3: Disable MemoryTracker.</li> <li>MemoryTracker only logs memory usage and does not interfere with executions even if the limit is exceeded.</li> </ul>	Yes
<pre>memory_tracker_untracked_reserved_memory_mb</pre>	50	The reserved memory that is not tracked by the memory tracker. Unit: MB.	Yes
memory_tracker_detail_log	false	Whether to enable the memory tracker log. When the value is true, the memory tracker log is generated.	Yes
<pre>memory_tracker_detail_log_interval_ms</pre>	60000	The time interval for generating the memory tracker log. Unit: Millisecond. memory_tracker_detail_log is true when this parameter takes effect.	Yes
memory_purge_enabled	true	Whether to enable the memory purge feature. When the value is true, the memory purge feature is enabled.	Yes
<pre>memory_purge_interval_seconds</pre>	10	The time interval for the memory purge feature to purge memory. Unit: Second.	Yes

Name	Predefined value	Description	Whether supports runtime dynamic modifications
		This parameter only takes effect if memory_purge_enabled is set to true.	

# performance optimization configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
max_job_size	1	The maximum number of concurrent jobs, i.e., the maximum number of threads used in the phase of query execution where concurrent execution is possible. It is recommended to be half of the physical CPU cores.	Yes
min_batch_size	8192	The minimum batch size for processing the dataset. Takes effect only when max_job_size is greater than 1.	Yes
optimize_appendvertices	false	When enabled, the MATCH statement is executed without filtering dangling edges.	Yes
path_batch_size	10000	The number of paths constructed per thread.	Yes

Last update: April 19, 2024

## 6.1.4 Storage Service configurations

NebulaGraph provides two initial configuration files for the Storage Service, nebula-storaged.conf.default and nebulastoraged.conf.production. Users can use them in different scenarios conveniently. The default file path is /usr/local/nebula/etc/.

Caution
t is not recommended to modify the value of local_config to false. If modified, the NebulaGraph service will first read the cached
configurations, which may cause configuration inconsistencies between clusters and cause unknown risks.

• It is not recommended to modify the configurations that are not introduced in this topic, unless you are familiar with the source code and fully understand the function of configurations.

#### How to use the configuration files

To use the initial configuration file, choose one of the above two files and delete the suffix .default or .production from the initial configuration file for the Meta Service to apply the configurations defined in it.

#### About parameter values

If a parameter is not set in the configuration file, NebulaGraph uses the default value. Not all parameters are predefined. And the predefined parameters in the two initial configuration files are different. This topic uses the parameters in <code>nebula-metad.conf.default</code>. For parameters that are not included in <code>nebula-metad.conf.default</code>, see <code>nebula-storaged.conf.production</code>.

# Caution

Some parameter values in the configuration file can be dynamically modified during runtime. We label these parameters as **Yes** that supports runtime dynamic modification in this article. When the local\_config value is set to true, the dynamically modified configuration is not persisted, and the configuration will be restored to the initial configuration after the service is restarted. For more information, see Modify configurations.

# Note

The configurations of the Raft Listener and the Storage service are different. For details, see Deploy Raft listener.

For all parameters and their current values, see Configurations.

# **Basics configurations**

Name	Predefined value	Description	Whether supports runtime dynamic modifications
daemonize	true	When set to true, the process is a daemon process.	No
pid_file	pids/nebula- storaged.pid	The file that records the process ID.	No
timezone_name	UTC+00:00:00	Specifies the NebulaGraph time zone. This parameter is not predefined in the initial configuration files, if you need to use this parameter, add it manually. For the format of the parameter value, see Specifying the Time Zone with TZ. For example,timezone_name=UTC+08:00 represents the GMT+8 time zone.	No
local_config	true	When set to true, the process gets configurations from the configuration files.	No

#### Q Note

• While inserting property values of time types, NebulaGraph transforms time types (except TIMESTAMP) to the corresponding UTC according to the time zone specified by timezone\_name. The time-type values returned by nGQL queries are all UTC.

• timezone\_name is only used to transform the data stored in NebulaGraph. Other time-related data of the NebulaGraph processes still uses the default time zone of the host, such as the log printing time.

# Logging configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
log_dir	logs	The directory that stores the Storage service log. It is recommended to put logs on a different hard disk from the data.	No
minloglevel	0	Specifies the minimum level of the log. That is, log messages at or above this level. Optional values are 0 (INFO), 1 (WARNING), 2 (ERROR), 3 (FATAL). It is recommended to set it to 0 during debugging and 1 in a production environment. If it is set to 4, NebulaGraph will not print any logs.	Yes
V	0	Specifies the detailed level of VLOG. That is, log all VLOG messages less or equal to the level. Optional values are 0, 1, 2, 3, 4, 5. The VLOG macro provided by glog allows users to define their own numeric logging levels and control verbose messages that are logged with the parameter v. For details, see Verbose Logging.	Yes
logbufsecs	0	Specifies the maximum time to buffer the logs. If there is a timeout, it will output the buffered log to the log file. 0 means real-time output. This configuration is measured in seconds.	No
redirect_stdout	true	When set to true, the process redirects the stdout and stderr to separate output files.	No
<pre>stdout_log_file</pre>	graphd- stdout.log	Specifies the filename for the stdout log.	No
stderr_log_file	graphd- stderr.log	Specifies the filename for the stderr log.	No
stderrthreshold	3	Specifies the minloglevel to be copied to the stderr log.	No
<pre>timestamp_in_logfile_name</pre>	true	Specifies if the log file name contains a timestamp. true indicates yes, false indicates no.	No

# Networking configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
meta_server_addrs	127.0.0.1:9559	Specifies the IPs (or hostnames) and ports of all Meta Services. Multiple addresses are separated with commas.	No
local_ip	127.0.0.1	Specifies the local IP (or hostname) for the Storage Service. The local IP address is used to identify the nebula-storaged process. If it is a distributed cluster or requires remote access, modify it to the corresponding address.	No
port	9779	Specifies RPC daemon listening port of the Storage service. The neighboring ports -1 (9778) and +1 (9780) are also used. 9778: The port used by the Admin service, which receives Meta commands for Storage. 9780: The port used for Raft communication between Storage services.	No
ws_ip	0.0.0.0	Specifies the IP address for the HTTP service.	No
ws_http_port	19779	Specifies the port for the HTTP service.	No
heartbeat_interval_secs	10	Specifies the default heartbeat interval. Make sure the heartbeat_interval_secs values for all services are the same, otherwise NebulaGraph <b>CANNOT</b> work normally. This configuration is measured in seconds.	Yes

Eaution

It is recommended to use a real IP when using IP address. Otherwise, 127.0.0.1/0.0.0 cannot be parsed correctly in some cases.

# Raft configurations

Name	Predefined value	Description	Whether supports runtime dynamic modifications
raft_heartbeat_interval_secs	30	Specifies the time to expire the Raft election. The configuration is measured in seconds.	Yes
<pre>raft_rpc_timeout_ms</pre>	500	Specifies the time to expire the Raft RPC. The configuration is measured in milliseconds.	Yes
wal_ttl	14400	Specifies the lifetime of the RAFT WAL. The configuration is measured in seconds.	Yes

# Disk configurations

Name	Predefined value	Description
data_path	data/storage	Specifies the data storage path. Multiple paths are separated with commas. For NebulaGraph of the community edition, one RocksDB instance corresponds to one path.
minimum_reserved_bytes	268435456	Specifies the minimum remaining space of each data storage path. When the value is lower than this standard, the cluster data writing may fail. This configuration is measured in bytes.
rocksdb_batch_size	4096	Specifies the block cache for a batch operation. The configuration is measured in bytes.
rocksdb_block_cache	4	Specifies the block cache for BlockBasedTable. The configuration is measured in megabytes.
disable_page_cache	false	Enables or disables the operating system's page cache for NebulaGraph. By default, the parameter value is false and page cache is enabled. If the value is set to true, page cache is disabled and sufficient block cache space must be configured for NebulaGraph.
engine_type	rocksdb	Specifies the engine type.
rocksdb_compression	Lz4	Specifies the compression algorithm for RocksDB. Optional values are no, snappy, lz4, lz4hc, zlib, bzip2, and zstd. This parameter modifies the compression algorithm for each level. If you want to set different compression algorithms for each level, use the parameter rocksdb_compression_per_level.
rocksdb_compression_per_level	١	Specifies the compression algorithm for each level. The priority is higher than rocksdb_compression. For example, no:no:lz4:lz4:snappy:zstd:snappy. You can also not set certain levels of compression algorithms, for example, no:no:lz4:lz4::zstd, level L4 and L6 use the compression algorithm of rocksdb_compression.
enable_rocksdb_statistics	false	When set to false, RocksDB statistics is disabled.
rocksdb_stats_level	kExceptHistogramOrTimers	Specifies the stats level for RocksDB. Optional values are kExceptHistogramOrTimers, kExceptTimers, kExceptDetailedTimers, kExceptTimeForMutex, and kAll.
enable_rocksdb_prefix_filtering	true	When set to true, the prefix bloom filter for RocksDB is enabled. Enabling prefix bloom filter makes the graph traversal faster but occupies more memory.
enable_rocksdb_whole_key_filtering	false	When set to true, the whole key bloom filter for RocksDB is enabled.
rocksdb_filtering_prefix_length	12	Specifies the prefix length for each key. Optional values are 12 and 16. The configuration is measured in bytes.
enable_partitioned_index_filter	false	When set to true, it reduces the amount of memory used by the bloom filter. But in some random-seek situations, it may reduce the read performance. This parameter is not predefined in the initial configuration files, if you need to use this parameter, add it manually.

## **RocksDB** options

Name	Predefined value	Description	Whether supports runtime dynamic modifications
rocksdb_db_options	0	Specifies the RocksDB database options.	No
rocksdb_column_family_options	{"write_buffer_size":"67108864", "max_write_buffer_number":"4", "max_bytes_for_level_base":"268435456"}	Specifies the RocksDB column family options.	No
rocksdb_block_based_table_options	{"block_size":"8192"}	Specifies the RocksDB block based table options.	No

The format of the RocksDB option is {"<option\_name>":"<option\_value>"}. Multiple options are separated with commas.

Supported options of rocksdb\_db\_options and rocksdb\_column\_family\_options are listed as follows.

#### rocksdb\_db\_options

- max\_total\_wal\_size delete\_obsolete\_files\_period\_micros max\_background\_jobs stats\_dump\_period\_sec compaction\_readahead\_size writable\_file\_max\_buffer\_size bytes\_per\_sync wal\_bytes\_per\_sync delayed\_write\_rate avoid\_flush\_during\_shutdown max\_open\_files stats\_persist\_period\_sec stats\_history\_buffer\_size strict\_bytes\_per\_sync enable\_rocksdb\_prefix\_filtering enable\_rocksdb\_rhole\_key\_filtering rocksdb\_filtering\_prefix\_length num\_compaction\_threads rate\_limit
- rocksdb\_column\_family\_options
  - write\_buffer\_size max\_write\_buffer\_number level0\_file\_num\_compaction\_trigger level0\_slowdown\_writes\_trigger target\_file\_size\_base target\_file\_size\_base target\_file\_size\_multiplier max\_bytes\_for\_level\_base max\_bytes\_for\_level\_multiplier disable\_auto\_compactions

For more information, see RocksDB official documentation.

# Misc configurations

# Caution

The configuration snapshot in the following table is different from the snapshot in NebulaGraph. The snapshot here refers to the stock data on the leader when synchronizing Raft.

Name	Predefined value	Description	Whether supports runtime dynamic modifications
query_concurrently	true	Whether to turn on multi-threaded queries. Enabling it can improve the latency performance of individual queries, but it will reduce the overall throughput under high pressure.	Yes
<pre>auto_remove_invalid_space</pre>	true	After executing DROP SPACE, the specified graph space will be deleted. This parameter sets whether to delete all the data in the specified graph space at the same time. When the value is true, all the data in the specified graph space will be deleted at the same time.	Yes
num_io_threads	16	The number of network I/O threads used to send RPC requests and receive responses.	No
num_max_connections	0	Max active connections for all networking threads. 0 means no limit. Max connections for each networking thread = num_max_connections / num_netio_threads	No
num_worker_threads	32	The number of worker threads for one RPC-based Storage service.	No
<pre>max_concurrent_subtasks</pre>	10	The maximum number of concurrent subtasks to be executed by the task manager.	No
<pre>snapshot_part_rate_limit</pre>	10485760	The rate limit when the Raft leader synchronizes the stock data with other members of the Raft group. Unit: bytes/s.	Yes
<pre>snapshot_batch_size</pre>	1048576	The amount of data sent in each batch when the Raft leader synchronizes the stock data with other members of the Raft group. Unit: bytes.	Yes
rebuild_index_part_rate_limit	4194304	The rate limit when the Raft leader synchronizes the index data rate with other members of the Raft group during the index rebuilding process. Unit: bytes/s.	Yes
rebuild_index_batch_size	1048576	The amount of data sent in each batch when the Raft leader synchronizes the index data with other members of the Raft group during the index rebuilding process. Unit: bytes.	Yes

## Memory Tracker configurations

# Q Note

Memory Tracker is a memory management tool designed to monitor and limit memory usage. For large-scale queries, Memory Tracker can prevent Out Of Memory (OOM) issues. If you're using Memory Tracker in a containerized environment, you need to add the relevant configurations to the configuration file of the Storage service.

. Create the directory /sys/fs/cgroup/storaged/, and then add and configure the memory.max file under the directory.

2. Add the following configurations to etc/nebula-storaged.conf.

--containerized=true

- --containerized=true
  --cgroup\_v2\_controllers=/sys/fs/cgroup/graphd/cgroup.controllers
  --cgroup\_v2\_memory\_stat\_path=/sys/fs/cgroup/graphd/memory.stat
  --cgroup\_v2\_memory\_max\_path=/sys/fs/cgroup/graphd/memory.max
  --cgroup\_v2\_memory\_current\_path=/sys/fs/cgroup/graphd/memory.current

For more details, see Memory Tracker: Memory Management Practice in NebulaGraph Database.

Name	Predefined value	Description	Whether supports runtime dynamic modifications
<pre>memory_tracker_limit_ratio</pre>	0.8	The value of this parameter can be set to (0, 1], 2, and 3. (0, 1]: The percentage of available memory. Formula: Percentage of available memory = Available memory / (Total memory - Reserved memory). When an ongoing query results in memory usage exceeding the configured limit, the query fails and subsequently the memory is released. <b>Note</b> : For the hybrid deployment of a cluster with cloud-based and on-premises nodes, the value of memory_tracker_limit_ratio should be set to a <b>lower</b> value. For example, when the graphd is expected to occupy only 50% of memory, the value can be set to less than 0.5. 2 : Dynamic Self Adaptive mode. MemoryTracker dynamically adjusts the available memory based on the system's current available memory. <b>Note</b> : This feature is experimental. As memory usage cannot be monitored in real time in dynamic adaptive mode, an OOM error may still occur to handle large memory allocations. 3 : Disable MemoryTracker. MemoryTracker only logs memory usage and does not interfere with executions even if the limit is exceeded.	Yes

${\tt memory\_tracker\_untracked\_reserved\_memory\_mb}$	50	The reserved memory that is not tracked by the Memory Tracker. Unit: MB.	Yes
<pre>memory_tracker_detail_log</pre>	false	Whether to enable the Memory Tracker log. When the value is true, the Memory Tracker log is generated.	Yes
<pre>memory_tracker_detail_log_interval_ms</pre>	60000	The time interval for generating the Memory Tracker log. Unit: Millisecond. memory_tracker_detail_log is true when this parameter takes effect.	Yes
memory_purge_enabled	true	Whether to enable the memory purge feature. When the value is true, the memory purge feature is enabled.	Yes
memory_purge_interval_seconds	10	The time interval for the memory purge feature to purge memory. Unit: Second. This parameter only takes effect if memory_purge_enabled is set to true.	Yes

## For super-Large vertices

When the query starting from each vertex gets an edge, truncate it directly to avoid too many neighboring edges on the superlarge vertex, because a single query occupies too much hard disk and memory. Or you can truncate a certain number of edges specified in the Max\_edge\_returned\_per\_vertex parameter. Excess edges will not be returned. This parameter applies to all spaces.

Property name	Default value	Description	Whether supports runtime dynamic modifications
max_edge_returned_per_vertex	2147483647	Specifies the maximum number of edges returned for each dense vertex. Excess edges are truncated and not returned. This parameter is not predefined in the initial configuration files, if you need to use this parameter, add it manually.	No

## Storage configurations for large dataset

<b>A</b> arning	
One graph space takes up at least about 300 MB of memory.	

When you have a large dataset (in the RocksDB directory) and your memory is tight, we suggest that you set the enable\_partitioned\_index\_filter parameter to true. The performance is affected because RocksDB indexes are cached.

Last update: April 19, 2024

## 6.1.5 Kernel configurations

This topic introduces the Kernel configurations in Nebula Graph.

#### **Resource control**

Q Note

You may run the ulimit command to control the resource threshold. However, the changes made only take effect for the current session or sub-process. To make permanent changes, edit file /etc/security/limits.conf. The configuration is as follows:

# <domain></domain>	<type></type>	<item></item>	<value></value>
*	soft	core	unlimited
*	hard	core	unlimited
*	soft	nofile	130000
*	hard	nofile	130000

The configuration modification takes effect for new sessions.

The parameter descriptions are as follows.

Parameter	Description
doma i n	Control Domain. This parameter can be a user name, a user group name (starting with $@$ ), or $\ge$ to indicate all users.
type	Control type. This parameter can be soft or hard. soft indicates a soft threshold (the default threshold) for the resource and hard indicates a maximum value that can be set by the user. The ulimit command can be used to increase soft, but not to exceed hard.
item	Resource types. For example, core limits the size of the core dump file, and nofile limits the maximum number of file descriptors a process can open.
value	Resource limit value. This parameter can be a number, or unlimited to indicate that there is no limit.

You can run man limits.conf for more helpful information.

#### Memory

#### VM.SWAPPINESS

vm.swappiness specifies the percentage of the available memory before starting swap. The greater the value, the more likely the swap occurs. We recommend that you set it to 0. When set to 0, the page cache is removed first. Note that when vm.swappiness is 0, it does not mean that there is no swap.

#### VM.MIN\_FREE\_KBYTES

vm.min\_free\_kbytes specifies the minimum number of kilobytes available kept by Linux VM. If you have a large system memory, we recommend that you increase this value. For example, if your physical memory 128GB, set it to 5GB. If the value is not big enough, the system cannot apply for enough continuous physical memory.

#### VM.MAX\_MAP\_COUNT

vm.max\_map\_count limits the maximum number of vma (virtual memory area) for a process. The default value is 65530. It is enough for most applications. If your memory application fails because the memory consumption is large, increase the vm.max\_map\_count value.

#### VM.DIRTY\_\*

These values control the dirty data cache for the system. For write-intensive scenarios, you can make adjustments based on your needs (throughput priority or delay priority). We recommend that you use the system default value.

#### TRANSPARENT HUGE PAGES

Transparent Huge Pages (THP) is a memory management feature of the Linux kernel, which enhances the system's ability to use large pages. In most database systems, Transparent Huge Pages can degrade performance, so it is recommended to disable it.

Perform the following steps:

1. Edit the GRUB configuration file /etc/default/grub.

sudo vi /etc/default/grub

2. Add transparent\_hugepage=never to the GRUB\_CMDLINE\_LINUX option, and then save and exit.

GRUB\_CMDLINE\_LINUX="... transparent\_hugepage=never"

- 3. Update the GRUB configuration.
- For CentOS:

sudo grub2-mkconfig -o /boot/grub2/grub.cfg

#### • For Ubuntu:

sudo update-grub

#### 4. Reboot the computer.

sudo reboot

If you don't want to reboot, you can run the following commands to temporarily disable THP until the next reboot.

echo 'never' > /sys/kernel/mm/transparent\_hugepage/enabled echo 'never' > /sys/kernel/mm/transparent\_hugepage/defrag

#### Networking

NET.IPV4.TCP\_SLOW\_START\_AFTER\_IDLE

The default value of net.ipv4.tcp\_slow\_start\_after\_idle is 1. If set, the congestion window is timed out after an idle period. We recommend that you set it to 0, especially for long fat scenarios (high latency and large bandwidth).

#### NET.CORE.SOMAXCONN

net.core.somaxconn specifies the maximum number of connection queues listened by the socket. The default value is 128. For scenarios with a large number of burst connections, we recommend that you set it to greater than 1024.

#### NET.IPV4.TCP\_MAX\_SYN\_BACKLOG

net.ipv4.tcp\_max\_syn\_backlog specifies the maximum number of TCP connections in the SYN\_RECV (semi-connected) state. The setting rule for this parameter is the same as that of net.core.somaxconn.

#### NET.CORE.NETDEV\_MAX\_BACKLOG

net.core.netdev\_max\_backlog specifies the maximum number of packets. The default value is 1000. We recommend that you increase it to greater than 10,000, especially for 10G network adapters.

#### NET.IPV4.TCP\_KEEPALIVE\_\*

These values keep parameters alive for TCP connections. For applications that use a 4-layer transparent load balancer, if the idle connection is disconnected unexpectedly, decrease the values of tcp\_keepalive\_time and tcp\_keepalive\_intvl.

#### NET.IPV4.TCP\_RMEM/WMEM

net.ipv4.tcp\_wmem/rmem specifies the minimum, default, and maximum size of the buffer pool sent/received by the TCP socket. For long fat links, we recommend that you increase the default value to bandwidth (GB) \* RTT (ms).

#### SCHEDULER

For SSD devices, we recommend that you set scheduler to noop or none. The path is /sys/block/DEV\_NAME/queue/scheduler.

#### Other parameters

KERNEL.CORE\_PATTERN

we recommend that you set it to core and set kernel.core\_uses\_pid to 1.

## Modify parameters

SYSCTL

sysctl <conf\_name>

Checks the current parameter value.

• sysctl -w <conf\_name>=<value>

Modifies the parameter value. The modification takes effect immediately. The original value is restored after restarting.

• sysctl -p [<file\_path>]

Loads Linux parameter values from the specified configuration file. The default path is /etc/sysctl.conf .

### PRLIMIT

The prlimit command gets and sets process resource limits. You can modify the hard threshold by using it and the sudo command. For example, prlimit --nofile = 130000 --pid = \$\$ adjusts the maximum number of open files permitted by the current process to 14000. And the modification takes effect immediately. Note that this command is only available in RedHat 7u or higher versions.

Last update: January 8, 2024

# 6.2 Log management

## 6.2.1 Runtime logs

Runtime logs are provided for DBAs and developers to locate faults when the system fails.

**NebulaGraph** uses glog to print runtime logs, uses gflags to control the severity level of the log, and provides an HTTP interface to dynamically change the log level at runtime to facilitate tracking.

## Log directory

The default runtime log directory is /usr/local/nebula/logs/.

If the log directory is deleted while NebulaGraph is running, the log would not continue to be printed. However, this operation will not affect the services. To recover the logs, restart the services.

#### **Parameter descriptions**

- mintoglevel: Specifies the minimum level of the log. That is, no logs below this level will be printed. Optional values are 0 (INFO), 1 (WARNING), 2 (ERROR), 3 (FATAL). It is recommended to set it to 0 during debugging and 1 in a production environment. If it is set to 4, NebulaGraph will not print any logs.
- v: Specifies the detailed level of the log. The larger the value, the more detailed the log is. Optional values are 0, 1, 2, 3.

The default severity level for the metad, graphd, and storaged logs can be found in their respective configuration files. The default path is /usr/local/nebula/etc/.

### Check the severity level

Check all the flag values (log values included) of the current gflags with the following command.

\$ \$ curl <ws_ip>:<ws_port>/flags</ws_port></ws_ip>		
Parameter	Description	
ws_ip	The IP address for the HTTP service, which can be found in the configuration files above. The default value is $127.0.0.1$ .	
ws_port	The port for the HTTP service, which can be found in the configuration files above. The default values are 19559 (Meta), 19669 (Graph), and 19779 (Storage) respectively.	

#### Examples are as follows:

• Check the current minloglevel in the Meta service:

\$ curl 127.0.0.1:19559/flags | grep 'minloglevel'

• Check the current v in the Storage service:

\$ curl 127.0.0.1:19779/flags | grep -w 'v'

#### Change the severity level

Change the severity level of the log with the following command.

\$ curl -X PUT -H "Content-Type: application/json" -d '{"<key>":<value>[,"<key>":<value>]}' "<ws\_ip>:<ws\_port>/flags"

Parameter	Description
key	The type of the log to be changed. For optional values, see Parameter descriptions.
value	The level of the log. For optional values, see Parameter descriptions.
ws_ip	The IP address for the HTTP service, which can be found in the configuration files above. The default value is 127.0.0.1.
ws_port	The port for the HTTP service, which can be found in the configuration files above. The default values are 19559 (Meta), 19669 (Graph), and 19779 (Storage) respectively.

## Examples are as follows:

\$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19779/flags" # storaged \$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19669/flags" # graphd \$ curl -X PUT -H "Content-Type: application/json" -d '{"minloglevel":0,"v":3}' "127.0.0.1:19559/flags" # metad

If the log level is changed while NebulaGraph is running, it will be restored to the level set in the configuration file after restarting the service. To permanently modify it, see Configuration files.

## RocksDB runtime logs

RocksDB runtime logs are usually used to debug RocksDB parameters and stored in /usr/local/nebula/data/storage/nebula/\$id/data/LOG. \$id is the ID of the example.

## Log recycling

Glog does not inherently support log recycling. To implement this feature, you can either use cron jobs in Linux to regularly remove old log files or use the log management tool, logrotate, to rotate logs for regular archiving and deletion.

LOG RECYCLING USING CRON JOBS

This section provides an example of how to use cron jobs to regularly delete old log files from the Graph service's runtime logs.

1. In the Graph service configuration file, apply the following settings and restart the service:

```
timestamp_in_logfile_name = true
max_log_size = 500
```

- By setting timestamp\_in\_logfile\_name to true, the log file name includes a timestamp, allowing regular deletion of old log files.
- The max\_log\_size parameter sets the maximum size of a single log file in MB, such as 500. Once this size is exceeded, a new log file is automatically created. The default value is 1800.
- 2. Use the following command to open the cron job editor.

crontab -e

3. Add a cron job command to the editor to regularly delete old log files.

\* \* \* \* find <log\_path> -name "<YourProjectName>" -mtime +7 -delete

# Caution

The find command in the above command should be executed by the root user or a user with sudo privileges.

- \*\*\*\*\*: This cron job time field signifies that the task is executed every minute. For other settings, see Cron Expression.
- <log\_path>: The path of the service runtime log file, such as /usr/local/nebula/logs.
- <YourProjectName> : The log file name, such as nebula-graphd.\*.
- -mtime +7 : This deletes log files that are older than 7 days. Alternatively, use -mmin +n to delete log files older than n minutes. For details, see the find command.
- -delete : This deletes log files that meet the conditions.

For example, to automatically delete the Graph service runtime log files older than 7 days at 3 o'clock every morning, use:

0 3 \* \* \* find /usr/local/nebula/logs -name nebula-graphd.\* -mtime +7 -delete

4. Save the cron job and exit the editor.

LOG RECYCLING USING LOGROTATE

Logrotate is a tool that can rotate specified log files for archiving and recycling.

#### O Note

You must be the root user or a user with sudo privileges to install or run logrotate.

This section provides an example of how to use logrotate to manage the Graph service's INFO level log file (/usr/local/nebula/logs/ nebula-graphd.INFO.impl).

1. In the Graph service configuration file, set timestamp\_in\_logfile\_name to false so that the logrotate tool can recognize the log file name. Then, restart the service.

timestamp\_in\_logfile\_name = false

- 2. Install logrotate.
- For Debian/Ubuntu:

sudo apt-get install logrotate

• For CentOS/RHEL:

sudo yum install logrotate

3. Create a logrotate configuration file, add log rotation rules, and save the configuration file.

In the /etc/logrotate.d directory, create a new logrotate configuration file nebula-graphd.INFO.

sudo vim /etc/logrotate.d/nebula-graphd.INF0

#### Then, add the following content:

# The absolute path of the log file needs to be configured # And the file name cannot be a symbolic link file, such as `nebula-graph.INFO` /usr/local/nebula/logs/nebula-graphd.INFO.impl { daily rotate 2 copytruncate nocompress missingok notifempty create 644 root root dateext dateformat .%Y-%m-%d-%s maxsize lk

Parameter Description daily Rotate the log daily. Other available time units include hourly, daily, weekly, monthly, and yearly. Keep the most recent 2 log files before deleting the older one. rotate 2 Copy the current log file and then truncate it, ensuring no disruption to the logging process. copytruncate Do not compress the old log files. nocompress missingok Do not report errors if the log file is missing. Do not rotate the log file if it's empty. notifempty Create a new log file with the specified permissions and ownership. create 644 root root dateext Add a date extension to the log file name. The default is the current date in the format -%Y%m%d. You can extend this using the dateformat option. dateformat .%Y-%m-%d-This must follow immediately after dateext and defines the file name after log rotation. Before V3.9.0, only %Y, %m, %d, and %s parameters were supported. %5 Starting from V3.9.0, the <sup>%</sup>H parameter is also supported. maxsize 1k Rotate the log when it exceeds 1 kilobyte (1024 bytes) in size or when the specified time unit (e.g., daily) passes. You can use size units like k and M, with the default unit being bytes.

Modify the parameters in the configuration file according to actual needs. For more information about parameter configuration, see logrotate.

4. Test the logrotate configuration.

To verify whether the logrotate configuration is correct, use the following command for testing.

sudo logrotate --debug /etc/logrotate.d/nebula-graphd.INF0

5. Execute logrotate.

Although logrotate is typically executed automatically by cron jobs, you can manually execute the following command to perform log rotation immediately.

sudo logrotate -fv /etc/logrotate.d/nebula-graphd.INF0

-fv: f stands for forced execution, v stands for verbose output.

6. Verify the log rotation results.

After log rotation, new log files are found in the /usr/local/nebula/logs directory, such as nebula-graphd.INF0.impl.2024-01-04-1704338204. The original log content is cleared, but the file is retained for new log entries. When the number of log files exceeds the value set by rotate, the oldest log file is deleted.

For example, rotate 2` means keeping the 2 most recently generated log files. When the number of log files exceeds 2, the oldest log file is deleted.

 [test@test logs]\$ ll

 -rw-r--r-- 1 root root
 0 Jan 4 11:18 nebula-graphd.INF0.impl

 -rw-r--r-- 1 root root
 6894 Jan 4 11:16 nebula-graphd.INF0.impl.2024-01-04-1704338204 # This file is deleted when a new log file is generated

 -rw-r--r-- 1 root root
 222 Jan 4 11:18 nebula-graphd.INF0.impl.2024-01-04-1704338207

 -rw-r--r-- 1 root root
 0 Jan 4 11:18 nebula-graphd.INF0.impl.2024-01-04-1704338287

 -rw-r--r-- 1 root root
 0 Jan 4 11:18 nebula-graphd.INF0.impl.2024-01-04-1704338287

 -rw-r--r-- 1 root root
 222 Jan 4 11:18 nebula-graphd.INF0.impl.2024-01-04-1704338287

 -rw-r--r-- 1 root root
 222 Jan 4 11:18 nebula-graphd.INF0.impl.2024-01-04-1704338287

If you need to rotate multiple log files, create multiple configuration files in the /etc/logrotate.d directory, with each configuration file corresponding to a log file. For example, to rotate the INFO level log file and the WARNING level log file of the Meta service, create two configuration files nebula-metad.INFO and nebula-metad.WARNING, and add log rotation rules in them respectively.

Last update: January 5, 2024

# 7. Monitor

# 7.1 Query NebulaGraph metrics

NebulaGraph supports querying the monitoring metrics through HTTP ports.

## 7.1.1 Metrics structure

Each metric of NebulaGraph consists of three fields: name, type, and time range. The fields are separated by periods, for example, num\_queries.sum.600. Different NebulaGraph services (Graph, Storage, or Meta) support different metrics. The detailed description is as follows.

Field	Example	Description
Metric name	num_queries	Indicates the function of the metric.
Metric type	sum	Indicates how the metrics are collected. Supported types are SUM, AVG, RATE, and the P-th sample quantiles such as P75, P95, P99, and P999.
Time range	600	The time range in seconds for the metric collection. Supported values are 5, 60, 600, and 3600, representing the last 5 seconds, 1 minute, 10 minutes, and 1 hour.

## 7.1.2 Query metrics over HTTP

#### Syntax

curl -G "http://<host>:<port>/stats?stats=<metric\_name\_list> [&format=json]"

Parameter	Description	
host	The IP (or hostname) of the server. You can find it in the configuration file in the installation directory.	
port	The HTTP port of the server. You can find it in the configuration file in the installation directory. The default ports are 19559 (Meta), 19669 (Graph), and 19779 (Storage).	
metric_name_list	The metrics names. Multiple metrics are separated by commas (,).	
&format=json	Optional. Returns the result in the JSON format.	

#### . Note

If NebulaGraph is deployed with Docker Compose, run docker-compose ps to check the ports that are mapped from the service ports inside of the container and then query through them.

## Query a single metric

Query the query number in the last 10 minutes in the Graph Service.

\$ curl -6 "http://192.168.8.40:19669/stats?stats=num\_queries.sum.600"
num\_queries.sum.600=400

### Query multiple metrics

Query the following metrics together:

- The average heartbeat latency in the last 1 minute.
- The average latency of the slowest 1% heartbeats, i.e., the P99 heartbeats, in the last 10 minutes.

```
$ curl -6 "http://192.168.8.40:19559/stats?stats=heartbeat_latency_us.avg.60,heartbeat_latency_us.p99.600"
heartbeat_latency_us.avg.60=281
heartbeat_latency_us.p99.600=985
```

## Return a JSON result.

Query the number of new vertices in the Storage Service in the last 10 minutes and return the result in the JSON format.

```
$ curl -6 "http://192.168.8.40:19779/stats?stats=num_add_vertices.sum.600&format=json"
[{"value":1,"name":"num_add_vertices.sum.600"}]
```

#### Query all metrics in a service.

If no metric is specified in the query, NebulaGraph returns all metrics in the service.

```
$ curl -G "http://192.168.8.40:19559/stats"
heartbeat_latency_us.avg.5=304
heartbeat_latency_us.avg.60=308
heartbeat_latency_us.avg.600=299
heartbeat_latency_us.avg.3600=285
heartbeat_latency_us.p75.5=652
heartbeat_latency_us.p75.60=669
heartbeat_latency_us.p75.600=651
heartbeat_latency_us.p75.3600=642
heartbeat_latency_us.p95.5=930
heartbeat_latency_us.p95.60=963
heartbeat_latency_us.p95.600=933
heartbeat_latency_us.p95.3600=929
heartbeat_latency_us.p99.5=986
heartbeat_latency_us.p99.60=1409
heartbeat_latency_us.p99.600=989
heartbeat_latency_us.p99.3600=986
num_heartbeats.rate.5=0
num_heartbeats.rate.60=0
num_heartbeats.rate.600=0
num heartbeats.rate.3600=0
num_heartbeats.sum.5=2
num_heartbeats.sum.60=40
num heartbeats.sum.600=394
num_heartbeats.sum.3600=2364
```

### Space-level metrics

The Graph service supports a set of space-level metrics that record the information of different graph spaces separately.

Space-level metrics can be queried only by querying all metrics. For example, run curl -6 "http://192.168.8.40:19559/stats" to show all metrics. The returned result contains the graph space name in the form of '{space=space\_name}', such as num\_active\_queries{space=basketballplayer}.sum.5=0.

# Caution

To enable space-level metrics, set the value of enable\_space\_level\_metrics to true in the Graph service configuration file before starting NebulaGraph. For details about how to modify the configuration, see Configuration Management.

# 7.1.3 Metric description

# Graph

Parameter	Description
num_active_queries	The number of changes in the number of active queries. Formula: The number of started queries minus the number of finished queries within a specified time.
num_active_sessions	The number of changes in the number of active sessions. Formula: The number of logged in sessions minus the number of logged out sessions within a specified time. For example, when querying <code>num_active_sessions.sum.5</code> , if there were 10 sessions logged in and 30 sessions logged out in the last 5 seconds, the value of this metric is -20 (10-30).
<pre>num_aggregate_executors</pre>	The number of executions for the Aggregation operator.
num_auth_failed_sessions_bad_username_password	The number of sessions where authentication failed due to incorrect username and password.
<pre>num_auth_failed_sessions_out_of_max_allowed</pre>	The number of sessions that failed to authenticate logins because the value of the parameter $\mbox{FLAG_OUT_OF_MAX_ALLOWED_CONNECTIONS}$ was exceeded.
num_auth_failed_sessions	The number of sessions in which login authentication failed.
num_indexscan_executors	The number of executions for index scan operators.
num_killed_queries	The number of killed queries.
num_opened_sessions	The number of sessions connected to the server.
num_queries	The number of queries.
<pre>num_query_errors_leader_changes</pre>	The number of the raft leader changes due to query errors.
num_query_errors	The number of query errors.
num_reclaimed_expired_sessions	The number of expired sessions actively reclaimed by the server.
num_rpc_sent_to_metad_failed	The number of failed RPC requests that the Graphd service sent to the Metad service.
<pre>num_rpc_sent_to_metad</pre>	The number of RPC requests that the Graphd service sent to the Metad service.
<pre>num_rpc_sent_to_storaged_failed</pre>	The number of failed RPC requests that the Graphd service sent to the Storaged service.
<pre>num_rpc_sent_to_storaged</pre>	The number of RPC requests that the Graphd service sent to the Storaged service.
num_sentences	The number of statements received by the Graphd service.
num_slow_queries	The number of slow queries.
num_sort_executors	The number of executions for the Sort operator.
optimizer_latency_us	The latency of executing optimizer statements.
query_latency_us	The latency of queries.
<pre>slow_query_latency_us</pre>	The latency of slow queries.
<pre>num_queries_hit_memory_watermark</pre>	The number of queries reached the memory watermark.
<pre>resp_part_completeness</pre>	The completeness of the partial success. You need to set accept_partial_success to true in the graph configuration first

# Meta

Parameter	Description
<pre>commit_log_latency_us</pre>	The latency of committing logs in Raft.
<pre>commit_snapshot_latency_us</pre>	The latency of committing snapshots in Raft.
heartbeat_latency_us	The latency of heartbeats.
num_heartbeats	The number of heartbeats.
num_raft_votes	The number of votes in Raft.
transfer_leader_latency_us	The latency of transferring the raft leader.
<pre>num_agent_heartbeats</pre>	The number of heartbeats for the AgentHBProcessor.
<pre>agent_heartbeat_latency_us</pre>	The latency of the AgentHBProcessor.
replicate_log_latency_us	The latency of replicating the log record to most nodes by Raft.
<pre>num_send_snapshot</pre>	The number of times that Raft sends snapshots to other nodes.
append_log_latency_us	The latency of replicating the log record to a single node by Raft.
append_wal_latency_us	The Raft write latency for a single WAL.
num_grant_votes	The number of times that Raft votes for other nodes.
num_start_elect	The number of times that Raft starts an election.

Storage

Parameter	Description
add_edges_latency_us	The latency of adding edges.
<pre>add_vertices_latency_us</pre>	The latency of adding vertices.
<pre>commit_log_latency_us</pre>	The latency of committing logs in Raft.
<pre>commit_snapshot_latency_us</pre>	The latency of committing snapshots in Raft.
delete_edges_latency_us	The latency of deleting edges.
delete_vertices_latency_us	The latency of deleting vertices.
<pre>get_neighbors_latency_us</pre>	The latency of querying neighbor vertices.
<pre>get_dst_by_src_latency_us</pre>	The latency of querying the destination vertex by the source vertex.
num_get_prop	The number of executions for the GetPropProcessor.
<pre>num_get_neighbors_errors</pre>	The number of execution errors for the GetNeighborsProcessor.
<pre>num_get_dst_by_src_errors</pre>	The number of execution errors for the GetDstBySrcProcessor.
<pre>get_prop_latency_us</pre>	The latency of executions for the GetPropProcessor.
<pre>num_edges_deleted</pre>	The number of deleted edges.
num_edges_inserted	The number of inserted edges.
<pre>num_raft_votes</pre>	The number of votes in Raft.
<pre>num_rpc_sent_to_metad_failed</pre>	The number of failed RPC requests that the Storage service sent to the Meta service.
<pre>num_rpc_sent_to_metad</pre>	The number of RPC requests that the Storaged service sent to the Metad service.
num_tags_deleted	The number of deleted tags.
<pre>num_vertices_deleted</pre>	The number of deleted vertices.
<pre>num_vertices_inserted</pre>	The number of inserted vertices.
<pre>transfer_leader_latency_us</pre>	The latency of transferring the raft leader.
<pre>lookup_latency_us</pre>	The latency of executions for the LookupProcessor.
num_lookup_errors	The number of execution errors for the LookupProcessor.
num_scan_vertex	The number of executions for the ScanVertexProcessor.
<pre>num_scan_vertex_errors</pre>	The number of execution errors for the ScanVertexProcessor.
update_edge_latency_us	The latency of executions for the UpdateEdgeProcessor.
<pre>num_update_vertex</pre>	The number of executions for the UpdateVertexProcessor.
<pre>num_update_vertex_errors</pre>	The number of execution errors for the UpdateVertexProcessor.
<pre>kv_get_latency_us</pre>	The latency of executions for the Getprocessor.
<pre>kv_put_latency_us</pre>	The latency of executions for the PutProcessor.
kv_remove_latency_us	The latency of executions for the RemoveProcessor.
<pre>num_kv_get_errors</pre>	The number of execution errors for the GetProcessor.
num_kv_get	The number of executions for the GetProcessor.
<pre>num_kv_put_errors</pre>	The number of execution errors for the PutProcessor.
num_kv_put	The number of executions for the PutProcessor.

Parameter	Description
<pre>num_kv_remove_errors</pre>	The number of execution errors for the RemoveProcessor.
num_kv_remove	The number of executions for the RemoveProcessor.
<pre>forward_tranx_latency_us</pre>	The latency of transmission.
<pre>scan_edge_latency_us</pre>	The latency of executions for the ScanEdgeProcessor.
num_scan_edge_errors	The number of execution errors for the ScanEdgeProcessor.
num_scan_edge	The number of executions for the ScanEdgeProcessor.
<pre>scan_vertex_latency_us</pre>	The latency of executions for the ScanVertexProcessor.
num_add_edges	The number of times that edges are added.
<pre>num_add_edges_errors</pre>	The number of errors when adding edges.
num_add_vertices	The number of times that vertices are added.
num_start_elect	The number of times that Raft starts an election.
<pre>num_add_vertices_errors</pre>	The number of errors when adding vertices.
num_delete_vertices_errors	The number of errors when deleting vertices.
append_log_latency_us	The latency of replicating the log record to a single node by Raft.
num_grant_votes	The number of times that Raft votes for other nodes.
replicate_log_latency_us	The latency of replicating the log record to most nodes by Raft.
num_delete_tags	The number of times that tags are deleted.
num_delete_tags_errors	The number of errors when deleting tags.
num_delete_edges	The number of edge deletions.
<pre>num_delete_edges_errors</pre>	The number of errors when deleting edges
num_send_snapshot	The number of times that snapshots are sent.
update_vertex_latency_us	The latency of executions for the UpdateVertexProcessor.
append_wal_latency_us	The Raft write latency for a single WAL.
<pre>num_update_edge</pre>	The number of executions for the UpdateEdgeProcessor.
<pre>delete_tags_latency_us</pre>	The latency of deleting tags.
<pre>num_update_edge_errors</pre>	The number of execution errors for the UpdateEdgeProcessor.
<pre>num_get_neighbors</pre>	The number of executions for the GetNeighborsProcessor.
<pre>num_get_dst_by_src</pre>	The number of executions for the GetDstBySrcProcessor.
<pre>num_get_prop_errors</pre>	The number of execution errors for the GetPropProcessor.
num_delete_vertices	The number of times that vertices are deleted.
num_lookup	The number of executions for the LookupProcessor.
num_sync_data	The number of times the Storage service synchronizes data from the Drainer.
<pre>num_sync_data_errors</pre>	The number of errors that occur when the Storage service synchronizes data from the Drainer.
<pre>sync_data_latency_us</pre>	The latency of the Storage service synchronizing data from the Drainer.

# Graph space

#### Q Note

Space-level metrics are created dynamically, so that only when the behavior is triggered in the graph space, the corresponding metric is created and can be queried by the user.

Parameter	Description
num_active_queries	The number of queries currently being executed.
num_queries	The number of queries.
num_sentences	The number of statements received by the Graphd service.
optimizer_latency_us	The latency of executing optimizer statements.
query_latency_us	The latency of queries.
num_slow_queries	The number of slow queries.
num_query_errors	The number of query errors.
<pre>num_query_errors_leader_changes</pre>	The number of raft leader changes due to query errors.
num_killed_queries	The number of killed queries.
<pre>num_aggregate_executors</pre>	The number of executions for the Aggregation operator.
num_sort_executors	The number of executions for the Sort operator.
num_indexscan_executors	The number of executions for index scan operators.
<pre>num_auth_failed_sessions_bad_username_password</pre>	The number of sessions where authentication failed due to incorrect username and password.
<pre>num_auth_failed_sessions</pre>	The number of sessions in which login authentication failed.
num_opened_sessions	The number of sessions connected to the server.
<pre>num_queries_hit_memory_watermark</pre>	The number of queries reached the memory watermark.
<pre>num_reclaimed_expired_sessions</pre>	The number of expired sessions actively reclaimed by the server.
num_rpc_sent_to_metad_failed	The number of failed RPC requests that the Graphd service sent to the Metad service.
num_rpc_sent_to_metad	The number of RPC requests that the Graphd service sent to the Metad service.
<pre>num_rpc_sent_to_storaged_failed</pre>	The number of failed RPC requests that the Graphd service sent to the Storaged service.
<pre>num_rpc_sent_to_storaged</pre>	The number of RPC requests that the Graphd service sent to the Storaged service.
<pre>slow_query_latency_us</pre>	The latency of slow queries.

Last update: March 29, 2024

# 7.2 RocksDB statistics

NebulaGraph uses RocksDB as the underlying storage. This topic describes how to collect and show the RocksDB statistics of NebulaGraph.

## 7.2.1 Enable RocksDB

By default, the function of RocksDB statistics is disabled. To enable RocksDB statistics, you need to:

- 1. Modify the --enable\_rocksdb\_statistics parameter as true in the nebula-storaged.conf file. The default path of the configuration file is /use/local/nebula/etc.
- 2. Restart the service to make the modification valid.

## 7.2.2 Get RocksDB statistics

Users can use the built-in HTTP service in the storage service to get the following types of statistics. Results in the JSON format are supported.

- All RocksDB statistics.
- Specified RocksDB statistics.

### 7.2.3 Examples

Use the following command to get all RocksDB statistics:

```
curl -L "http://${storage_ip}:${port}/rocksdb_stats"
```

#### For example:

```
curl -L "http://172.28.2.1:19779/rocksdb_stats"
rocksdb.blobdb.blob.file.bytes.read=0
rocksdb.blobdb.blob.file.bytes.written=0
rocksdb.blobdb.blob.file.bytes.synced=0
```

Use the following command to get specified RocksDB statistics:

curl -L "http://\${storage\_ip}:\${port}/rocksdb\_stats?stats=\${stats\_name}"

For example, use the following command to get the information of rocksdb.bytes.read and rocksdb.block.cache.add.

```
curl -L "http://172.28.2.1:19779/rocksdb_stats?stats=rocksdb.bytes.read,rocksdb.block.cache.add"
rocksdb.block.cache.add=14
rocksdb.bytes.read=1632
```

Use the following command to get specified RocksDB statistics in the JSON format:

```
curl -L "http://${storage_ip}:${port}/rocksdb_stats?stats=${stats_name}&format=json"
```

For example, use the following command to get the information of rocksdb.bytes.read and rocksdb.block.cache.add and return the results in the JSON format.

# }

Last update: October 25, 2023

# 8. Data security

# 8.1 Authentication and authorization

## 8.1.1 Authentication

NebulaGraph replies on local authentication to implement access control.

NebulaGraph creates a session when a client connects to it. The session stores information about the connection, including the user information. If the authentication system is enabled, the session will be mapped to corresponding users.

# Note

By default, the authentication is disabled and NebulaGraph allows connections with the username root and any password.

### Local authentication

Local authentication indicates that usernames and passwords are stored locally on the server, with the passwords encrypted. Users will be authenticated when trying to visit NebulaGraph.

ENABLE LOCAL AUTHENTICATION

- 1. Modify the nebula-graphd.conf file (/usr/local/nebula/etc/ is the default path) to set the following parameters:
- --enable\_authorize : Set its value to true to enable authentication.

#### Q Note

• By default, the authentication is disabled and NebulaGraph allows connections with the username root and any password.

• You can use the username root and password nebula to log into NebulaGraph after enabling local authentication. This account has the build-in God role. For more information about roles, see Roles and privileges.

- --failed\_login\_attempts: This parameter is optional, and you need to add this parameter manually. Specify the attempts of continuously entering incorrect passwords for a single Graph service. When the number exceeds the limitation, your account will be locked. For multiple Graph services, the allowed attempts are number of services \* failed\_login\_attempts.
- --password\_lock\_time\_in\_secs : This parameter is optional, and you need to add this parameter manually. Specify the time how long your account is locked after multiple incorrect password entries are entered. Unit: second.
- 2. Restart the NebulaGraph services. For how to restart, see Manage NebulaGraph services.

Last update: April 9, 2024
### 8.1.2 User management

User management is an indispensable part of NebulaGraph access control. This topic describes how to manage users and roles.

After enabling authentication, only valid users can connect to NebulaGraph and access the resources according to the user roles.

## Note

• By default, the authentication is disabled. NebulaGraph allows connections with the username root and any password.

• Once the role of a user is modified, the user has to re-login to make the new role takes effect.

### CREATE USER

The root user with the  ${\bf GOD}$  role can run <code>CREATE USER</code> to create a new user.

• Syntax

CREATE USER [IF NOT EXISTS] <user\_name> [WITH PASSWORD '<password>'];

- IF NOT EXISTS : Detects if the user name exists. The user will be created only if the user name does not exist.
- user\_name : Sets the name of the user. The maximum length is 16 characters.
- password : Sets the password of the user. The default password is the empty string ( ''). The maximum length is 24 characters.
- Example

nebula> CREATE USER user1 WITH PASSWORD	'nebula'
nebula> SHOW USERS;	
++	+
Account   IP Whitelist	
+	+
"root"   ""	
"user1"   ""	

### GRANT ROLE

Users with the **GOD** role or the **ADMIN** role can run **GRANT** ROLE to assign a built-in role in a graph space to a user. For more information about NebulaGraph built-in roles, see Roles and privileges.

### • Syntax

GRANT ROLE <role\_type> ON <space\_name> TO <user\_name>;

### • Example

nebula> GRANT ROLE USER ON basketballplayer TO user1;

### **REVOKE ROLE**

Users with the **GOD** role or the **ADMIN** role can run **REVOKE** ROLE to revoke the built-in role of a user in a graph space. For more information about NebulaGraph built-in roles, see Roles and privileges.

### • Syntax

REVOKE ROLE <role\_type> ON <space\_name> FROM <user\_name>;

### • Example

nebula> REVOKE ROLE USER ON basketballplayer FROM user1;

### DESCRIBE USER

Users can run DESCRIBE USER to list the roles for a specified user.

### • Syntax

DESCRIBE USER <user\_name>;
DESC USER <user\_name>;

### • Example

```
nebula> DESCRIBE USER user1;
+----+
| role | space |
+----+
| "ADMIN" | "basketballplayer" |
+-----+
```

### SHOW ROLES

Users can run  $\ensuremath{\mathsf{SHOW}}$  ROLES to list the roles in a graph space.

### • Syntax

SHOW ROLES IN <space\_name>;

### • Example

<pre>nebula&gt; SHOW ROLES IN basketballplayer;</pre>
++
Account   Role Type
++
"user1"   "ADMIN"
++

### CHANGE PASSWORD

Users can run CHANGE PASSWORD to set a new password for a user. The old password is needed when setting a new one.

• Syntax

CHANGE PASSWORD <user\_name> FROM '<old\_password>' TO '<new\_password>';

• Example

```
nebula> CHANGE PASSWORD user1 FROM 'nebula' T0 'nebula123';
```

### ALTER USER

The root user with the **GOD** role can run ALTER USER to set a new password. The old password is not needed when altering the user.

### • Syntax

ALTER	USER	<user< th=""><th>_name&gt;</th><th>WITH</th><th>PASSWORD</th><th>'<password>'</password></th><th>;</th></user<>	_name>	WITH	PASSWORD	' <password>'</password>	;

### - Example

```
nebula> ALTER USER user2 WITH PASSWORD 'nebula';
```

### DROP USER

The root user with the  ${\bf GOD}$  role can run DROP USER to remove a user.

#### Q Note

Removing a user does not close the current session of the user, and the user role still takes effect in the session until the session is closed.

### • Syntax

DROP USER [IF EXISTS] <user\_name>;

• Example

nebula> DROP USER user1;

### SHOW USERS

The root user with the  ${\bf GOD}$  role can run <code>SHOW USERS</code> to list all the users.

### • Syntax

SHOW USERS;

• Example

nebula> SHOW USERS;						
++-	+					
Account	IP Whitelist					
++-	+					
"root"	""					
user1"	""					
"user2"	"192.168.10.10"					
++	+					

Last update: November 3, 2023

### 8.1.3 Roles and privileges

A role is a collection of privileges. You can assign a role to a user for access control.

### Built-in roles

NebulaGraph does not support custom roles, but it has multiple built-in roles:

### • GOD

- GOD is the original role with **all privileges** not limited to graph spaces. It is similar to root in Linux and administrator in Windows.
- When the Meta Service is initialized, the one and only GOD role user root is automatically created with the password nebula.

# Caution

Modify the password for root timely for security.

- When the --enable\_authorize parameter in the nebula-graphd.conf file (the default directory is /usr/local/nebula/etc/) is set to true:
- One cluster can only have one user with the GOD role. This user can manage all graph spaces in a cluster.
- Manual authorization of the God role is not supported. Only the root user with the default God role can be used.
- ADMIN
- An ADMIN role can **read and write** both the Schema and the data in a specific graph space.
- An ADMIN role of a graph space can grant DBA, USER, and GUEST roles in the graph space to other users.

### Note

Only roles lower than ADMIN can be authorized to other users.

- DBA
- A DBA role can **read and write** both the Schema and the data in a specific graph space.
- A DBA role of a graph space CANNOT grant roles to other users.
- USER
- A USER role can **read and write** data in a specific graph space.
- $\bullet$  The Schema information is  $\ensuremath{\textit{read-only}}$  to the USER roles in a graph space.
- GUEST
- A GUEST role can **only read** the Schema and the data in a specific graph space.

# Note

- NebulaGraph does not support custom roles. Users can only use the default built-in roles.
- A user can have only one role in a graph space. For authenticated users, see User management.

## Role privileges and allowed nGQL

The privileges of roles and the nGQL statements that each role can use are listed as follows.

Privilege	God	Admin	DBA	User	Guest	Allowed nGQL
Read space	Y	Y	Y	Y	Y	USE, DESCRIBE SPACE
Read schema	Y	Y	Y	Y	Y	DESCRIBE TAG , DESCRIBE EDGE , DESCRIBE TAG INDEX , DESCRIBE EDGE INDEX
Write schema	Y	Y	Y		Y	CREATE TAG, ALTER TAG, CREATE EDGE, ALTER EDGE, DROP TAG, DELETE TAG, DROP EDGE, CREATE TAG INDEX, CREATE EDGE INDEX, DROP TAG INDEX, DROP EDGE INDEX
Write user	Y					CREATE USER, DROP USER, ALTER USER
Write role	Y	Y				GRANT, REVOKE
Read data	Y	Y	Y	Y	Y	GO, SET, PIPE, MATCH, ASSIGNMENT, LOOKUP, YIELD, ORDER BY, FETCH VERTICES, Find, FETCH EDGES, FIND PATH, LIMIT, GROUP BY, RETURN
Write data	Y	Y	Y	Y		INSERT VERTEX, UPDATE VERTEX, INSERT EDGE, UPDATE EDGE, DELETE VERTEX, DELETE EDGES, DELETE TAG
Show operations	Y	Y	Y	Y	Y	SHOW, CHANGE PASSWORD
Job	Y	Y	Y	Y		SUBMIT JOB COMPACT, SUBMIT JOB FLUSH, SUBMIT JOB STATS, STOP JOB, RECOVER JOB, BUILD TAG INDEX, BUILD EDGE INDEX,INGEST, DOWNLOAD
Write space	Y					CREATE SPACE, DROP SPACE, CREATE SNAPSHOT, DROP SNAPSHOT, BALANCE, CONFIG

#### **S** Caution

• The results of SHOW operations are limited to the role of a user. For example, all users can run SHOW SPACES, but the results only include the graph spaces that the users have privileges.

• Only the GOD role can run SHOW USERS and SHOW SNAPSHOTS .

Last update: October 25, 2023

# 8.2 SSL encryption

NebulaGraph supports SSL encrypted transfers between the Client, Graph Service, Meta Service, and Storage Service, and this topic describes how to set up SSL encryption.

### 8.2.1 Precaution

Enabling SSL encryption will slightly affect the performance, such as causing operation latency.

### 8.2.2 Certificate modes

To use SSL encryption, SSL certificates are required. NebulaGraph supports two certificate modes.

### • Self-signed certificate mode

A certificate that is generated by the server itself and signed by itself. In the self-signed certificate mode, the server needs to generate its own SSL certificate and key, and then use its own private key to sign the certificate. It is suitable for building secure communications for systems and applications within a LAN.

### • CA-signed certificate mode

A certificate granted by a trusted third-party Certificate Authority (CA). In the CA signed certificate mode, the server needs to apply for an SSL certificate from a trusted CA and ensure the authenticity and trustworthiness of the certificate through the auditing and signing of the certificate authority center. It is suitable for public network environment, especially for websites, e-commerce and other occasions that need to protect user information security.

### 8.2.3 Authentication policies

Policies for the NebulaGraph community edition.

Scene	TLS
External device access to Graph	Modify the Graph configuration file to add the following parameters: enable_graph_ssl = trueca_path=xxxxxxcert_path=xxxxxxkey_path=xxxxxx
Graph access Meta	In the Graph/Meta configuration file, add the following parameters: enable_meta_ssl = trueca_path=xxxxxxcert_path=xxxxxxkey_path=xxxxxx
Graph access StorageMeta access Storage	In the Graph/Meta/Storage configuration file, add the following parameters: enable_storage_ssl = trueca_path=xxxxxxcert_path=xxxxxxkey_path=xxxxxx
Graph access Meta/StorageMeta access Storage	In the Graph/Meta/Storage configuration file, add the following parameters: enable_meta_ssl = trueenable_storage_ssl = trueca_path=xxxxxxcert_path=xxxxxx key_path=xxxxxx
External device access to GraphGraph access Meta/StorageMeta access Storage	In the Graph/Meta/Storage configuration file, add the following parameters: enable_ssl = trueca_path=xxxxxxcert_path=xxxxxxkey_path=xxxxxx

### The parameters are described below.

Parameter	Default value	Description
cert_path	-	The path to the SSL public key certificate. This certificate is usually a .pem or .crt file, which is used to prove the identity of the server side, and contains information such as the public key, certificate owner, digital signature, and so on.
key_path	-	The path to the SSL key. The SSL key is usually a .key file.
password_path	-	(Optional) The path to the password file for the SSL key. Some SSL keys are encrypted and require a corresponding password to decrypt. We need to store the password in a separate file and use this parameter to specify the path to the password file.
ca_path	-	The path to the SSL root certificate. The root certificate is a special SSL certificate that is considered the highest level in the SSL trust chain and is used to validate and authorize other SSL certificates.
enable_ssl	false	Whether to enable SSL encryption in all services. only.
enable_graph_ssl	false	Whether to enable SSL encryption in the Graph service only.
enable_meta_ssl	false	Whether to enable SSL encryption in the Meta service only.
enable_storage_ssl	false	Whether to enable SSL encryption in the Storage service only.

### 8.2.4 Example of TLS

1. For example, using self-signed certificates and TLS for data transfers between the client NebulaGraph Python, the Graph service, the Meta service, and the Storage service. You need to set up all three Graph/Meta/Storage configuration files as follows:

--enable\_ssl=true
--ca\_path=xxxxxx
--cert\_path=xxxxxx
--key\_path=xxxxxx

2. When the changes are complete, restart these services to make the configuration take effect.

3. To connect to the Graph service using NebulaGraph Python, you need to set up a secure socket and add a trusted CA. For code examples, see nebula-test-run.py.

Last update: October 25, 2023

# 9. Backup and restore

# 9.1 NebulaGraph BR Community

### 9.1.1 What is Backup & Restore

Backup & Restore (BR for short) is a Command-Line Interface (CLI) tool to back up data of graph spaces of NebulaGraph and to restore data from the backup files.

### Features

The BR has the following features. It supports:

- Backing up and restoring data in a one-click operation.
- Restoring data in the following backup file types:
- Local Disk (SSD or HDD). It is recommend to use local disk in test environment only.
- Amazon S3 compatible interface, such as Alibaba Cloud OSS, MinIO,Ceph RGW, etc.
- Backing up and restoring the entire NebulaGraph cluster.
- Backing up data of specified graph spaces (experimental).

### Limitations

- Supports NebulaGraph v3.x only.
- Supports full backup, but not incremental backup.
- Currently, NebulaGraph Listener and full-text indexes do not support backup.
- If you back up data to the local disk, the backup files will be saved in the local path of each server. You can also mount the NFS on your host to restore the backup data to a different host.
- Restoration requires that the number of the storage servers in the original cluster is the same as that of the storage servers in the target cluster and storage server IPs must be the same. Restoring the specified space will clear all the remaining spaces in the cluster.
- During the backup process, both DDL and DML statements in any specified graph spaces are blocked. We recommend that you do the operation within the low peak period of the business, for example, from 2:00 AM to 5:00 AM.
- During the restoration process, there is a time when NebulaGraph stops running.
- Using BR in a container-based NebulaGraph cluster is not supported.

### How to use BR

To use the BR, follow these steps:

- 1. Install BR.
- 2. Use BR to back up data.
- 3. Use BR to restore data from backup files.

Last update: November 3, 2023

### 9.1.2 Install BR

This topic introduces the installation of BR in bare-metal deployment scenarios.

### Notes

To use the BR (Community Edition) tool, you need to install the NebulaGraph Agent service, which is taken as a daemon for each machine in the cluster that starts and stops the NebulaGraph service, and uploads and downloads backup files. The BR (Community Edition) tool and the Agent plug-in are installed as described below.

### Version compatibility

NebulaGraph	BR	Agent
3.5.x ~ 3.6.0	3.6.0	3.6.x ~ 3.7.0
3.3.0 ~ 3.4.x	3.3.0	0.2.0 ~ 3.4.0
3.0.x ~ 3.2.x	0.6.1	0.1.0 ~ 0.2.0

### Install BR with a binary file

### 1. Install BR.

wget https://github.com/vesoft-inc/nebula-br/releases/download/v3.6.0/br-3.6.0-linux-amd64

2. Change the binary file name to br.

sudo mv br-3.6.0-linux-amd64 br

3. Grand execute permission to BR.

sudo chmod +x br

4. Run ./br version to check BR version.

[nebula-br]\$ ./br version Nebula Backup And Restore Utility Tool,V-3.6.0

### Install BR with the source code

Before compiling the BR, do a check of these:

- Go 1.14.x or a later version is installed.
- make is installed.

To compile the BR, follow these steps:

1. Clone the nebula-br repository to your machine.

git clone https://github.com/vesoft-inc/nebula-br.git

2. Change to the br directory.

<mark>cd</mark> nebula-br

3. Compile the BR.

make

Users can enter bin/br version on the command line. If the following results are returned, the BR is compiled successfully.

[nebula-br]\$ bin/br version NebulaGraph Backup And Restore Utility Tool,V-3.6.0

### Install Agent

NebulaGraph Agent is installed as a binary file in each machine and serves the BR tool with the RPC protocol.

In **each machine**, follow these steps:

### 1. Install Agent.

wget https://github.com/vesoft-inc/nebula-agent/releases/download/v3.7.0/agent-3.7.0-linux-amd64

#### 2. Rename the Agent file to agent.

sudo mv agent-3.7.0-linux-amd64 agent

### 3. Add execute permission to Agent.

sudo chmod +x agent

### 4. Start Agent.

#### Q Note

Before starting Agent, make sure that the Meta service has been started and Agent has read and write access to the corresponding NebulaGraph cluster directory and backup directory.

sudo nohup ./agent --agent\_node\_ip>:8888" --meta="<metad\_node\_ip>:9559" --ratelimit=<file\_size\_bt> > nebula\_agent.log 2>&1 &

- --agent : The IP address and port number of Agent.
- --meta : The IP address and access port of any Meta service in the cluster.
- --ratelimit: (Optional) Limits the speed of file uploads and downloads to prevent bandwidth from being filled up and making other services unavailable. Unit: Bytes.

For example:

sudo nohup ./agent --agent="192.168.8.129:8888" --meta="192.168.8.129:9559" --ratelimit=1048576 > nebula\_agent.log 2>&1 &

# Caution

The IP address format for --agent should be the same as that of Meta and Storage services set in the configuration files. That is, use the real IP addresses or use 127.0.0.1. Otherwise Agent does not run.

5. Log into NebulaGraph and then run the following command to view the status of Agent.

Host	Port   Status	Role   Git Info Sh	a   Version
"192.168.8.129"	8888   "ONLINE"	"AGENT"   "96646b8"	i i

### FAQ

THE ERROR `E\_LIST\_CLUSTER\_NO\_AGENT\_FAILURE

If you encounter E\_LIST\_CLUSTER\_NO\_AGENT\_FAILURE error, it may be due to the Agent service is not started or the Agent service is not registered to Meta service. First, execute SHOW HOSTS AGENT to check the status of the Agent service on all nodes in the cluster, when

the status shows OFFLINE, it means the registration of Agent failed, then check whether the value of the -meta option in the command to start the Agent service is correct.

### 9.1.3 Use BR to back up data

After the BR is installed, you can back up data of the entire graph space. This topic introduces how to use the BR to back up data.

### Prerequisites

To back up data with the BR, do a check of these:

- Install BR and Agent and run Agent on each host in the cluster.
- The NebulaGraph services are running.
- If you store the backup files locally, create a directory with the same absolute path on the meta servers, the storage servers, and the BR machine for the backup files and get the absolute path. Make sure the account has write privileges for this directory.

### Arning

In the production environment, we recommend that you mount Network File System (NFS) storage to the meta servers, the storage servers, and the BR machine for local backup, or use Amazon S3 or Alibaba Cloud OSS for remote backup. When you restore the data from local files, you must manually move these backup files to a specified directory, which causes redundant data and troubles. For more information, see Restore data from backup files.

### Procedure

In the BR installation directory (the default path of the compiled BR is ./bin/br ), run the following command to perform a full backup for the entire cluster.

#### Q Note

Make sure that the local path where the backup file is stored exists.

<sup>\$ ./</sup>br backup full --meta <ip\_address> --storage <storage\_path>

For example:

• Run the following command to perform a full backup for the entire cluster whose meta service address is 192.168.8.129:9559, and save the backup file to /home/nebula/backup/.

### Caution

If there are multiple metad addresses, you can use any one of them.

# Caution

If you back up data to a local disk, only the data of the leader metad is backed up by default. So if there are multiple metad processes, you need to manually copy the directory of the leader metad (path <storage\_path>/meta) and overwrite the corresponding directory of other follower meatd processes.

\$ ./br backup full --meta "192.168.8.129:9559" --storage "local:///home/nebula/backup/"

• Run the following command to perform a full backup for the entire cluster whose meta service address is 192.168.8.129:9559, and save the backup file to backup in the br-test bucket of the object storage service compatible with S3 protocol.

\$ ./br backup full --meta "192.168.8.129:9559" --s3.endpoint "http://192.168.8.129:9000" --storage="s3://br-test/backup/" --s3.access\_key=minioadmin --s3.secret\_key=minioadmin --s3.secret\_key

The parameters are as follows.

Parameter	Data type	Required	Default value	Description
-h,-help	-	No	None	Checks help for restoration.
debug	-	No	None	Checks for more log information.
log	string	No	"br.log"	Specifies detailed log path for restoration and backup.
meta	string	Yes	None	The IP address and port of the meta service.
space	string	Yes	None	(Experimental feature) Specifies the names of the spaces to be backed up. All spaces will be backed up if not specified. Multiple spaces can be specified, and format isspaces nba_01spaces nba_02.
storage	string	Yes	None	The target storage URL of BR backup data. The format is: <schema>://<path>. Schema: Optional values are local and s3. When selecting s3, you need to fill in s3.access_key, s3.endpoint, s3.region, and s3.secret_key. PATH: The path of the storage location.</path></schema>
 s3.access_key	string	No	None	Sets AccessKey ID.
s3.endpoint	string	No	None	Sets the S3 endpoint URL, please specify the HTTP or HTTPS scheme explicitly.
s3.region	string	No	None	Sets the region or location to upload or download the backup.
 s3.secret_key	string	No	None	Sets SecretKey for AccessKey ID.

## Next to do

After the backup files are generated, you can use the BR to restore them for NebulaGraph. For more information, see Use BR to restore data.

Last update: November 3, 2023

### 9.1.4 Use BR to restore data

If you use the BR to back up data, you can use it to restore the data to NebulaGraph. This topic introduces how to use the BR to restore data from backup files.

# Caution

During the restoration process, the data on the target NebulaGraph cluster is removed and then is replaced with the data from the backup files. If necessary, back up the data on the target cluster.

Caution

The restoration process is performed OFFLINE.

### Prerequisites

- Install BR and Agent and run Agent on each host in the cluster.
- No application is connected to the target NebulaGraph cluster.
- Make sure that the target and the source NebulaGraph clusters have the same topology, which means that they have exactly the same number of hosts. The number of data folders for each host is consistently distributed.

## Procedures

In the BR installation directory (the default path of the compiled BR is ./br), run the following command to perform a full backup for the entire cluster.

 $_{\mbox{1.}}$  Users can use the following command to list the existing backup information:

\$ ./br show --storage <storage\_path>

For example, run the following command to list the backup information in the local /home/nebula/backup path.

\$ ./br showstorage "local:///home/nebula/backup"					
+	++	+	++	++	
NAME	CREATE TIME	SPACES	FULL BACKUP	ALL SPACES	
+	++	+	++	+	
BACKUP_2022_02_10_07_40_41	2022-02-10 07:40:41	basketballplayer	true	true	
BACKUP_2022_02_11_08_26_43	2022-02-11 08:26:47	basketballplayer,foesa	true	true	
+	+	+	++	+	

Or, you can run the following command to list the backup information stored in S3 URL s3://192.168.8.129:9000/br-test/backup.

\$ ./br show --s3.endpoint "http://192.168.8.129:9000" --storage="s3://br-test/backup/" --s3.access\_key=minioadmin --s3.secret\_key=minioadmin --s3.region=default

Parameter	Data type	Required	Default value	Description
-h,-help	-	No	None	Checks help for restoration.
-debug	-	No	None	Checks for more log information.
-log	string	No	"br.log"	Specifies detailed log path for restoration and backup.
storage	string	Yes	None	The target storage URL of BR backup data. The format is: <schema>://<path>. Schema: Optional values are local and s3. When selecting s3, you need to fill in s3.access_key, s3.endpoint, s3.region, and s3.secret_key. PATH: The path of the storage location.</path></schema>
s3.access_key	string	No	None	Sets AccessKey ID.
s3.endpoint	string	No	None	Sets the S3 endpoint URL, please specify the HTTP or HTTPS scheme explicitly.
s3.region	string	No	None	Sets the region or location to upload or download the backup.
s3.secret_key	string	No	None	Sets SecretKey for AccessKey ID.

### 2. Run the following command to restore data.

\$ ./br restore full --meta <ip\_address> --storage <storage\_path> --name <backup\_name>

For example, run the following command to upload the backup files from the local /home/nebuta/backup/ to the cluster where the meta service's address is 192.168.8.129:9559.

\$ ./br restore full --meta "192.168.8.129:9559" --storage "local:///home/nebula/backup/" --name BACKUP\_2021\_12\_08\_18\_38\_08

Or, you can run the following command to upload the backup files from the S3 URL s3://192.168.8.129:9000/br-test/backup.

\$ ./br restore full --meta "192.168.8.129:9559" --s3.endpoint "http://192.168.8.129:9000" --storage="s3://br-test/backup/" --s3.access\_key=minioadmin --s3.secret\_key=minioadmin --s3.s

If the following information is returned, the data is restored successfully.

Restore succeed.

# Caution

If your new cluster hosts' IPs are not all the same as the backup cluster, after restoration, you should run add hosts to add the Storage host IPs in the new cluster one by one.

The parameters are as follows.

Parameter	Data type	Required	Default value	Description
-h,-help	-	No	None	Checks help for restoration.
-debug	-	No	None	Checks for more log information.
- Log	string	No	"br.log"	Specifies detailed log path for restoration and backup.
-meta	string	Yes	None	The IP address and port of the meta service.
-name	string	Yes	None	The name of backup.
storage	string	Yes	None	The target storage URL of BR backup data. The format is: \ <schema>://\<path>. Schema: Optional values are local and s3. When selecting s3, you need to fill in s3.access_key, s3.endpoint, s3.region, and s3.secret_key. PATH: The path of the storage location.</path></schema>
s3.access_key	string	No	None	Sets AccessKey ID.
s3.endpoint	string	No	None	Sets the S3 endpoint URL, please specify the HTTP or HTTPS scheme explicitly.
s3.region	string	No	None	Sets the region or location to upload or download the backup.
s3.secret_key	string	No	None	Sets SecretKey for AccessKey ID.

3. Run the following command to clean up temporary files if any error occurred during backup. It will clean the files in cluster and external storage. You could also use it to clean up old backups files in external storage.

\$ ./br cleanup --meta <ip\_address> --storage <storage\_path> --name <backup\_name>

The parameters are as follows.

Parameter	Data type	Required	Default value	Description
-h,-help	-	No	None	Checks help for restoration.
-debug	-	No	None	Checks for more log information.
-log	string	No	"br.log"	Specifies detailed log path for restoration and backup.
-meta	string	Yes	None	The IP address and port of the meta service.
-name	string	Yes	None	The name of backup.
storage	string	Yes	None	The target storage URL of BR backup data. The format is: \ <schema>://\<path>. Schema: Optional values are local and s3. When selecting s3, you need to fill in s3.access_key, s3.endpoint, s3.region, and s3.secret_key. PATH: The path of the storage location.</path></schema>
s3.access_key	string	No	None	Sets AccessKey ID.
s3.endpoint	string	No	None	Sets the S3 endpoint URL, please specify the HTTP or HTTPS scheme explicitly.
s3.region	string	No	None	Sets the region or location to upload or download the backup.
s3.secret_key	string	No	None	Sets SecretKey for AccessKey ID.

Last update: October 25, 2023

# 9.2 Backup and restore data with snapshots

NebulaGraph supports using snapshots to back up and restore data. When data loss or misoperation occurs, the data will be restored through the snapshot.

### 9.2.1 Prerequisites

NebulaGraph authentication is disabled by default. In this case, all users can use the snapshot feature.

If authentication is enabled, only the GOD role user can use the snapshot feature. For more information about roles, see Roles and privileges.

### 9.2.2 Precautions

- To prevent data loss, create a snapshot as soon as the system structure changes, for example, after operations such as ADD HOST, DROP HOST, CREATE SPACE, DROP SPACE, and BALANCE are performed.
- NebulaGraph cannot automatically delete the invalid files created by a failed snapshot task. You have to manually delete them by using DROP SNAPSHOT.
- Customizing the storage path for snapshots is not supported for now.

### 9.2.3 Create snapshots

Run CREATE SNAPSHOT to create a snapshot for all the graph spaces based on the current time for NebulaGraph. Creating a snapshot for a specific graph space is not supported yet.

# Note

If the creation fails, refer to the later section to delete the corrupted snapshot and then recreate the snapshot.

nebula> CREATE SNAPSHOT;

### 9.2.4 View snapshots

To view all existing snapshots, run SHOW SNAPSHOTS.

nebula> SHOW SNAPSHOTS;		
Name	Status	Hosts
+	+   "VALID"	++
"SNAPSHOT_2021_03_09_09_10_52"	"VALID"	"127.0.0.1:9779"

The parameters in the return information are described as follows.

Parameter	Description
Name	The name of the snapshot directory. The prefix SNAPSHOT indicates that the file is a snapshot file, and the suffix indicates the time the snapshot was created (UTC).
Status	The status of the snapshot. VALID indicates that the creation succeeded, while INVALID indicates that it failed
Hosts	The IPs (or hostnames) and ports of all Storage servers at the time the snapshot was created.

### Snapshot path

Snapshots are stored in the path specified by the data\_path parameter in the Meta and Storage configuration files. When a snapshot is created, the checkpoints directory is checked in the datastore path of the leader Meta service and all Storage services for the existence, and if it is not there, it is automatically created. The newly created snapshot is stored as a subdirectory within the checkpoints directory. For example, SNAPSHOT\_2021\_03\_09\_08\_43\_12 . The suffix 2021\_03\_09\_08\_43\_12 is generated automatically based on the creation time (UTC).

To fast locate the path where the snapshots are stored, you can use the Linux command find in the datastore path. For example:

```
$ cd /usr/local/nebula-graph-ent-3.6.0/data
$ find |grep 'SNAPSHOT_2021_03_09_08_43_12'
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12/data
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_09_08_43_12/data/000081.sst
...
```

### 9.2.5 Delete snapshots

To delete a snapshot with the given name, run  $\ensuremath{\mathsf{DROP}}$  <code>SNAPSHOT</code> .

DROP SNAPSHOT <snapshot\_name>;

### Example:

nebula> nebula>	DROP SHOW	SNAPSHOT SNAPSHOT	SNAPSHOT_2 S;	202:	L_03_09_0	. 8	43_12;	
+   Name					Status	+-	Hosts	
"SNAP:	SHOT_2	2021_03_0	9_09_10_52'	1	"VALID"		"127.0.0.1	:9779"

Note

Deleting the only snapshot within the checkpoints directory also deletes the checkpoints directory.

### 9.2.6 Restore data with snapshots

### **A**rning

When you restore data with snapshots, make sure that the graph spaces backed up in the snapshot have not been dropped. Otherwise, the data of the graph spaces cannot be restored.

Currently, there is no command to restore data with snapshots. You need to manually copy the snapshot file to the corresponding folder, or you can make it by using a shell script. The logic implements as follows:

 After the snapshot is created, the checkpoints directory is generated in the installation directory of the leader Meta service and all Storage services, and saves the created snapshot. Taking this topic as an example, when there are two graph spaces, the snapshots created are saved in /usr/local/nebula/data/meta/nebula/0/checkpoints, /usr/local/nebula/data/storage/ nebula/3/checkpoints and /usr/local/nebula/data/storage/nebula/4/checkpoints.

```
$ ls /usr/local/nebula/data/meta/nebula/0/checkpoints/
SNAPSH0T_2021_03_09_09_10_52
$ ls /usr/local/nebula/data/storage/nebula/3/checkpoints/
SNAPSH0T_2021_03_09_09_10_52
$ ls /usr/local/nebula/data/storage/nebula/4/checkpoints/
SNAPSH0T_2021_03_09_00_10_52
```

2. To restore the lost data through snapshots, you can take a snapshot at an appropriate time, copy the folders data and wal in the corresponding snapshot directory to its parent directory (at the same level with checkpoints) to overwrite the previous data and wal, and then restart the cluster.

### Arning

The data and wal directories of all Meta services should be overwritten at the same time. Otherwise, the new leader Meta service will use the latest Meta data after a cluster is restarted.

Last update: November 22, 2023

# 10. Synchronize and migrate

# 10.1 BALANCE syntax

We can submit tasks to load balance Storage services in NebulaGraph. For more information about storage load balancing and examples, see Storage load balance.

Note
For other job management commands, see Job manager and the JOB statements.
The syntax for load balance is described as follows.

Syntax	Description
SUBMIT JOB BALANCE LEADER	Starts a job to balance the distribution of all the storage leaders in all graph spaces. It returns the job ID.

For details about how to view, stop, and restart a job, see Job manager and the JOB statements.

Last update: October 25, 2023

# 11. Import and export

# 11.1 Import tools

There are many ways to write NebulaGraph 3.6.0:

- Import with the command -f: This method imports a small number of prepared nGQL files, which is suitable to prepare for a small amount of manual test data.
- Import with Studio: This method uses a browser to import multiple CSV files of this machine. A single file cannot exceed 100 MB, and its format is limited.
- Import with Importer: This method imports multiple CSV files on a single machine with unlimited size and flexible format. Suitable for scenarios with less than one billion records of data.
- Import with Exchange: This method imports from various distribution sources, such as Neo4j, Hive, MySQL, etc., which requires a Spark cluster. Suitable for scenarios with more than one billion records of data.
- Read and write APIs with Spark-connector/Flink-connector: This method requires you to write a small amount of code to make use of the APIs provided by Spark/Flink connector.
- Import with C++/GO/Java/Python SDK: This method imports in the way of writing programs, which requires certain programming and tuning skills.

The following figure shows the positions of these ways:



### 11.1.1 Export tools

- Read and write APIs with Spark-connector/Flink-connector: This method requires you to write a small amount of code to make use of the APIs provided by Spark/Flink connector.
- Export the data in database to a CSV file or another graph space (different NebulaGraph database clusters are supported) using the export function of the Exchange.

# Sterpriseonly

The export function is exclusively available in the Enterprise Edition. If you require access to this version, please contact us.

Last update: February 18, 2024

# 11.2 NebulaGraph Importer

NebulaGraph Importer (Importer) is a standalone tool for importing data from CSV files into NebulaGraph. Importer can read and batch import CSV file data from multiple data sources, and also supports batch update and delete operations.

### 11.2.1 Features

- Support multiple data sources, including local, S3, OSS, HDFS, FTP, SFTP, and GCS.
- Support importing data from CSV format files. A single file can contain multiple tags, multiple edge types or a mix of both.
- Support filtering the data from source.
- Support batch operation, including insert, update, delete.
- Support connecting to multiple Graph services simultaneously for importing and dynamic load balancing.
- Support reconnect or retry after failure.
- Support displaying statistics in multiple dimensions, including import time, import percentage, etc. Support for printing statistics in Console or logs.
- Support SSL.

### 11.2.2 Advantage

- Lightweight and fast: no complex environment can be used, fast data import.
- Flexible filtering: You can flexibly filter CSV data through configuration files.

### 11.2.3 Version compatibility

The version correspondence between NebulaGraph and NebulaGraph Importer is as follows.

NebulaGraph version	NebulaGraph Importer version
3.x.x	3.x.x, 4.x.x
2.x.x	2.x.x, 3.x.x

#### Q Note

Importer 4.0.0 has redone the Importer for improved performance, but the configuration file is not compatible with older versions. It is recommended to use the new version of Importer.

### 11.2.4 Release note

Release

### 11.2.5 Prerequisites

Before using NebulaGraph Importer, make sure:

- NebulaGraph service has been deployed. The deployment method is as follows:
- Deploy NebulaGraph with Docker Compose
- Install NebulaGraph with RPM or DEB package
- Install NebulaGraph by compiling the source code
- Schema is created in NebulaGraph, including space, Tag and Edge type, or set by parameter manager.hooks.before.statements.

### 11.2.6 Steps

Prepare the CSV file to be imported and configure the YAML file to use the tool to batch write data into NebulaGraph.

Note

For details about the YAML configuration file, see Configuration File Description at the end of topic.

### Download binary package and run

1. Download the executable binary package.

# Note

The file installation path based on the RPM/DEB package is /usr/bin/nebula-importer.

2. Under the directory where the binary file is located, run the following command to start importing data.

./<binary\_file\_name> --config <yaml\_config\_file\_path>

### Source code compile and run

Compiling the source code requires deploying a Golang environment. For details, see Build Go environment.

1. Clone repository.

git clone -b release-4.1 https://github.com/vesoft-inc/nebula-importer.git

#### Q Note

Use the correct branch. Different branches have different RPC protocols.

2. Access the directory nebula-importer.

cd nebula-importer

3. Compile the source code.

make build

4. Start the service.

./bin/nebula-importer --config <yaml\_config\_file\_path>

### Run in Docker mode

Instead of installing the Go locale locally, you can use Docker to pull the image of the NebulaGraph Importer and mount the local configuration file and CSV data file into the container. The command is as follows:

```
docker pull vesoft/nebula-importer:~version>
docker run --rm -ti \
    --network=host \
    -v config_file>:<config_file> \
    -v <data_dir>:<data_dir> \
    vesoft/nebula-importer:<version> \
    --config_config_file>
```

- <config\_file> : The absolute path to the YAML configuration file.
- <data\_dir> : The absolute path to the CSV data file. If the file is not local, ignore this parameter.
- <version> : NebulaGraph 3.x Please fill in 'v3'.

#### O Note

A relative path is recommended. If you use a local absolute path, check that the path maps to the path in the Docker.

### Example:

```
docker pull vesoft/nebula-importer:v4
docker run --rm -ti \
    --network-host \
    -v /home/user/config.yaml:/home/user/config.yaml \
    -v /home/user/data:/home/user/data \
    vesoft/nebula-importer:v4 \
    --config /home/user/config.yaml
```

### 11.2.7 Configuration File Description

Various example configuration files are available within the Github of the NebulaGraph Importer. The configuration files are used to describe information about the files to be imported, NebulaGraph server information, etc. The following section describes the fields within the configuration file in categories.

```
Note
```

If users download a binary package, create the configuration file manually.

### **Client configuration**

Client configuration stores the configuration associated with the client's connection to the NebulaGraph.

The example configuration is as follows:

```
client:
version: v3
address: "192.168.1.100:9669,192.168.1.101:9669"
user: root
password: nebula
ssl:
    enable: true
    certPath: "/home/xxx/cert/importer.crt"
    keyPath: "/home/xxx/cert/importer.key"
    caPath: "/home/xxx/cert/importer.key"
    caPath: "/home/xxx/cert/importer.crt"
    insecureSkipVerify: false
    concurrencyPerAddress: 10
    reconnectInitialInterval: 1s
```

### retry: 3 retryInitialInterval: 1s

Parameter	Default value	Required	Description
client.version	v3	Yes	Specifies the major version of the NebulaGraph. Currently only v3 is supported.
client.address	"127.0.0.1:9669"	Yes	Specifies the address of the NebulaGraph. Multiple addresses are separated by commas.
client.user	root	No	NebulaGraph user name.
client.password	nebula	No	The password for the NebulaGraph user name.
client.ssl.enable	false	No	Specifies whether to enable SSL authentication.
client.ssl.certPath	-	No	Specifies the storage path for the SSL public key certificate.This parameter is required when SSL authentication is enabled.
client.ssl.keyPath	-	No	S pecifies the storage path for the SSL key.This parameter is required when SSL authentication is enabled.
client.ssl.caPath	-	No	Specifies the storage path for the CA root certificate.This parameter is required when SSL authentication is enabled.
client.ssl.insecureSkipVerify	false	No	Specifies whether the client skips verifying the server's certificate chain and hostname. If set to 'true', any certificate chain and hostname provided by the server is accepted.
client.concurrencyPerAddress	10	No	The number of concurrent client connections for a single graph service.
client.retryInitialInterval	1s	No	Reconnect interval time.
client.retry	3	No	The number of retries for failed execution of the nGQL statement.
client.retryInitialInterval	1s	No	Retry interval time.

### Manager configuration

Manager configuration is a human-controlled configuration after connecting to the database.

The example configuration is as follows:

UPDATE CONFIGS storage:wal\_ttl=86400; UPDATE CONFIGS storage:rocksdb\_column\_family\_options = { disable\_auto\_compactions = false };

Parameter	Default value	Required	Description
manager.spaceName	-	Yes	Specifies the NebulaGraph space to import the data into. Do not support importing multiple map spaces at the same time.
manager.batch	128	No	The batch size for executing statements (global configuration).
Setting the batch size individually for a data source can using the parameter sources.batch below.			
manager.readerConcurrency	50	No	The number of concurrent reads of the data source by the reader.
manager.importerConcurrency	512	No	The number of concurrent nGQL statements generated to be executed, and then will call the client to execute these nGQL statements.
manager.statsInterval	10s	No	The time interval for printing statistical information
<pre>manager.hooks.before.[].statements</pre>	-	No	The command to execute in the graph space before importing.
<pre>manager.hooks.before.[].wait</pre>	-	No	The wait time after statements are executed.
<pre>manager.hooks.after.[].statements</pre>	-	No	The commands to execute in the graph space after importing.
<pre>manager.hooks.after.[].wait</pre>	-	No	The wait time after statements are executed.

### Log configuration

Log configuration is the logging-related configuration.

The example configuration is as follows:

log: level: INFO console: true files: - logs/nebula-importer.	Log		
Parameter	Default value	Required	Description
log.level	INFO	No	Specifies the log level. Optional values are DEBUG , INFO , WARN , ERROR , PANIC , FATAL .
log.console	true	No	Whether to print the logs to console synchronously when storing logs.
log.files	-	No	The log file path. The log directory must exist.

### Source configuration

The Source configuration requires the configuration of data source information, data processing methods, and Schema mapping.

The example configuration is as follows:

sources - path: ./person.csv # Required. Specifies the path where the data files are stored. If a relative path is used, the path and current configuration file directory are spliced. Wildcard filename is also supported, for example: ./follower-\*.csv, please make sure that all matching files with the same schema. - s3: # AWS S3 # Optional. The endpoint of S3 service, can be omitted if using AWS S3. # Required. The region of S3 service. endpoint: endpoint region: us-east-1 bucket: gdelt-open-data # Required. The bucket of file in S3 service. key: events/20190918.export.csv # Required. The object key of file in S3 service. accessKeyID: "" # Optional. The access key of S3 service. If it is public data, no need to configure. accessKeySecret: "" # Optional. The secret key of S3 service. If it is public data, no need to configure. # Optional. The secret key of S3 service. If it is public data, no need to configure. - 0SS: endpoint: https://oss-cn-hangzhou.aliyuncs.com # Required. The endpoint of OSS service. # Required. The bucket of file in OSS service. bucket: bucketName # Required. The bucket of file in OSS service. key: objectKey # Required. The object key of file in OSS service. accesskeyDicaccesskey # Required. The access key of OSS service. accesskeySecret: secretKey # Required. The secret key of OSS service. - ftp: host: 192.168.0.10 # Required. The host of FTP service. port: 21 # Required. The port of FTP service. user: user # Required. The user of FTP service. password: password # Required. The password of FTP service.
path: "/events/20190918.export.csv" # Required. The path of file in the FTP service. - sftp: host: 192.168.0.10 # Required. The host of SFTP service. port: 22 # Required. The port of SFTP service. user: user # Required. The user of SFTP service. password: password # 0ptional. The password of SFTP service. keyFile: keyFile # 0ptional. The ssh key file path of SFTP service. keyData: keyData \$ 0ptional. The ssh key file content of SFTP service. passphrase: passphrase # Optional. The ssh key passphrase of SFTP service. path: "/events/20190918.export.csv" # Required. The path of file in the SFTP service. - hdfs address: "127.0.0.1:8020" # Required. The address of HDFS service. audress: 12/10/01/10/20 # Required. The audress of nors service. user: "hdfs" # Optional. The user of HDFS service. servicePrincipalName: <Kerberos Service Principal Name> # Optional. The name of the Kerberos service instance for the HDFS service when Kerberos authentication is enabled. krbSConfigFile: <Kerberos config file> # Optional. The path to the Kerberos configuration file for the HDFS service when Kerberos authentication is enabled. Defaults to `/etc/ krb5.conf` ccacheFile: <Kerberos ccache file> # Optional. The path to the Kerberos ccache file for the HDFS service when Kerberos authentication is enabled. keyTabFile: <Kerberos keytab file> # Optional. The path to the Kerberos keytab file for the HDFS service when Kerberos authentication is enabled. password: <Kerberos password> # Optional. The Kerberos password for the HDFS service when Kerberos authentication is enabled. dataTransferProtection: <Kerberos Data Transfer Protection> # Optional. The type of transport encryption when Kerberos authentication is enabled. Optional values are ntication', 'integrity', 'privacy'. disablePAFXFAST: false # Optional. Whether to disable the use of PA\_FX\_FAST for clients. path: "/events/20190918.export.csv" # Required. The path to the file in the HDFS service. Wildcard filenames are also supported, e.g. `/events/\*.export.csv`, make sure all authentication`, matching files have the same schema. - gcs: # Google Cloud Storage # bucket: chicago-crime-sample # Required. The name of the bucket in the GCS service. key: stats/000000000000.csv # Required. The path to the file in the GCS service. withoutAuthentication: false # Optional. Whether to anonymize access. Defaults to false, which means access with credentials. # When using credentials access, one of the credentials/lile and credentialsJSON parameters is sufficient. credentialsFile: "/path/to/your/credentials/file" # Optional. The path to the credentials file for the GCS service. credentialsJSON: '{ # Optional. The JSON content of the credentials for the GCS service. "type": "service\_account", type : servic\_account ; "project\_id": "your-project-id", "private\_key\_id": "key-id", "private\_key": "----BEGIN PRIVATE KEY-----\nxxxxx\n----END PRIVATE KEY-----\n", "client\_email": "your-client@your-project-id.iam.gserviceaccount.com", "client\_email: "your-clienteyour-project-in.nem.spectreexections and a second sec "client\_x509\_cert\_url": "https://www.googleapis.com/robot/v1/metadata/x509/your-client%40your-project-id.iam.gserviceaccount.com", "universe\_domain": "googleapis.com" batch: 256 csv: delimiter: "|" withHeader: false lazyQuotes: false tags: - name: Person mode: INSERT filter: expr: Record[1] == "XXX" i d · type: "STRING" function: "hash" index · 0 concatItems - person\_ - 0 - \_id props: name: "firstName" type: "STRING" index: 1 name: "lastName' type: "STRING" index: 2 name: "gender" type: "STRING" index: 3 nullable: true defaultValue: female

- name: "birthday"
type: "DATE"
index: 4
nullable: true
nullValue: NULL
- name: "creationDate"
type: "DATETIME"
index: 5
- name: "locationIP"
type: "STRING"
index: 6
- name: "browserUsed"
type: "STRING"
index: 7
<pre>- path: ./knows.csv</pre>
batch: 256
edges:
- name: KNOWS # person_knows_person
# mode: INSERT
# filter:
<pre># expr: Record[1] == "XXX"</pre>
src:
id:
type: "STRING"
concatItems:
- person_
- 0
id
dst:
10:
type: "STRING"
concatitems:
- person_
- 1
1a
props:
- name: "CreationDate"
index: 2
multiples true
ueraurivarue, 0000-00-00100:00:00

The configuration mainly includes the following parts:

- Specify the data source information.
- Specifies the batch size for executing statements.
- Specifies the CSV file format information.
- Specifies the schema mapping for Tag.
- Specifies the schema mapping for Edge type.

Parameter	Default value	Required	Description
sources.path sources.s3 sources.oss sources.ftp sources.sftp sources.hdfs	-	No	Specify data source information, such as source . Configure multiple sources in mu items for different data sources.
sources.batch	256	No	The batch size for executing statements manager.batch.
sources.csv.delimiter	•,	No	Specifies the delimiter for the CSV file. ( like tabs ( \t ) and hexadecimal values (e quotes, such as "\t" for tabs and "\x03" For details on escaping special characte
sources.csv.withHeader	false	No	Whether to ignore the first record in the
sources.csv.lazyQuotes	false	No	Whether to allow lazy quotes. If LazyQuote doubled quote may appear in a quoted fi
sources.tags.name	-	Yes	The tag name.
sources.tags.mode	INSERT	No	Batch operation types, including insert,
sources.tags.filter.expr	-	No	Filter the data and only import if the filt =, <, >, <= and >=. Logical operators so (Record[0] == "Mahinda" or Record[0] == "Michael
sources.tags.id.type	STRING	No	The type of the VID.
sources.tags.id.function	-	No	Functions to generate the VID. Currently
sources.tags.id.index	-	No	The column number corresponding to th this parameter must be configured.
sources.tags.id.concatItems	-	No	Used to concatenate two or more arrays for a constant, int for an index column. take effect.
sources.tags.ignoreExistedIndex	true	No	Whether to enable IGNORE_EXISTED_INDEX, th
sources.tags.props.name	-	Yes	The tag property name, which must mat
sources.tags.props.type	STRING	No	Property data type, supporting BOOL, INT GEOGRAPHY(POINT), GEOGRAPHY(LINESTRING) and G
sources.tags.props.index	-	Yes	The property corresponds to the column
sources.tags.props.nullable	false	No	Whether this prop property can be NULL,
sources.tags.props.nullValue	-	No	Ignored when nullable is false. The value when the value is equal to nullValue.
sources.tags.props.alternativeIndices	-	No	Ignored when nullable is false. The prop not equal to nullValue.
sources.tags.props.defaultValue	-	No	Ignored when nullable is false. The prop alternativeIndices are nullValue.
sources.edges.name	-	Yes	The edge type name.
sources.edges.mode	INSERT	No	Batch operation types, including insert,
sources.edges.filter.expr	-	No	<pre>Filter the data and only import if the filta =, &lt;, &gt;, &lt;= and &gt;=. Logical operators su (Record[0] == "Mahinda" or Record[0] == "Michael</pre>
sources.edges.src.id.type	STRING	No	The data type of the VID at the starting

Parameter	Default value	Required	Description
sources.edges.src.id.index	-	Yes	The column number in the data file corre
sources.edges.dst.id.type	STRING	No	The data type of the VID at the destinati
sources.edges.dst.id.index	-	Yes	The column number in the data file corre
sources.edges.rank.index	-	No	The column number in the data file corre
sources.edges.ignoreExistedIndex	true	No	Whether to enable IGNORE_EXISTED_INDEX, th
sources.edges.props.name	-	No	The edge type property name, which mu
sources.edges.props.type	STRING	No	Property data type, supporting BOOL, INT GEOGRAPHY(POINT), GEOGRAPHY(LINESTRING) and G
sources.edges.props.index	-	No	The property corresponds to the column
sources.edges.props.nullable	-	No	Whether this prop property can be NULL,
sources.edges.props.nullValue	-	No	Ignored when nullable is false. The value when the value is equal to nullValue.
sources.edges.props.defaultValue	-	No	Ignored when nullable is false. The prop alternativeIndices are nullValue.

#### Q Note

The sequence numbers of the columns in the CSV file start from 0, that is, the sequence numbers of the first column are 0, and the sequence numbers of the second column are 1.

Last update: October 25, 2023

# 11.3 NebulaGraph Exchange

### 11.3.1 Introduction

### What is NebulaGraph Exchange

NebulaGraph Exchange (Exchange) is an Apache Spark<sup>™</sup> application for bulk migration of cluster data to NebulaGraph in a distributed environment, supporting batch and streaming data migration in a variety of formats.

Exchange consists of Reader, Processor, and Writer. After Reader reads data from different sources and returns a DataFrame, the Processor iterates through each row of the DataFrame and obtains the corresponding value based on the mapping between fields in the configuration file. After iterating through the number of rows in the specified batch, Writer writes the captured data to the NebulaGraph at once. The following figure illustrates the process by which Exchange completes the data conversion and migration.



### EDITIONS

Exchange has two editions, the Community Edition and the Enterprise Edition. The Community Edition is open source developed on GitHub. The Enterprise Edition supports not only the functions of the Community Edition but also adds additional features. For details, see Comparisons.
#### SCENARIOS

Exchange applies to the following scenarios:

- Streaming data from Kafka and Pulsar platforms, such as log files, online shopping data, activities of game players, information on social websites, financial transactions or geospatial services, and telemetry data from connected devices or instruments in the data center, are required to be converted into the vertex or edge data of the property graph and import them into the NebulaGraph database.
- Batch data, such as data from a time period, needs to be read from a relational database (such as MySQL) or a distributed file system (such as HDFS), converted into vertex or edge data for a property graph, and imported into the NebulaGraph database.
- A large volume of data needs to be generated into SST files that NebulaGraph can recognize and then imported into the NebulaGraph database.
- The data saved in NebulaGraph needs to be exported.

#### (s) ....terpriseonly

Exporting the data saved in NebulaGraph is supported by Exchange Enterprise Edition only.

#### ADVANTAGES

Exchange has the following advantages:

- High adaptability: It supports importing data into the NebulaGraph database in a variety of formats or from a variety of sources, making it easy to migrate data.
- SST import: It supports converting data from different sources into SST files for data import.
- SSL encryption: It supports establishing the SSL encryption between Exchange and NebulaGraph to ensure data security.
- Resumable data import: It supports resumable data import to save time and improve data import efficiency.

# Note

Resumable data import is currently supported when migrating Neo4j data only.

- Asynchronous operation: An insert statement is generated in the source data and sent to the Graph service. Then the insert operation is performed.
- Great flexibility: It supports importing multiple Tags and Edge types at the same time. Different Tags and Edge types can be from different data sources or in different formats.
- Statistics: It uses the accumulator in Apache Spark<sup>™</sup> to count the number of successful and failed insert operations.
- Easy to use: It adopts the Human-Optimized Config Object Notation (HOCON) configuration file format and has an objectoriented style, which is easy to understand and operate.

## VERSION COMPATIBILITY

Exchange supports Spark versions 2.2.x, 2.4.x, and 3.x.x, which are named nebula-exchange\_spark\_2.2, nebula-exchange\_spark\_2.4, and nebula-exchange\_spark\_3.0 for different Spark versions.

The correspondence between the NebulaGraph Exchange version (the JAR version), the NebulaGraph core version and the Spark version is as follows.

Exchange version	NebulaGraph version	Spark version
nebula-exchange_spark_3.0-3.0-SNAPSHOT.jar	nightly	3.3.x \ 3.2.x \ 3.1.x \ 3.0.x
nebula-exchange_spark_2.4-3.0-SNAPSHOT.jar	nightly	2.4.x
nebula-exchange_spark_2.2-3.0-SNAPSHOT.jar	nightly	2.2.x
nebula-exchange_spark_3.0-3.4.0.jar	3.x.x	3.3.x \ 3.2.x \ 3.1.x \ 3.0.x
nebula-exchange_spark_2.4-3.4.0.jar	3.x.x	2.4.x
nebula-exchange_spark_2.2-3.4.0.jar	3.x.x	2.2.x
nebula-exchange_spark_3.0-3.3.0.jar	3.x.x	3.3.x \ 3.2.x \ 3.1.x \ 3.0.x
nebula-exchange_spark_2.4-3.3.0.jar	3.x.x	2.4.x
nebula-exchange_spark_2.2-3.3.0.jar	3.x.x	2.2.x
nebula-exchange_spark_3.0-3.0.0.jar	3.x.x	3.3.x \ 3.2.x \ 3.1.x \ 3.0.x
nebula-exchange_spark_2.4-3.0.0.jar	3.x.x	2.4.x
nebula-exchange_spark_2.2-3.0.0.jar	3.x.x	2.2.x
nebula-exchange-2.6.3.jar	2.6.1 \ 2.6.0	2.4.x
nebula-exchange-2.6.2.jar	2.6.1 \ 2.6.0	2.4.x
nebula-exchange-2.6.1.jar	2.6.1 \ 2.6.0	2.4.x
nebula-exchange-2.6.0.jar	2.6.1 \ 2.6.0	2.4.x
nebula-exchange-2.5.2.jar	2.5.1 \ 2.5.0	2.4.x
nebula-exchange-2.5.1.jar	2.5.1 \ 2.5.0	2.4.x
nebula-exchange-2.5.0.jar	2.5.1 \ 2.5.0	2.4.x
nebula-exchange-2.1.0.jar	2.0.1 \ 2.0.0	2.4.x
nebula-exchange-2.0.1.jar	2.0.1 \ 2.0.0	2.4.x
nebula-exchange-2.0.0.jar	2.0.1 \ 2.0.0	2.4.x

## DATA SOURCE

Exchange 3.6.1 supports converting data from the following formats or sources into vertexes and edges that NebulaGraph can recognize, and then importing them into NebulaGraph in the form of nGQL statements:

- Data stored in HDFS or locally:
- Apache Parquet
- Apache ORC
- JSON
- CSV
- Apache HBase™
- Data repository:
- Hive
- MaxCompute
- Graph database: Neo4j (Client version 2.4.5-M1)
- Relational database:
- MySQL
- PostgreSQL
- Oracle
- Column database: ClickHouse
- Stream processing software platform: Apache Kafka®
- Publish/Subscribe messaging platform: Apache Pulsar 2.4.5
- JDBC

In addition to importing data as nGQL statements, Exchange supports generating SST files for data sources and then importing SST files via Console.

RELEASE NOTE

Release

Last update: October 25, 2023

# Limitations

This topic describes some of the limitations of using Exchange 3.x.

## ENVIRONMENT

Exchange 3.x supports the following operating systems:

- CentOS 7
- macOS

SOFTWARE DEPENDENCIES

To ensure the healthy operation of Exchange, ensure that the following software has been installed on the machine:

- Java version 1.8
- Scala version 2.10.7, 2.11.12, or 2.12.10
- Apache Spark. The requirements for Spark versions when using Exchange to export data from data sources are as follows. In the following table, Y means that the corresponding Spark version is supported, and N means not supported.

# Note

Use the correct Exchange JAR file based on the Spark version. For example, for Spark version 2.4, use nebula-exchange\_spark\_2.4-3.6.1.jar.

Data source	Spark 2.2	Spark 2.4	Spark 3
CSV file	Y	Ν	Y
JSON file	Y	Υ	Y
ORC file	Υ	Υ	Y
Parquet file	Υ	Υ	Y
HBase	Y	Υ	Y
MySQL	Y	Υ	Y
PostgreSQL	Y	Υ	Y
Oracle	Y	Υ	Y
ClickHouse	Y	Υ	Y
Neo4j	Ν	Υ	Ν
Hive	Y	Υ	Y
MaxCompute	Ν	Υ	Ν
Pulsar	Ν	Υ	Untested
Kafka	Ν	Υ	Untested
NebulaGraph	Ν	Υ	Ν

Hadoop Distributed File System (HDFS) needs to be deployed in the following scenarios:

• Migrate HDFS data

• Generate SST files

Last update: October 25, 2023

# 11.3.2 Get Exchange

This topic introduces how to get the JAR file of NebulaGraph Exchange.

# Download the JAR file directly

The JAR file of Exchange Community Edition can be downloaded directly.

To download Exchange Enterprise Edition, contact us.

# Get the JAR file by compiling the source code

You can get the JAR file of Exchange Community Edition by compiling the source code. The following introduces how to compile the source code of Exchange.

Sector States

You can get Exchange Enterprise Edition in NebulaGraph Enterprise Edition Package only.

#### PREREQUISITES

- Install Maven.
- Install the correct version of Apache Spark. Exporting data from different sources requires different Spark versions. For more information, see Software dependencies.

# Steps

1. Clone the repository nebula-exchange in the / directory.

git clone -b release-3.6 https://github.com/vesoft-inc/nebula-exchange.git

2. Switch to the directory nebula-exchange.

#### cd nebula-exchange

- 3. Package NebulaGraph Exchange. Run the following command based on the Spark version:
- For Spark 2.2:

```
mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true \
-pl nebula-exchange_spark_2.2 - am -Pscala-2.11 -Pspark-2.2
```

• For Spark 2.4:

mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true \
-pl nebula-exchange\_spark\_2.4 -am -Pscala-2.11 -Pspark-2.4

• For Spark 3.0:

mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true \
-pl nebula-exchange\_spark\_3.0 -am -Pscala-2.12 -Pspark-3.0

After the compilation is successful, you can find the nebula-exchange\_spark\_x.x-release-3.6.jar file in the nebula-exchange\_spark\_x.x/target/ directory. x.x indicates the Spark version, for example, 2.4.

#### Q Note

The JAR file version changes with the release of the NebulaGraph Java Client. Users can view the latest version on the Releases page.

When migrating data, you can refer to configuration file target/classes/application.conf.

FAILED TO DOWNLOAD THE DEPENDENCY PACKAGE

If downloading dependencies fails when compiling:

- Check the network settings and ensure that the network is normal.
- Modify the mirror part of Maven installation directory libexec/conf/settings.xml:

<mirror> <id>alimaven</id> <mirror0f>central</mirror0f> <name>aliyun maven</name> <url>http://maven.aliyun.com/nexus/content/repositories/central/</url> </mirror>

Last update: December 14, 2023

# 11.3.3 Exchange configurations

# **Options for import**

After editing the configuration file, run the following commands to import specified source data into the NebulaGraph database.

## IMPORT DATA

<spark\_install\_path>/bin/spark-submit --master "spark://HOST:PORT" --class com.vesoft.nebula.exchange <nebula-exchange-2.x.y.jar\_path> -c <application.conf\_path>

Note
If the value of the properties contains Chinese characters, the encoding error may appear. Please add the following options when
submitting the Spark task:

--conf spark.driver.extraJavaOptions=-Dfile.encoding=utf-8 --conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8

# The following table lists command parameters.

Parameter	Required	Default value	Description	
class	Yes	-	Specify the main class of the driver.	
master	Yes	-	Specify the URL of the master process in a Spark cluster. For more information, see master-urls. Optional values are: tocal : Local Mode. Run Spark applications on a single thread. Suitable for importing small data sets in a test environment. yarn : Run Spark applications on a YARN cluster. Suitable for importing large data sets in a production environment. spark:// HOST:PORT : Connect to the specified Spark standalone cluster. mesos:// HOST:PORT : Connect to the specified Mesos cluster. k8s://HOST:PORT : Connect to the specified Kubernetes cluster.	
-c/config	Yes	-	Specify the path of the configuration file.	
-h/hive	No	false	Specify whether importing Hive data is supported.	
-D/dry	No	false	Specify whether to check the format of the configuration file. This parameter is used to check the format of the configuration file only, it does not check the validity of tags and edges configurations and does not import data. Don't add this parameter if you need to import data.	
-r/reload	No	-	Specify the path of the reload file that needs to be reloaded.	

For more Spark parameter configurations, see Spark Configuration.

# The version number of a JAR file is subject to the name of the JAR file that is actually compiled. If users use the yarn mode to submit a job, see the following command, especially the two '--conf' commands in the example. SSPARK\_HOME/bin/spark-submit --master yarn ( --class com.vesoft.mebula.exchange.Exchange \ --files application.conf \ --conf spark.driver.extraClassPath=./ ( --conf spa

IMPORT THE RELOAD FILE

If some data fails to be imported during the import, the failed data will be stored in the reload file. Use the parameter -r to import the data in reload file.

<spark\_install\_path>/bin/spark-submit --master "spark://HOST:PORT" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.x.y.jar\_path> -c <application.conf\_path> -r "<reload\_file\_path>"

If the import still fails, go to Official Forum for consultation.

Last update: December 18, 2023

# Parameters in the configuration file

This topic describes how to automatically generate a template configuration file when users use NebulaGraph Exchange, and introduces the configuration file application.conf.

GENERATE TEMPLATE CONFIGURATION FILE AUTOMATICALLY

Specify the data source to be imported with the following command to get the template configuration file corresponding to the data source.

java -cp <exchange\_jar\_package> com.vesoft.exchange.common.GenerateConfigTemplate -s <source\_type> -p <config\_file\_save\_path>

# Example:

java -cp nebula-exchange\_spark\_2.4-3.0-SNAPSHOT.jar com.vesoft.exchange.common.GenerateConfigTemplate -s csv -p /home/nebula/csv\_application.conf

#### CONFIGURATION INSTRUCTIONS

Before configuring the application.conf file, it is recommended to copy the file name application.conf and then edit the file name according to the file type of a data source. For example, change the file name to csv\_application.conf if the file type of the data source is CSV.

The application.conf file contains the following content types:

- Spark configurations
- Hive configurations (optional)
- NebulaGraph configurations
- Vertex configurations
- Edge configurations

# Spark configurations

This topic lists only some Spark parameters. For more information, see Spark Configuration.

Parameter	Туре	Default value	Required	Description
<pre>spark.app.name</pre>	string	-	No	The drive name in Spark.
spark.driver.cores	int	1	No	The number of CPU cores used by a driver, only applicable to a cluster mode.
spark.driver.maxResultSize	string	16	No	The total size limit (in bytes) of the serialized results of all partitions in a single Spark operation (such as collect). The minimum value is 1M, and 0 means unlimited.
spark.executor.memory	string	16	No	The amount of memory used by a Spark driver which can be specified in units, such as 512M or 1G.
spark.cores.max	int	16	No	The maximum number of CPU cores of applications requested across clusters (rather than from each node) when a driver runs in a coarse-grained sharing mode on a standalone cluster or a Mesos cluster. The default value is spark.deploy.defaultCores on a Spark standalone cluster manager or the value of the infinite parameter (all available cores) on Mesos.

# Hive configurations (optional)

Users only need to configure parameters for connecting to Hive if Spark and Hive are deployed in different clusters. Otherwise, please ignore the following configurations.

Paran	neter	Туре	Default value	Required	Description
hive.wa	arehouse	string	-	Yes	The warehouse path in HDFS. Enclose the path in double quotes and start with hdfs://.
hive.co	onnectionURL	string	-	Yes	The URL of a JDBC connection. For example, "jdbc:mysql:// 127.0.0.1:3306/hive_spark? characterEncoding=UTF-8".
hive.co	onnectionDriverName	string	"com.mysql.jdbc.Driver"	Yes	The driver name.
hive.co	onnectionUserName	list[string]	-	Yes	The username for connections.
hive.co	onnectionPassword	list[string]	-	Yes	The account password.

NebulaGraph configurations

Parameter	Туре	Default value	Required	Description
nebula.address.graph	list[string]	["127.0.0.1:9669"]	Yes	The addresses of all Graph services, including IPs and ports, separated by commas (,). Example: ["ip1:port1","ip2:port2","ip3:port3"].
nebula.address.meta	list[string]	["127.0.0.1:9559"]	Yes	The addresses of all Meta services, including IPs and ports, separated by commas (,). Example: ["ip1:port1","ip2:port2","ip3:port3"].
nebula.user	string	-	Yes	The username with write permissions for NebulaGraph.
nebula.pswd	string	-	Yes	The account password.
nebula.space	string	-	Yes	The name of the graph space where data needs to be imported.
nebula.ssl.enable.graph	bool	false	Yes	Enables the SSL encryption between Exchange and Graph services. If the value is true, the SSL encryption is enabled and the following SSL parameters take effect. If Exchange is run on a multi-machine cluster, you need to store the corresponding files in the same path on each machine when setting the following SSL-related paths.
nebula.ssl.sign	string	са	Yes	Specifies the SSL sign. Optional values are ca and self.
nebula.ssl.ca.param.caCrtFilePath	string	Specifies the storage path of the CA certificate. It takes effect when the value of nebula.ssl.sign is ca.		
nebula.ssl.ca.param.crtFilePath	string	"/path/ crtFilePath"	Yes	Specifies the storage path of the CRT certificate. It takes effect when the value of nebula.ssl.sign is ca.
nebula.ssl.ca.param.keyFilePath	string	"/path/ keyFilePath"	Yes	Specifies the storage path of the key file. It takes effect when the value of nebula.ssl.sign is ca.
nebula.ssl.self.param.crtFilePath	string	"/path/ crtFilePath"	Yes	Specifies the storage path of the CRT certificate. It takes effect when the value of nebula.ssl.sign is self.
nebula.ssl.self.param.keyFilePath	string	"/path/ keyFilePath"	Yes	Specifies the storage path of the key file. It takes effect when the value of nebula.ssl.sign is self.
nebula.ssl.self.param.password	string	"nebula"	Yes	Specifies the storage path of the password. It takes effect when the value of nebula.ssl.sign is self.
nebula.path.local	string	"/tmp"	No	

Parameter	Туре	Default value	Required	<b>Description</b> The local SST file path which needs to be set when users import SST files.
nebula.path.remote	string	"/sst"	No	The remote SST file path which needs to be set when users import SST files.
nebula.path.hdfs.namenode	string	"hdfs://name_node: 9000"	No	The NameNode path which needs to be set when users import SST files.
nebula.connection.timeout	int	3000	No	The timeout set for Thrift connections. Unit: ms.
nebula.connection.retry	int	3	No	Retries set for Thrift connections.
nebula.execution.retry	int	3	No	Retries set for executing nGQL statements.
nebula.error.max	int	32	No	The maximum number of failures during the import process. When the number of failures reaches the maximum, the Spark job submitted will stop automatically .
nebula.error.output	string	/tmp/errors	No	The path to output error logs. Failed nGQL statement executions are saved in the error log.
nebula.rate.limit	int	1024	No	The limit on the number of tokens in the token bucket when importing data.
nebula.rate.timeout	int	1000	No	The timeout period for getting tokens from a token bucket. Unit: milliseconds.

#### Q Note

NebulaGraph doesn't support vertices without tags by default. To import vertices without tags, enable vertices without tags in the NebulaGraph cluster and then add parameter nebula.enableTagless to the Exchange configuration with the value true. For example:

nebula: {
 address:{
 graph:["127.0.0.1:9669"]
 meta:["127.0.0.1:9559"]
 }
 user: root
 pswd: nebula
 space: test
 enableTagless: true
 ......
}

Vertex configurations

For different data sources, the vertex configurations are different. There are many general parameters and some specific parameters. General parameters and specific parameters of different data sources need to be configured when users configure vertices.

# General parameters

Parameter	Туре	Default value	Required	Description
tags.name	string	-	Yes	The tag name defined in NebulaGraph.
tags.type.source	string	-	Yes	Specify a data source. For example, csv.
tags.type.sink	string	client	Yes	Specify an import method. Optional values are client and $\ensuremath{SST}$ .
tags.writeMode	string	INSERT	No	Types of batch operations on data, including batch inserts, updates, and deletes. Optional values are INSERT, UPDATE, DELETE.
tags.deleteEdge	string	false	No	Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when tags.writeMode is DELETE.
tags.fields	list[string]	-	Yes	The header or column name of the column corresponding to properties. If there is a header or a column name, please use that name directly. If a CSV file does not have a header, use the form of [_c0, _c1, _c2] to represent the first column, the second column, the third column, and so on.
tags.nebula.fields	list[string]	-	Yes	Property names defined in NebulaGraph, the order of which must correspond to tags.fields. For example, [_c1, _c2] corresponds to [name, age], which means that values in the second column are the values of the property name, and values in the third column are the values of the property age.
tags.vertex.field	string	-	Yes	The column of vertex IDs. For example, when a CSV file has no header, users can use _c0 to indicate values in the first column are vertex IDs.
tags.vertex.udf.separator	string	-	No	Support merging multiple columns by custom rules. This parameter specifies the join character.
tags.vertex.udf.oldColNames	list	-	No	Support merging multiple columns by custom rules. This parameter specifies the names of the columns to be merged. Multiple columns are separated by commas.
tags.vertex.udf.newColName	string	-	No	Support merging multiple columns by custom rules. This parameter specifies the new column name.
tags.vertex.prefix	string	-	No	Add the specified prefix to the VID. For example, if the VID is 12345, adding the prefix tag1 will result in tag1_12345. The underscore cannot be modified.
tags.vertex.policy	string	-	No	Supports only the value hash . Performs hashing operations on VIDs of type string.
tags.batch	int	256	Yes	The maximum number of vertices written into NebulaGraph in a single batch.

	Parameter	Ту	уре	Default value	Rec	quired	Description
	tags.partition	in	t	32	Yes		The number of partitions to be created when the data is written to NebulaGraph. If tags.partition $\leq 1$ , the number of partitions to be created in NebulaGraph is the same as that in the data source.
Spec	cific parameters of 1	Parquet/JSON	/ORC data so	urces			
	Parameter	Туре	Default value		Required	1	Description
	tags.path	string	-		Yes	1	The path of vertex data files in HDFS. Enclose the path in double quotes and start with $hdfs://$ .
Spec	cific parameters of (	CSV data sour	rces				
	Parameter	Туре	Default value	Requi	red	Descrip	tion
	tags.path	string	-	Yes		The path double q	n of vertex data files in HDFS. Enclose the path in guotes and start with hdfs://.
	tags.separator	string	,	Yes		The sepa characte ASCII oc for the c UNICOD characte hexadeci	arator. The default value is a comma (,). For special ers, such as the control character ^A, you can use stal \001 or UNICODE encoded hexadecimal \u0001, ontrol character ^B, use ASCII octal \002 or DE encoded hexadecimal \u0002, for the control er ^C, use ASCII octal \003 or UNICODE encoded imal \u0003.
	tags.header	bool	true	Yes		Whether	the file has a header.
Spec	cific parameters of 1	Hive data sou	rces				
	Parameter	Туре	Defaul <sup>:</sup> value	t	Required	l	Description
	tags.exec	string	-		Yes		The statement to query data sources. For example, select name,age from mooc.users.
Spec	ific parameters of 1	MaxCompute	data sources				
	Parameter	Туре	Defa value	ult 9	Required	d	Description
	tags.table	string	-		Yes		The table name of the MaxCompute.
	tags.project	string	-		Yes		The project name of the MaxCompute.
	tags.odpsUrl	string	-		Yes		The odpsUrl of the MaxCompute service. For more information about odpsUrl, see Endpoints.
	tags.tunnelUrl	string	-		Yes		The tunnelUrl of the MaxCompute service. For more information about tunnelUrl, see Endpoints.
	tags.accessKeyId	string	-		Yes		The accessKeyId of the MaxCompute service.
	tags.accessKeySecret	string	-		Yes		The accessKeySecret of the MaxCompute service.
	tags.partitionSpec	string	-		No		Partition descriptions of MaxCompute tables.
	tags.sentence	string	-		No		Statements to query data sources. The table name in the SQL statement is the same as the value of the table above.

# Specific parameters of Neo4j data sources

Parameter	Туре	Default value	Required	Description
tags.exec	string	-	Yes	Statements to query data sources. For example: match (n:label) return n.neo4j-field-0.
tags.server	string	"bolt:// 127.0.0.1:7687"	Yes	The server address of Neo4j.
tags.user	string	-	Yes	The Neo4j username with read permissions.
tags.password	string	-	Yes	The account password.
tags.database	string	-	Yes	The name of the database where source data is saved in Neo4j.
tags.check_point_path	string	/tmp/test	No	The directory set to import progress information, which is used for resuming transfers. If not set, the resuming transfer is disabled.

# Specific parameters of MySQL/PostgreSQL data sources

Parameter	Туре	Default value	Required	Description
tags.host	string	-	Yes	The MySQL/PostgreSQL server address.
tags.port	string	-	Yes	The MySQL/PostgreSQL server port.
tags.database	string	-	Yes	The database name.
tags.table	string	-	Yes	The name of a table used as a data source.
tags.user	string	-	Yes	The MySQL/PostgreSQL username with read permissions.
tags.password	string	-	Yes	The account password.
tags.sentence	string	-	Yes	Statements to query data sources. For example: "select teamid, name from team order by teamid".

# Specific parameters of Oracle data sources

Parameter	Туре	Default value	Required	Description
tags.url	string	-	Yes	The Oracle server address.
tags.driver	string	-	Yes	The Oracle driver address.
tags.user	string	-	Yes	The Oracle username with read permissions.
tags.password	string	-	Yes	The account password.
tags.table	string	-	Yes	The name of a table used as a data source.
tags.sentence	string	-	Yes	Statements to query data sources. For example: "select playerid, name, age from player".

# Specific parameters of ClickHouse data sources

	Parameter	Туре	Default value	Required	Description		
	tags.url	string	-	Yes	The JDBC URL of ClickHouse.		
	tags.user	string	-	Yes	The ClickHouse username with read permissions.		
	tags.password	string	-	Yes	The account password.		
	tags.numPartition	string	-	Yes	The number of ClickHouse partitions.		
	tags.sentence	string	-	Yes	Statements to query data sources.		
Spe	Specific parameters of Hbase data sources						

Parameter	Туре	Default value	Required	Description
tags.host	string	127.0.0.1	Yes	The Hbase server address.
tags.port	string	2181	Yes	The Hbase server port.
tags.table	string	-	Yes	The name of a table used as a data source.
tags.columnFamily	string	-	Yes	The column family to which a table belongs.

# Specific parameters of Pulsar data sources

Parameter	Туре	Default value	Required	Description
tags.service	string	"pulsar:// localhost: 6650"	Yes	The Pulsar server address.
tags.admin	string	"http:// localhost: 8081"	Yes	The admin URL used to connect pulsar.
<pre>tags.options.<topic topics  topicspattern=""></topic topics ></pre>	string	-	Yes	Options offered by Pulsar, which can be configured by choosing one from topic, topics, and topicsPattern.
tags.interval.seconds	int	10	Yes	The interval for reading messages. Unit: seconds.

# Specific parameters of Kafka data sources

Parameter	Туре	Default value	Required	Description
tags.service	string	-	Yes	The Kafka server address.
tags.topic	string	-	Yes	The message type.
tags.interval.seconds	int	10	Yes	The interval for reading messages. Unit: seconds.

# Specific parameters for generating SST files

Parameter	Туре	Default value	Required	Description
tags.path	string	-	Yes	The path of the source file specified to generate SST files.
tags.repartitionWithNebula	bool	true	No	Whether to repartition data based on the number of partitions of graph spaces in NebulaGraph when generating the SST file. Enabling this function can reduce the time required to DOWNLOAD and INGEST SST files.

# Edge configurations

For different data sources, configurations of edges are also different. There are general parameters and some specific parameters. General parameters and specific parameters of different data sources need to be configured when users configure edges.

For the specific parameters of different data sources for edge configurations, please refer to the introduction of specific parameters of different data sources above, and pay attention to distinguishing tags and edges.

# General parameters

Parameter	Туре	Default value	Required	Description
edges.name	string	-	Yes	The edge type name defined in NebulaGraph.
edges.type.source	string	-	Yes	The data source of edges. For example, $\ \mbox{csv}$ .
edges.type.sink	string	client	Yes	The method specified to import data. Optional values are client and $\ensuremath{SST}$ .
edges.writeMode	string	INSERT	No	Types of batch operations on data, including batch inserts, updates, and deletes. Optional values are INSERT , UPDATE , DELETE .
edges.fields	list[string]	-	Yes	The header or column name of the column corresponding to properties. If there is a header or column name, please use that name directly. If a CSV file does not have a header, use the form of [_c0, _c1, _c2] to represent the first column, the second column, the third column, and so on.
edges.nebula.fields	list[string]	-	Yes	Edge names defined in NebulaGraph, the order of which must correspond to edges.fields. For example, [_c2, _c3] corresponds to [start_year, end_year], which means that values in the third column are the values of the start year, and values in the fourth column are the values of the end year.
edges.source.field	string	-	Yes	The column of source vertices of edges. For example, _c0 indicates a value in the first column that is used as the source vertex of an edge.
edges.source.prefix	string	-	No	Add the specified prefix to the VID. For example, if the VID is 12345, adding the prefix tag1 will result in tag1_12345. The underscore cannot be modified.
edges.source.policy	string	-	No	Supports only the value hash . Performs hashing operations on VIDs of type string.
edges.target.field	string	-	Yes	The column of destination vertices of edges. For example, _c0 indicates a value in the first column that is used as the destination vertex of an edge.
edges.target.prefix	string	-	No	Add the specified prefix to the VID. For example, if the VID is 12345, adding the prefix tag1 will result in tag1_12345. The underscore cannot be modified.
edges.target.policy	string	-	No	Supports only the value hash . Performs hashing operations on VIDs of type string.
edges.ranking	int	-	No	The column of rank values. If not specified, all rank values are 0 by default.
edges.batch	int	256	Yes	The maximum number of edges written into NebulaGraph in a single batch.
edges.partition	int	32	Yes	The number of partitions to be created when the data is written to NebulaGraph. If $edges.partition \leq 1$ , the number of partitions to be created in NebulaGraph is the same as that in the data source.

# Specific parameters for generating SST files

Parameter	Туре	Default value	Required	Description
edges.path	string	-	Yes	The path of the source file specified to generate SST files.
edges.repartitionWithNebula	bool	true	No	Whether to repartition data based on the number of partitions of graph spaces in NebulaGraph when generating the SST file. Enabling this function can reduce the time required to DOWNLOAD and INGEST SST files.

Last update: December 7, 2023

# 11.3.4 Use NebulaGraph Exchange

# Import data from CSV files

This topic provides an example of how to use Exchange to import NebulaGraph data stored in HDFS or local CSV files.

DATA SET

This topic takes the basketballplayer dataset as an example.

# ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- Hadoop: 2.9.2, pseudo-distributed deployment
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

# PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running normally.
- If files are stored locally and NebulaGraph is a cluster architecture, you need to place the files in the same directory locally on each machine in the cluster.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>						
## Use the graph space basketballplayer. nebula> USE basketballplayer;						
## Create the Tag player. nebula> CREATE TAG player(name string, age int);						
<pre>## Create the Tag team. nebula&gt; CREATE TAG team(name string);</pre>						
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>						
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>						

For more information, see Quick start workflow.

Step 2: Process CSV files

Confirm the following information:

1. Process CSV files to meet Schema requirements.

Note	
Exchange supports uploading CSV files with or without headers.	

## 2. Obtain the CSV file storage path.

Step 3: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set CSV data source configuration. In this example, the copied file is called csv\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
    # Spark configuration
    spark: {
        app: {
            name: NebulaGraph Exchange 3.6.1
        }
        driver: {
            cores: 1
            maxResultSize: 16
        }
        executor: {
            memory:16
        }
        cores: {
    }
}
```

max: 16 } } # NebulaGraph configuration nebula: { address:{ # Specify the IP addresses and ports for Graph and Meta services. # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
# Addresses are separated by commas. graph:["127.0.0.1:9669"] # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta. meta:["127.0.0.1:9559"] } # The account entered must have write permission for the NebulaGraph space. user: root pswd: nebula # Fill in the name of the graph space you want to write data to in the NebulaGraph. space: basketballplayer connection: { timeout: 3000 retry: 3 execution: { retry: 3 error: { max· 32 output: /tmp/errors rate: { limit: 1024 timeout: 1000 } } # Processing vertexes tags: [ # Set the information about the Tag player. { # Specify the Tag name defined in NebulaGraph. name: player type: { # Specify the data source file format to CSV. source: csv # Specify how to import the data into NebulaGraph: Client or SST. sink: client 1 # Specify the path to the CSV file. # If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example: "hdfs://ip:port/xx/xx". # If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example: "file:///tmp/xx.csv". path: "hdfs://192.168.\*.\*:9000/data/vertex\_player.csv" # If the CSV file does not have a header, use [\_c0, \_c1, \_c2, ..., \_cn] to represent its header and indicate the columns as the source of the property values. # If the CSV file has headers, use the actual column names. fields: [\_c1, \_c2] # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph. # The sequence of fields and nebula.fields must correspond to each other. nebula.fields: [age, name] # Specify a column of data in the table as the source of vertex VID in the NebulaGraph. # The value of vertex must be the same as the column names in the above fields or csv.fields. # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID. vertex: { field:\_c0 # udf:{ separator:"\_" oldColNames: [field-0, field-1, field-2] newColName:new-field # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tag1' will result in 'tag1\_12345'. The underscore cannot be modified. # prefix:"tag1" # Performs hashing operations on VIDs of type string. # policy:hash # The delimiter specified. The default value is comma. separator: ", # If the CSV file has a header, set the header to true. # If the CSV file does not have a header, set the header to false. The default value is false. header: false # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT. #writeMode: INSERT # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'. #deleteEdge: false

```
# The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
   partition: 32
  }
  # Set the information about the Tag Team.
    name: team
    type: {
      source: csv
     sink: client
    path: "hdfs://192.168.*.*:9000/data/vertex_team.csv"
   fields: [_c1]
nebula.fields: [name]
    vertex: {
     field:_c0
    }
    separator: ","
   header: false
batch: 256
   partition: 32
  }
 # If more vertexes need to be added, refer to the previous configuration to add them.
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
    # Specify the Edge Type name defined in NebulaGraph.
    name: follow
    type: {
      # Specify the data source file format to CSV.
     source: csv
      # Specify how to import the data into NebulaGraph: Client or SST.
     sink: client
    # Specify the path to the CSV file.
    # If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example: "hdfs://ip:port/xx/xx".
   # If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example: "file:///tmp/xx.csv" path: "hdfs://192.168.*.*:9000/data/edge_follow.csv"
   # If the CSV file does not have a header, use [_c0, _c1, _c2, ..., _cn] to represent its header and indicate the columns as the source of the property values.
# If the CSV file has headers, use the actual column names.
    fields: [_c2]
    # Specify the column names in the edge table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other
   nebula.fields: [degree]
    # Specify a column as the source for the source and destination vertexes.
    # The value of vertex must be the same as the column names in the above fields or csv.fields.
    # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    source: {
     field: _c0
    # udf:{
                 separator:"_"
    #
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
     field: _c1
    # udf:{
                  separator:"_"
                 oldColNames: [field-0, field-1, field-2]
                 newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tag1' will result in 'tag1_12345'. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # The delimiter specified. The default value is comma.
    separator: ",
    # Specify a column as the source of the rank (optional).
    #ranking: rank
   # If the CSV file has a header, set the header to true.
    # If the CSV file does not have a header, set the header to false. The default value is false.
```

```
header: false
      # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
      #writeMode: INSERT
      # The number of data written to NebulaGraph in a single batch.
      batch: 256
      # The number of partitions to be created when the data is written to NebulaGraph.
      partition: 32
    # Set the information about the Edge Type serve.
      name: serve
      type: {
        source: csv
        sink: client
      path: "hdfs://192.168.*.*:9000/data/edge_serve.csv"
      fields: [_c2,_c3]
nebula.fields: [start_year, end_year]
      source: {
       field: _c0
      target: {
       field: _c1
      separator: ",'
      header: false
      batch: 256
      partition: 32
    # If more edges need to be added, refer to the previous configuration to add them.
}
```

Step 4: Import data into NebulaGraph

Run the following command to import CSV data into NebulaGraph. For descriptions of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <csv\_application.conf\_path>

# Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

## For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/target/classes/csv\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

## Step 5: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 6: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

Last update: November 7, 2023

# Import data from JSON files

This topic provides an example of how to use Exchange to import NebulaGraph data stored in HDFS or local JSON files.

DATA SET

This topic takes the basketballplayer dataset as an example. Some sample data are as follows:

# • player

```
{"id":"player100", "age":42, "name":"Tim Duncan"}
{"id":"player101", "age":36, "name":"Tony Parker"}
{"id":"player102", "age":33, "name":"LaMarcus Aldridge"}
{"id":"player103", "age":32, "name":"Rudy Gay"}
....
```

• team

```
{"id":"team200","name":"Warriors"}
{"id":"team201","name":"Nuggets"}
...
```

# • follow

```
{"src":"player100","dst":"player101","degree":95}
{"src":"player101","dst":"player102","degree":90}
```

## • serve

```
{"src":"player100","dst":"team204","start_year":"1997","end_year":"2016"}
{"src":"player101","dst":"team204","start_year":"1999","end_year":"2018"}
```

#### ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- Hadoop: 2.9.2, pseudo-distributed deployment
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and NebulaGraph is a cluster architecture, you need to place the files in the same directory locally on each machine in the cluster.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	start_year int, end_year int

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

```
## Create a graph space.
nebula> CREATE SPACE basketballplayer \
    (partition_num = 10, \
    replica_factor = 1, \
    vid_type = FIXED_STRING(30));
```

## Use the graph space basketballplayer.
nebula> USE basketballplayer;

## Create the Tag player.
nebula> CREATE TAG player(name string, age int);

## Create the Tag team.
nebula> CREATE TAG team(name string);

## Create the Edge type follow.
nebula> CREATE EDGE follow(degree int);

## Create the Edge type serve.
nebula> CREATE EDGE serve(start\_year int, end\_year int);

For more information, see Quick start workflow.

Step 2: Process JSON files

Confirm the following information:

- 1. Process JSON files to meet Schema requirements.
- 2. Obtain the JSON file storage path.

Step 3: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set JSON data source configuration. In this example, the copied file is called json\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
 spark: {
   app: {
     name: NebulaGraph Exchange 3.6.1
   driver: {
     cores: 1
      maxResultSize: 1G
   executor: {
       memory:1G
   }
   cores: {
     max: 16
   }
  }
  # NebulaGraph configuration
  nebula: {
   address:{
      # Specify the IP addresses and ports for Graph and all Meta services.
      # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
# Addresses are separated by commas.
     graph:["127.0.0.1:9669"]
     # the address of any of the meta services.
      # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
    }
   # The account entered must have write permission for the NebulaGraph space.
   user: root
   pswd: nebula
    # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
      timeout: 3000
     retry: 3
    execution: {
     retry: 3
    error: {
     max: 32
     output: /tmp/errors
   rate: {
      limit: 1024
      timeout: 1000
   }
  }
  # Processing vertexes
  tags: [
   # Set the information about the Tag player.
    {
     # Specify the Tag name defined in NebulaGraph.
      name: player
      type: {
       # Specify the data source file format to JSON.
       source: json
       # Specify how to import the data into NebulaGraph: Client or SST.
       sink: client
      }
      # Specify the path to the JSON file.
      # If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://ip:port/xx/xx".
      # If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example, "file:///tmp/xx.json"
      path: "hdfs://192.168.*.*:9000/data/vertex_player.json"
      # Specify the key name in the JSON file in fields, and its corresponding value will serve as the data source for the properties specified in the NebulaGraph.
      # If multiple column names need to be specified, separate them by commas
      fields: [age,name]
      # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
      # The sequence of fields and nebula.fields must correspond to each other.
      nebula.fields: [age, name]
      # Specify a column of data in the table as the source of vertex VID in the NebulaGraph.
      # The value of vertex must be the same as that in the JSON file.
      # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
      vertex: {
       field:id
      # udf:{
```

```
separator:" "
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
    #
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: TNSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
   batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
   partition: 32
  # Set the information about the Tag Team.
    name: team
    type: {
     source: json
sink: client
    path: "hdfs://192.168.*.*:9000/data/vertex_team.json"
    fields: [name]
    nebula.fields: [name]
    vertex: {
     field:id
   batch: 256
   partition: 32
  }
 # If more vertexes need to be added, refer to the previous configuration to add them.
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
    # Specify the Edge Type name defined in NebulaGraph.
    name: follow
    type: {
      # Specify the data source file format to JSON.
     source: json
      # Specify how to import the data into NebulaGraph: Client or SST.
     sink: client
    1
    # Specify the path to the JSON file.
    # If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://ip:port/xx/xx".
   # If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example, "file:///tmp/xx.json" path: "hdfs://192.168.*.*:9000/data/edge_follow.json"
    # Specify the key name in the JSON file in fields, and its corresponding value will serve as the data source for the properties specified in the NebulaGraph.
    # If multiple column names need to be specified, separate them by commas.
    fields: [degree]
    # Specify the column names in the edge table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
   nebula.fields: [degree]
    # Specify a column as the source for the source and destination vertexes.
    # The value of vertex must be the same as that in the JSON file
    # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    source: {
     field: src
    # udf:{
                 separator:" '
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
             3
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
# Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
     field: dst
    # udf:{
                 separator:"_"
                 oldColNames: [field-0, field-1, field-2]
                 newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tag1' will result in 'tag1_12345'. The underscore cannot be modified.
    # prefix:"tag1"
```

```
# Performs hashing operations on VIDs of type string.
    # policy:hash
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
#writeMode: INSERT
    # The number of data written to NebulaGraph in a single batch.
   batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
   partition: 32
  }
  # Set the information about the Edge Type serve.
    name: serve
    type: {
      source: json
     sink: client
    path: "hdfs://192.168.*.*:9000/data/edge_serve.json"
    fields: [start_year,end_year]
    nebula.fields: [start_year, end_year]
    source: {
     field: src
    ,
target: {
     field: dst
    batch: 256
   partition: 32
# If more edges need to be added, refer to the previous configuration to add them
```

Step 4: Import data into NebulaGraph

}

Run the following command to import JSON data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <json\_application.conf\_path>

#### Q Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

#### For example:

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-echange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-
exchange/nebula-exchange/target/classes/json_application.conf
```

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 5: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 6: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

```
Last update: November 7, 2023
```

# Import data from ORC files

This topic provides an example of how to use Exchange to import NebulaGraph data stored in HDFS or local ORC files.

DATA SET

This topic takes the basketballplayer dataset as an example.

# ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- Hadoop: 2.9.2, pseudo-distributed deployment
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

## PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and NebulaGraph is a cluster architecture, you need to place the files in the same directory locally on each machine in the cluster.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	start_year int, end_year int

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer. nebula> USE basketballplayer;
## Create the Tag player. nebula> CREATE TAG player(name string, age int);
<pre>## Create the Tag team. nebula&gt; CREATE TAG team(name string);</pre>
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>

For more information, see Quick start workflow.

Step 2: Process ORC files

Confirm the following information:

- 1. Process ORC files to meet Schema requirements.
- 2. Obtain the ORC file storage path.

Step 3: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set ORC data source configuration. In this example, the copied file is called orc\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
 # Spark configuration
 spark: {
   app: {
     name: NebulaGraph Exchange 3.6.1
   driver: {
     cores: 1
     maxResultSize: 1G
   executor: {
       memory:1G
   }
   cores: {
     max: 16
   }
 }
 # NebulaGraph configuration
 nebula: {
   address:{
     # Specify the IP addresses and ports for Graph and all Meta services.
     # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port"
```

```
# Addresses are separated by commas.
    graph:["127.0.0.1:9669"]
      the address of any of the meta services.
    # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
    meta:["127.0.0.1:9559"]
  1
  # The account entered must have write permission for the NebulaGraph space.
 user: root
pswd: nebula
  # Fill in the name of the graph space you want to write data to in the NebulaGraph.
  space: basketballplayer
  connection: {
    timeout: 3000
   retry: 3
 execution: {
   retry: 3
  }
 error: {
    max: 32
   output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
 }
}
# Processing vertexes
tags: [
  # Set the information about the Tag player.
  {
    name: player
    type: {
      # Specify the data source file format to ORC.
      source: or
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink: client
    }
   # Specify the path to the ORC file.
# If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://ip:port/xx/xx".
# If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example, "file:///tmp/xx.orc".
path: "hdfs://192.168.*.*:9000/data/vertex_player.orc"
    # Specify the key name in the ORC file in fields, and its corresponding value will serve as the data source for the properties specified in the NebulaGraph.
    # If multiple values need to be specified, separate them with commas.
    fields: [age,name]
    # Specify the property names defined in NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
    nebula.fields: [age, name]
    # Specify a column of data in the table as the source of VIDs in the NebulaGraph.
    # The value of vertex must be consistent with the field in the ORC file.
# Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    vertex:
      field:id
    # udf:{
                  separator:"_"
oldColNames:[field-0,field-1,field-2]
                   newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when `writeMode` is `DELETE`.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  }
  # Set the information about the Tag team.
    name: team
    type: {
       source: orc
      sink: client
    path: "hdfs://192.168.*.*:9000/data/vertex_team.orc"
    fields: [name]
```

```
nebula.fields: [name]
    vertex:
      field:id
    batch: 256
    partition: 32
 }
 # If more vertexes need to be added, refer to the previous configuration to add them.
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
  {
    # Specify the Edge Type name defined in NebulaGraph.
    name: follow
    type: {
      # Specify the data source file format to ORC.
     source: orc
     # Specify how to import the data into NebulaGraph: Client or SST.
     sink: client
    }
    # Specify the path to the ORC file.
    # If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://ip:port/xx/xx".
# If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example, "file:///tmp/xx.orc".
    path: "hdfs://192.168.*.*:9000/data/edge_follow.orc"
    # Specify the key name in the ORC file in fields, and its corresponding value will serve as the data source for the properties specified in the NebulaGraph.
    # If multiple values need to be specified, separate them with commas.
    fields: [degree]
    # Specify the property names defined in NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
    nebula.fields: [degree]
    # Specify a column as the source for the source and destination vertexes.
    # The value of vertex must be consistent with the field in the ORC file.
# Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    source: {
     field: src
    # udf:{
                  separator:"_"
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
field: dst
    # udf:{
                  separator:" "
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
    #
              3
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
   partition: 32
  }
  # Set the information about the Edge type serve.
    name: serve
    type: {
      source: orc
     sink: client
    path: "hdfs://192.168.*.*:9000/data/edge_serve.orc"
    fields: [start_year,end_year]
    nebula.fields: [start_year, end_year]
    source: {
     field: src
    target: {
```
```
field: dst
}
batch: 256
partition: 32
}
# If more edges need to be added, refer to the previous configuration to add them.
}
```

Step 4: Import data into NebulaGraph

Run the following command to import ORC data into NebulaGraph. For a description of the parameters, see Options for import.

\${\$PARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <orc\_application.conf\_path>

Q Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/target/classes/orc\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 5: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 6: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

## Import data from Parquet files

This topic provides an example of how to use Exchange to import NebulaGraph data stored in HDFS or local Parquet files.

DATA SET

This topic takes the basketballplayer dataset as an example.

### ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- Hadoop: 2.9.2, pseudo-distributed deployment
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and NebulaGraph is a cluster architecture, you need to place the files in the same directory locally on each machine in the cluster.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property	
Tag	player	name string, age int	
Tag	team	name string	
Edge Type	follow	degree int	
Edge Type	serve	start_year int, end_year int	

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer. nebula> USE basketballplayer;
## Create the Tag player. nebula> CREATE TAG player(name string, age int);
<pre>## Create the Tag team. nebula&gt; CREATE TAG team(name string);</pre>
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>

For more information, see Quick start workflow.

Step 2: Process Parquet files

Confirm the following information:

- 1. Process Parquet files to meet Schema requirements.
- 2. Obtain the Parquet file storage path.

```
Step 3: Modify configuration files
```

After Exchange is compiled, copy the conf file target/classes/application.conf to set Parquet data source configuration. In this example, the copied file is called parquet\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
 # Spark configuration
 spark: {
   app: {
     name: NebulaGraph Exchange 3.6.1
   driver: {
     cores: 1
     maxResultSize: 1G
   }
   executor: {
       memory:1G
   }
   cores: {
     max: 16
   }
 3
 # NebulaGraph configuration
 nebula: {
   address:{
```

```
# Specify the IP addresses and ports for Graph and all Meta services.
# If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
    # Addresses are separated by commas.
    graph:["127.0.0.1:9669"]
    # the address of any of the meta services.
    # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
    meta:["127.0.0.1:9559"]
  # The account entered must have write permission for the NebulaGraph space.
  user: root
  pswd: nebula
  # Fill in the name of the graph space you want to write data to in the NebulaGraph.
  space: basketballplayer
  connection: {
    timeout: 3000
    retry: 3
  execution: {
   retry: 3
  error: {
   max: 32
    output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
  }
3
# Processing vertexes
tags: [
  # Set the information about the Tag player.
    # Specify the Tag name defined in NebulaGraph.
    name: player
    type: {
     # Specify the data source file format to Parquet.
      source: parquet
      # Specifies how to import the data into NebulaGraph: Client or SST.
      sink: client
    }
    # Specify the path to the Parquet file.
# If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://ip:port/xx/xx"
    # If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example, "file:///tmp/xx.parquet".
    path: "hdfs://192.168.*.13:9000/data/vertex plaver.parguet"
    # Specify the key name in the Parquet file in fields, and its corresponding value will serve as the data source for the properties specified in the NebulaGraph.
    # If multiple values need to be specified, separate them with commas
    fields: [age,name]
    # Specify the property name defined in NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
    nebula.fields: [age, name]
    # Specify a column of data in the table as the source of VIDs in the NebulaGraph.
    # The value of vertex must be consistent with the field in the Parquet file.
    # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    vertex: {
     field:id
    # udf:{
                 separator:"_"
oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
             }
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when `writeMode` is `DELETE`.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  1
  # Set the information about the Tag team.
    name: team
    type: {
      source: parquet
     sink: client
```

```
path: "hdfs://192.168.11.13:9000/data/vertex_team.parquet"
    fields: [name]
   nebula.fields: [name]
    vertex: {
     field:id
    1
    batch: 256
    partition: 32
 # If more vertexes need to be added, refer to the previous configuration to add them.
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
  {
    # Specify the Edge Type name defined in NebulaGraph.
    name: follow
    type: {
      # Specify the data source file format to Parquet.
     source: parquet
     # Specifies how to import the data into NebulaGraph: Client or SST.
     sink: client
    }
   # Specify the path to the Parquet file.
    # If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://ip:port/xx/xx".
   # If the file is stored locally, use double quotation marks to enclose the file path, starting with file://. For example, "file:///tmp/xx.parquet". path: "hdfs://192.168.11.13:9000/data/edge_follow.parquet"
    # Specify the key name in the Parquet file in fields, and its corresponding value will serve as the data source for the properties specified in the NebulaGraph.
# If multiple values need to be specified, separate them with commas.
    fields: [degree]
    # Specify the property name defined in NebulaGraph.
   # The sequence of fields and nebula.fields must correspond to each other.
nebula.fields: [degree]
    # Specify a column as the source for the source and destination vertexes.
    # The values of vertex must be consistent with the fields in the Parquet file.
    # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    source: {
     field: src
    # udf:{
                  separator:"_"
    #
                  oldColNames:[field-0,field-1,field-2]
    #
    #
                  newColName.new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
     field: dst
    # udf:{
                 separator:"_"
oldColNames:[field-0,field-1,field-2]
    #
                  newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # The number of data written to NebulaGraph in a single batch.
   batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
   partition: 32
  # Set the information about the Edge type serve.
    name: serve
    type: {
      source: parquet
      sink: client
    path: "hdfs://192.168.11.13:9000/data/edge_serve.parquet"
    fields: [start_year,end_year]
    nebula.fields: [start_year, end_year]
    source: {
     field: src
```



Step 4: Import data into NebulaGraph

Run the following command to import Parquet data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOWE}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange<3.6.1.jar\_path> -c <parquet\_application.conf\_path>

#### O Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

## For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/target/classes/parquet\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 5: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 6: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

## Import data from HBase

This topic provides an example of how to use Exchange to import NebulaGraph data stored in HBase.

DATA SET

This topic takes the basketballplayer dataset as an example.

In this example, the data set has been stored in HBase. All vertexes and edges are stored in the player, team, follow, and serve tables. The following are some of the data for each table.

hbase(main):002:0> scan "player" ROW player100 player100 player101 player101 player102 player102	COLUMN+CELL column=cf:age, timestamp=1618881347530, value=42 column=cf:name, timestamp=1618881354604, value=Tim Duncan column=cf:age, timestamp=1618881369124, value=36 column=cf:name, timestamp=1618881379102, value=30 column=cf:age, timestamp=1618881386987, value=31 column=cf:age, timestamp=1618881386987, value=31
player103 player103 	<pre>column=cf:name, timestamp=1618881402002, value=32 column=cf:name, timestamp=1618881407882, value=Rudy Gay</pre>
hbase(main):003:0> scan "team" ROW team200 team201 	COLUMN+CELL column=cf:name, timestamp=1618881445563, value=Warriors column=cf:name, timestamp=1618881453636, value=Nuggets
hbase(main):004:0> scan "follow" ROW player100 player101 player101 	COLUMN+CELL column=cf:degree, timestamp=1618881804853, value=95 column=cf:dst_player, timestamp=1618881791522, value=player101 column=cf:degree, timestamp=1618881824685, value=90 column=cf:dst_player, timestamp=1618881816042, value=player102
hbase(main):005:0> scan "serve" ROW player100 player100 player100 	COLUMN+CELL column=cf:end_year, timestamp=1618881899333, value=2016 column=cf:start_year, timestamp=1618881890117, value=1997 column=cf:teamid, timestamp=1618881875739, value=team204

ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- HBase: 2.2.7
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

## PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	start_year int, end_year int

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer. nebula> USE basketballplayer;
## Create the Tag player. nebula> CREATE TAG player(name string, age int);
## Create the Tag team. nebula> CREATE TAG team(name string);
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>
## Create the Edge type serve.

For more information, see Quick start workflow.

## Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set HBase data source configuration. In this example, the copied file is called hbase\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
    app: {
    name: NebulaGraph Exchange 3.6.1
    driver: {
      cores: 1
       maxResultSize: 1G
    cores: {
      max: 16
    }
  }
  # NebulaGraph configuration
  nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and all Meta services.
# If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
      # Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
# the address of any of the meta services.
      # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
meta:["127.0.0.1:9559"]
    # The account entered must have write permission for the NebulaGraph space.
    user: root
    pswd: nebula
     # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
      timeout: 3000
```

```
retry: 3
  execution: {
    retry: 3
  error: {
    max: 32
    output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
  }
# Processing vertexes
tags: [
  # Set information about Tag player.
  # If you want to set RowKey as the data source, enter rowkey and the actual column name of the column family.
  {
    # The Tag name in NebulaGraph.
    name: player
    type: {
    # Specify the data source file format to HBase.
      source: hbase
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink: client
    host:192.168.*.*
    port:2181
    table:"player"
    columnFamily:"cf"
    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
     # If multiple column names need to be specified, separate them by commas.
    fields: [age,name]
nebula.fields: [age,name]
    # Specify a column of data in the table as the source of vertex VID in the NebulaGraph.
# For example, if rowkey is the source of the VID, enter rowkey.
    vertex:{
        field:rowkev
     # udf:{
                  separator:"_"
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
              }
    # # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'.
    #deleteEdge: false
    # Number of pieces of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  # Set Tag Team information.
  {
    name: team
    type: {
   source: hbase
     sink: client
    host:192.168.*.*
    port:2181
    table:"team"
    columnFamily:"cf"
    fields: [name]
    nebula.fields: [name]
    vertex:{
        field:rowkey
    batch: 256
    partition: 32
  }
1
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
  {
```

```
# The corresponding Edge Type name in NebulaGraph.
name: follow
```

```
type: {
    # Specify the data source file format to HBase.
    source: hbase
    # Specify how to import the Edge type data into NebulaGraph.
# Specify how to import the data into NebulaGraph: Client or SST.
    sink: client
  }
  host:192.168.*.*
  port:2181
  table: "follow"
  columnFamily:"cf"
  # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the NebulaGraph.
# The sequence of fields and nebula.fields must correspond to each other.
  # If multiple column names need to be specified, separate them by commas.
  fields: [degree]
  nebula.fields: [degree]
  # In source, use a column in the follow table as the source of the edge's source vertex.
# In target, use a column in the follow table as the source of the edge's destination vertex.
  source:{
      field:rowkey
  # udf:{
                 separator:"_"
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
            }
  #
  # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
  # prefix:"tag1"
# Performs hashing operations on VIDs of type string.
  # policy:hash
  }
  target:{
       field:dst_player
  # udf:{
                 separator:"_"
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
  #
            }
  # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
  # Performs hashing operations on VIDs of type string.
  # policy:hash
  # (Optional) Specify a column as the source of the rank.
  #ranking: rank
  # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
  #writeMode: INSERT
  # The number of data written to NebulaGraph in a single batch.
  batch: 256
  # The number of partitions to be created when the data is written to NebulaGraph.
  partition: 32
1
# Set the information about the Edge Type serve.
{
  name: serve
  type: {
source: hbase
   sink: client
  ,
host:192.168.*.*
  port:2181
  table:"serve'
  columnFamily:"cf"
  fields: [start_year,end_year]
  nebula.fields: [start_year,end_year]
  source:{
      field:rowkey
  }
  target:{
      field:teamid
  }
  \ensuremath{\texttt{\#}} (Optional) Specify a column as the source of the rank.
  #ranking: rank
 batch: 256
 partition: 32
}
```

] } Step 3: Import data into NebulaGraph

Run the following command to import HBase data into NebulaGraph. For descriptions of the parameters, see Options for import.

\${SPARK\_HOWE}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <hbase\_application.conf\_path>

#### Q Note

JAR packages are available in two ways: compiled them yourself, or download the compiled ...jar file directly.

### For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/nebula-exchange/target/classes/hbase\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

#### Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

## Import data from MySQL/PostgreSQL

This topic provides an example of how to use Exchange to export MySQL data and import to NebulaGraph. It also applies to exporting data from PostgreSQL into NebulaGraph.

### DATA SET

This topic takes the basketballplayer dataset as an example.

In this example, the data set has been stored in MySQL. All vertexes and edges are stored in the player, team, follow, and serve tables. The following are some of the data for each table.

mysql> desc pl	.ayer;		-		
Field   T	ype	Null	Key   [	Default	Extra
playerid   v   age   i   name   v	varchar(30)   nt   varchar(30)	YES   YES   YES	         	IULL   IULL   IULL	     
mysql> desc te	am;	+	+	+	+
Field   Typ	ie   N +	ull   Ke	ey   Det	fault   Ex	tra
teamid   var   name   var ++	char(30)   Y char(30)   Y	'ES   'ES   +	NUI   NUI	.L   .L   +	   +
mysql> desc fo	llow;	-+	.++		+
Field	Туре	Null	Key	Default	Extra
src_player     dst_player     degree	varchar(30) varchar(30) int	YES   YES   YES		NULL NULL NULL	   
mysql> desc se	erve;	-+	+		+
Field	Туре	Null	Key	Default	Extra
playerid     teamid     start_year     end_year	varchar(30) varchar(30) int int	YES   YES   YES   YES -+	       +	NULL NULL NULL NULL	         +

## ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- MySQL: 8.0.23
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- $\bullet$  The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- mysql-connector-java-xxx.jar has been downloaded and placed in the directory SPARK\_HOME/jars of Spark.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- The Hadoop service has been installed and started.

#### PRECAUTIONS

nebula-exchange\_spark\_2.2 supports only single table queries, not multi-table queries.

#### STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

## Create a graph space. nebula> CREATE SPACE basketballplayer \ (partition\_num = 10, \ replica\_factor = 1, \ vid\_type = FIXED\_STRING(30));

## Use the graph space basketballplayer.
nebula> USE basketballplayer;

## Create the Tag player.
nebula> CREATE TAG player(name string, age int);

## Create the Tag team.
nebula> CREATE TAG team(name string);

## Create the Edge type follow.
nebula> CREATE EDGE follow(degree int);

## Create the Edge type serve. nebula> CREATE EDGE serve(start\_year int, end\_year int);

## For more information, see Quick start workflow.

Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set MySQL data source configuration. In this case, the copied file is called mysql\_application.conf. For details on each configuration item, see Parameters in the configuration file.

{
 # Spark configuration
 spark: {

```
app: {
    name: NebulaGraph Exchange 3.6.1
  driver {
    cores: 1
    maxResultSize: 1G
 cores: {
    max: 16
  }
}
# NebulaGraph configuration
nebula: {
  address:{
    # Specify the IP addresses and ports for Graph and Meta services.
    # If there are multiple addresses, the format is "ip1:port", "ip2:port", "ip3:port".
    # Addresses are separated by commas.
    graph:["127.0.0.1:9669"]
    # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
     meta:["127.0.0.1:9559"]
  # The account entered must have write permission for the NebulaGraph space.
  user: root
  pswd: nebula
  .
# Fill in the name of the graph space you want to write data to in the NebulaGraph.
  space: basketballplayer
  connection: {
    timeout: 3000
    retry: 3
  execution: {
    retry: 3
  error: {
    max: 32
    output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
# Processing vertexes
tags: [
  # Set the information about the Tag player.
  {
    # The Tag name in NebulaGraph.
    name: player
    type: {
      # Specify the data source file format to MySQL.
      source: mysql
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink: client
    }
    host:192.168.*.*
    port:3306
    user:"test"
password:"123456"
    database:"basketball"
    # Scanning a single table to read data.
     # nebula-exchange_spark_2.2 must configure this parameter. Sentence is not supported.
    # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as sentence.
table:"basketball.player"
    # Use query statement to read data.
    # This parameter is not supported by nebula-exchange_spark_2.2.
    # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as table. Multi-table queries are supported.
# sentence: "select * from people, player, team"
    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
# The sequence of fields and nebula.fields must correspond to each other.
     # If multiple column names need to be specified, separate them by commas
    fields: [age,name]
nebula.fields: [age,name]
    # Specify a column of data in the table as the source of VIDs in the NebulaGraph.
    vertex: {
     field:playerid
    # udf:{
                   separator:"_"
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
     # prefix:"tag1"
     # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
```

```
# Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when `writeMode` is `DELETE`.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph. partition: 32
  # Set the information about the Tag Team.
     name: team
    type: {
      source: mysql
      sink: client
    }
    host:192.168.*.*
    port:3306
     .
database:"basketball"
    table:"team"
    user:"test"
    password:"123456"
    sentence: "select teamid, name from team order by teamid;"
    fields: [name]
nebula.fields: [name]
    vertex: {
      field: teamid
    batch: 256
    partition: 32
  }
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
    # The corresponding Edge Type name in NebulaGraph.
    name: follow
    type: {
       # Specify the data source file format to MySQL.
      source: mysql
      # Specify how to import the Edge type data into NebulaGraph.
# Specify how to import the data into NebulaGraph: Client or SST.
      sink: client
    }
    host:192 168 * *
    port:3306
    user:"test'
    password:"123456"
    database:"basketball"
    # Scanning a single table to read data.
    # nebula-exchange_spark_2.2 must configure this parameter. Sentence is not supported.
    # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as sentence.
table:"basketball.follow"
    # Use query statement to read data.
# This parameter is not supported by nebula-exchange_spark_2.2.
    # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as table. Multi-table queries are supported.
# sentence: "select * from follow. serve"
    # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
     # If multiple column names need to be specified, separate them by commas.
    fields: [degree]
nebula.fields: [degree]
    # In source, use a column in the follow table as the source of the edge's source vertex.
# In target, use a column in the follow table as the source of the edge's destination vertex.
     source: {
      field: src plaver
     # udf:{
                   separator:"_"
oldColNames:[field-0,field-1,field-2]
                   newColName:new-field
               }
     # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tag1' will result in 'tag1_12345'. The underscore cannot be modified.
     # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
      field: dst_player
    # udf:{
```

#writeMode: INSERT

```
separator:" '
                    oldColNames:[field-0,field-1,field-2]
                    newColName:new-field
       # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
       # prefix:"tag1"
       # Performs hashing operations on VIDs of type string.
       # policy:hash
      # (Optional) Specify a column as the source of the rank.
      #ranking: rank
       # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
      #writeMode: INSERT
      # The number of data written to NebulaGraph in a single batch.
      batch: 256
      # The number of partitions to be created when the data is written to NebulaGraph.
      partition: 32
    # Set the information about the Edge Type serve.
      name: serve
      type: {
        source: mysql
sink: client
      host:192.168.*.*
      port:3306
database:"basketball"
      table:"serve'
user:"test"
      password:"123456"
       sentence:"select playerid,teamid,start_year,end_year from serve order by playerid;"
      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source: {
        field: playerid
      target: {
        field: teamid
      batch: 256
      partition: 32
}
```

Step 3: Import data into NebulaGraph

Run the following command to import MySQL data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange<3.6.1.jar\_path> -c <mysql\_application.conf\_path>

# Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/nebula-exchange/target/classes/mysql\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

## Import data from Oracle

This topic provides an example of how to use Exchange to export Oracle data and import to NebulaGraph.

DATA SET

This topic takes the basketballplayer dataset as an example.

In this example, the data set has been stored in Oracle. All vertexes and edges are stored in the player, team, follow, and serve tables. The following are some of the data for each table.

oracle> desc	player;	
++	++	+
Column	Null	Type
++	++	+
	-	
NAME	-	VARCHARZ(SU)
AGE	-	NUMBER
++	++	+
oracle> desc	team:	
+	++	+
Column	NuLL I	Type
Cocumit	nucc	туре
+	++	+
TEAMID	-	VARCHAR2(30)
NAME	-	VARCHAR2(30)
++	++	+
oracle> desc	follow	
±		
	1	· · ·
Cotumn	NULL	Туре
+	+	-++
SRC_PLAYER	-	VARCHAR2(30)
DST PLAYER	- 1	VARCHAR2(30)
DEGREE		NUMBER
T	+	
,		-,
oracles desc	corvo.	

oracle> desc :	serve;	
+	+	++
Column	Null	Туре
+	+	++
PLAYERID	-	VARCHAR2(30)
TEAMID	-	VARCHAR2(30)
START_YEAR	-	NUMBER
END_YEAR	-	NUMBER

ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- The Hadoop service has been installed and started.

#### PRECAUTIONS

nebula-exchange spark 2.2 supports only single table queries, not multi-table queries.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \     (partition_num = 10, \     replica_factor = 1, \     vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer. nebula> USE basketballplayer;
## Create the Tag player. nebula> CREATE TAG player(name string, age int);
## Create the Tag team. nebula> CREATE TAG team(name string);
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>

For more information, see Quick start workflow.

Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set Oracle data source configuration. In this case, the copied file is called oracle\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
 # Spark configuration
  spark: {
    app: {
      name: NebulaGraph Exchange 3.6.1
    driver: {
      cores: 1
      maxResultSize: 1G
    cores: {
      max: 16
    }
 }
 # NebulaGraph configuration
  nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and Meta services.
      # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
# Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
      # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
    # The account entered must have write permission for the NebulaGraph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the NebulaGraph.
```

```
space: basketballplayer
  connection: {
    timeout: 3000
   retry: 3
  execution: {
   retry: 3
  error: {
   max: 32
    output: /tmp/errors
 rate: {
    limit: 1024
    timeout: 1000
  }
# Processing vertexes
tags: [
  # Set the information about the Tag player.
  {
    # The Tag name in NebulaGraph.
    name: player
    type: {
      # Specify the data source file format to Oracle.
      source: oracle
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink: client
    }
   url:"jdbc:oracle:thin:@host:1521:basketball"
driver: "oracle.jdbc.driver.OracleDriver"
    user: "root"
    password: "123456"
   # Scanning a single table to read data.
# nebula-exchange_spark_2.2 must configure this parameter. Sentence is not supported.
    # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as sentence.
    table: "basketball.player"
    # Use query statement to read data.
    # Dis gater, Statiant to read odd.
# This parameter is not supported by nebula-exchange_spark_2.2.
# nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as table. Multi-table queries are supported.
    # sentence: "select * from people, player, team"
    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
# If multiple column names need to be specified, separate them by commas.
    fields: [age,name]
    nebula.fields: [age,name]
    # Specify a column of data in the table as the source of VIDs in the NebulaGraph.
    vertex: {
      field:playerid
    # udf:{
                   separator:"_"
    #
                   oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
              }
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
#writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when `writeMode` is `DELETE`.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  # Set the information about the Tag Team.
    name: team
    type: {
      source: oracle
      sink: client
    url:"jdbc:oracle:thin:@host:1521:basketball"
    driver: "oracle.jdbc.driver.OracleDriver
user: "root"
    password: "123456"
    table: "basketball.team"
    sentence: "select teamid, name from team"
    fields: [name]
    nebula.fields: [name]
```

```
vertex: {
      field: teamid
    hatch: 256
   partition: 32
  }
]
# Processing edges
edges: [
  # Set the information about the Edge Type follow.
  {
    # The corresponding Edge Type name in NebulaGraph.
    name: follow
    type: {
      # Specify the data source file format to Oracle.
      source: oracle
      # Specify how to import the Edge type data into NebulaGraph.
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink: client
    }
    url:"jdbc:oracle:thin:@host:1521:basketball"
    driver: "oracle.jdbc.driver.OracleDriver'
    user: "root"
password: "123456"
    # Scanning a single table to read data.
# nebula-exchange_spark_2.2 must configure this parameter. Sentence is not supported.
     # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as sentence.
    table: "basketball.follow'
    # Use query statement to read data.
    # This parameter is not supported by nebula-exchange spark 2.2.
     # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as table. Multi-table queries are supported.
    # sentence: "select * from follow, serve"
     # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
     # If multiple column names need to be specified, separate them by commas.
     fields: [degree]
    nebula.fields: [degree]
    # In source, use a column in the follow table as the source of the edge's source vertex.
# In target, use a column in the follow table as the source of the edge's destination vertex.
    source: {
      field: src_player
     # udf:{
     #
                  separator:" "
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
              3
    # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tagl' will result in 'tagl_12345'. The underscore cannot be modified.
    # Hou the spectrum prime
# prefix:"tag1"
# Performs hashing operations on VIDs of type string.
     # policy:hash
    target: {
      field: dst_player
    # udf:{
                  separator:"_"
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tagl' will result in 'tagl_12345'. The underscore cannot be modified.
     # prefix:"tag1"
     # Performs hashing operations on VIDs of type string.
    # policy:hash
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # The number of data written to NebulaGraph in a single batch.
    hatch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  }
  # Set the information about the Edge Type serve.
  {
    name: serve
    type: {
      source: oracle
      sink: client
```

	url:"jdbc:oracle:thin:@host:1521:basketball" driver: "oracle.jdbc.driver.OracleDriver" user: "root" password: "123456" table: "basketball.serve" sentence: "select playerid, teamid, start_year, end_year from serve"
	fields: [start year.end year]
	nebula.fields: [start year,end year]
	source: {
	field: playerid
	}
	target: {
	field: teamid
	}
	batch: 256
	partition: 32
	}
]	

Step 3: Import data into NebulaGraph

Run the following command to import Oracle data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <oracle\_application.conf\_path>

#### Q Note

}

JAR packages are available in two ways: compiled them yourself, or download the compiled ...jar file directly.

For example:

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-
exchange/nebula-exchange/target/classes/oracle_application.conf
```

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

## Import data from ClickHouse

This topic provides an example of how to use Exchange to import data stored on ClickHouse into NebulaGraph.

DATA SET

This topic takes the basketballplayer dataset as an example.

### ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- ClickHouse: docker deployment yandex/clickhouse-server tag: latest(2021.07.01)
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>		
## Use the graph space basketballplayer. nebula> USE basketballplayer;		
## Create the Tag player. nebula> CREATE TAG player(name string, age int);		
## Create the Tag team. nebula> CREATE TAG team(name string);		
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>		
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>		

For more information, see Quick start workflow.

## Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set ClickHouse data source configuration. In this example, the copied file is called clickhouse\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
    app: {
    name: NebulaGraph Exchange 3.6.1
    driver: {
      cores: 1
      maxResultSize: 1G
    cores: {
      max: 16
    }
  }
# NebulaGraph configuration
  nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and Meta services.
      # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
# Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
      # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
    # The account entered must have write permission for the NebulaGraph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
```

```
}
  execution: {
    retry: 3
  3
  error: {
    max: 32
   output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# Processing vertexes
tags: [
    # Set the information about the Tag player.
  {
    name: player
    type: {
      # Specify the data source file format to ClickHouse.
      source: clickhouse
      # Specify how to import the data of vertexes into NebulaGraph: Client or SST.
     sink: client
    }
    # JDBC URL of ClickHouse
    url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
    user:"user"
    password:"123456"
    # The number of ClickHouse partitions
    numPartition:"5"
    sentence:"select * from player"
    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other
    # If multiple column names need to be specified, separate them by commas.
    fields: [name,age]
    nebula.fields: [name,age]
    # Specify a column of data in the table as the source of vertex VID in the NebulaGraph.
    vertex: {
     field:playerid
    # udf:{
                 separator:"_"
oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
    #
             1
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  }
  # Set the information about the Tag Team.
    name: team
    type: {
      source: clickhouse
     sink: client
    url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
    user:"user"
    password:"123456"
    .
numPartition:"5"
    sentence:"select * from team"
    fields: [name]
nebula.fields: [name]
    vertex: {
     field:teamid
    batch: 256
    partition: 32
  }
# Processing edges
edges: [
```

```
# The corresponding Edge Type name in NebulaGraph.
  name: follow
  type: {
    # Specify the data source file format to ClickHouse.
   source: clickhouse
   \ensuremath{\texttt{\#}} Specify how to import the data into NebulaGraph: Client or SST.
   sink: client
  }
  # JDBC URL of ClickHouse
  url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
  user:"user"
  password:"123456"
  # The number of ClickHouse partitions.
  numPartition:"5"
  sentence:"select * from follow"
  # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the NebulaGraph.
  # The sequence of fields and nebula.fields must correspond to each other.
# If multiple column names need to be specified, separate them by commas.
  fields: [degree]
  nebula.fields: [degree]
  # In source, use a column in the follow table as the source of the edge's source vertexes.
  source: {
   field:src_player
  # udf:{
               separator:"_"
               oldColNames:[field-0,field-1,field-2]
               newColName:new-field
  #
           }
  # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tag1' will result in 'tag1_12345'. The underscore cannot be modified.
  # prefix:"tagl"
# Performs hashing operations on VIDs of type string.
  # policy:hash
  # In target, use a column in the follow table as the source of the edge's destination vertexes.
  target: {
   field:dst_player
  # udf:{
               separator:"_"
  #
               oldColNames:[field-0,field-1,field-2]
  #
  #
               newColName:new-field
  # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
  # Performs hashing operations on VIDs of type string.
  # policy:hash
  # (Optional) Specify a column as the source of the rank.
 #ranking: rank
 # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
 #writeMode: INSERT
  # The number of data written to NebulaGraph in a single batch.
  batch: 256
  # The number of partitions to be created when the data is written to NebulaGraph.
 partition: 32
}
# Set the information about the Edge Type serve.
  name: serve
  type: {
   source: clickhouse
   sink: client
  user:"user"
password:"123456"
  numPartition:"5"
  sentence:"select * from serve"
  fields: [start_year,end_year]
  nebula.fields: [start_year,end_year]
  source: {
   field:playerid
  ι
  target: {
   field:teamid
  1
  # (Optional) Specify a column as the source of the rank.
```

#ranking: rank

		batch: 256	
		partition: 32	
	}		
]			
}			

Step 3: Import data into NebulaGraph

Run the following command to import ClickHouse data into NebulaGraph. For descriptions of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula.exchange-3.6.1.jar\_path> -c <clickhouse\_application.conf\_path>

#### O Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

#### For example:

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/nebula-exchange/target/classes/clickhouse_application.conf
```

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

## Import data from Neo4j

This topic provides an example of how to use Exchange to import NebulaGraph data stored in Neo4j.

#### IMPLEMENTATION METHOD

Exchange uses **Neo4j Driver 4.0.1** to read Neo4j data. Before batch export, you need to write Cypher statements that are automatically executed based on labels and relationship types and the number of Spark partitions in the configuration file to improve data export performance.

When Exchange reads Neo4j data, it needs to do the following:

- 1. The Reader in Exchange replaces the statement following the Cypher RETURN statement in the exec part of the configuration file with COUNT(\*), and executes this statement to get the total amount of data, then calculates the starting offset and size of each partition based on the number of Spark partitions.
- 2. (Optional) If the user has configured the check\_point\_path directory, Reader reads the files in the directory. In the transferring state, Reader calculates the offset and size that each Spark partition should have.
- 3. In each Spark partition, the Reader in Exchange adds different SKIP and LIMIT statements to the Cypher statement and calls the Neo4j Driver for parallel execution to distribute data to different Spark partitions.
- 4. The Reader finally processes the returned data into a DataFrame.

At this point, Exchange has finished exporting the Neo4j data. The data is then written in parallel to the NebulaGraph database.

The whole process is illustrated below.



DATA SET

This topic takes the basketballplayer dataset as an example.

### ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz
- CPU cores: 14
- Memory: 251 GB
- Spark: Stand-alone, 2.4.6 pre-build for Hadoop 2.7
- Neo4j: 3.5.20 Community Edition
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with NebulaGraph write permission.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.

• Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.

#### STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

## Create a graph space nebula> CREATE SPACE basketballplayer \ (partition\_num = 10, \ replica\_factor = 1, \ vid\_type = FIXED\_STRING(30)); ## Use the graph space basketballplayer nebula> USE basketballplayer; ## Create the Tag player nebula> CREATE TAG player(name string, age int); ## Create the Tag team nebula> CREATE TAG team(name string); ## Create the Edge type follow nebula> CREATE EDGE follow(degree int);

## Create the Edge type serve
nebula> CREATE EDGE serve(start\_year int, end\_year int);

For more information, see Quick start workflow.

Step 2: Configuring source data

To speed up the export of Neo4j data, create indexes for the corresponding properties in the Neo4j database. For more information, refer to the Neo4j manual.

Step 3: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set Neo4j data source configuration. In this example, the copied file is called neo4j\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
   app: {
     name: NebulaGraph Exchange 3.6.1
   }
   driver: {
     cores: 1
     maxResultSize: 1G
    }
   executor: {
       memory:1G
   }
   cores: {
      max: 16
   }
 }
  # NebulaGraph configuration
 nebula: {
address:{
     graph:["127.0.0.1:9669"]
     # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
     meta:["127.0.0.1:9559"]
    }
   user: root
   pswd: nebula
   space: basketballplayer
   connection: {
     timeout: 3000
      retry: 3
    }
    execution: {
     retry: 3
   }
   error: {
     max: 32
     output: /tmp/errors
    }
   rate: {
      limit: 1024
      timeout: 1000
   }
  }
  # Processing vertexes
 tags: [
    # Set the information about the Tag player
    {
      name: player
      type: {
       source: neo4j
       sink: client
      }
      server: "bolt://192.168.*.*:7687"
     user: neo4j
      password:neo4j
      .
# bolt 3 does not support multiple databases, do not configure database names. 4 and above can configure database names.
      # database:neo4j
      exec: "match (n:player) return n.id as id, n.age as age, n.name as name"
      fields: [age,name]
      nebula.fields: [age,name]
      vertex: {
       field:id
      # udf:{
                   separator:"_"
```

```
oldColNames:[field-0,field-1,field-2]
    #
                 newColName:new-field
             }
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when `writeMode` is `DELETE`.
    #deleteEdge: false
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
# Set the information about the Tag Team
{
    name: team
    type: {
     source: neo4j
     sink: client
    }
    server: "bolt://192.168.*.*:7687"
    user: neo4j
    password:neo4j
    database:neo4j
    exec: "match (n:team) return n.id as id,n.name as name"
    fields: [name]
    nebula.fields: [name]
    vertex: {
     field:id
    3
    partition: 10
    batch: 1000
    check_point_path: /tmp/test
]
# Processing edges
edges:
  # Set the information about the Edge Type follow
  {
    name: follow
    type: {
     source: neo4j
     sink: client
    }
    server: "bolt://192.168.*.*:7687"
    user: neo4j
    password:neo4j
    # bolt 3 does not support multiple databases, do not configure database names. 4 and above can configure database names.
    # database:neo4j
    exec: "match (a:player)-[r:follow]->(b:player) return a.id as src, b.id as dst, r.degree as degree order by id(r)"
    fields: [degree]
nebula.fields: [degree]
    source: {
  field: src
    # udf:{
                 separator:"_"
oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
             }
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
     field: dst
    # udf:{
                 separator:"_"
                 oldColNames:[field-0,field-1,field-2]
                 newColName:new-field
             }
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    partition: 10
    batch: 1000
    check point path: /tmp/test
 # Set the information about the Edge Type serve
```

```
name: serve
type: {
  source: neo4j
  sink: client
server: "bolt://192.168.*.*:7687"
user: neo4j
password:neo4j
database:neo4j
exec: "match (a:plaver)-[r:serve]->(b:team) return a.id as src. b.id as dst. r.start vear as start vear. r.end vear as end vear order by id(r)"
fields: [start_year,end_year]
nebula.fields: [start_year,end_year]
source: {
  field: src
target: {
  field: dst
#ranking: rank
partition: 10
batch: 1000
check_point_path: /tmp/test
```

Exec configuration

When configuring either the tags.exec or edges.exec parameters, you need to fill in the Cypher query. To prevent loss of data during import, it is strongly recommended to include ORDER BY clause in Cypher queries. Meanwhile, in order to improve data import efficiency, it is better to select indexed properties for ordering. If there is no index, users can also observe the default order and select the appropriate properties for ordering to improve efficiency. If the pattern of the default order cannot be found, users can order them by the ID of the vertex or relationship and set the partition to a small value to reduce the ordering pressure of Neo4j.

#### Q Note

Using the ORDER BY clause lengthens the data import time.

Exchange needs to execute different SKIP and LIMIT Cypher statements on different Spark partitions, so SKIP and LIMIT clauses cannot be included in the Cypher statements corresponding to tags.exec and edges.exec.

tags.vertex or edges.vertex configuration

NebulaGraph uses ID as the unique primary key when creating vertexes and edges, overwriting the data in that primary key if it already exists. So, if a Neo4j property value is given as the NebulaGraph'S ID and the value is duplicated in Neo4j, duplicate IDs will be generated. One and only one of their corresponding data will be stored in the NebulaGraph, and the others will be overwritten. Because the data import process is concurrently writing data to NebulaGraph, the final saved data is not guaranteed to be the latest data in Neo4j.

check\_point\_path configuration

If breakpoint transfers are enabled, to avoid data loss, the state of the database should not change between the breakpoint and the transfer. For example, data cannot be added or deleted, and the partition quantity configuration should not be changed.

## Step 4: Import data into NebulaGraph

Run the following command to import Neo4j data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange <nebula-exchange-3.6.1.jar\_path> -c <neo4j\_application.conf\_path>

# Note

JAR packages are available in two ways: compiled them yourself, or download the compiled ...jar file directly.

## For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebulaexchange/nebula-exchange/target/classes/neo4j\_application.conf You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 5: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 6: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

Last update: October 25, 2023

## Import data from Hive

This topic provides an example of how to use Exchange to import NebulaGraph data stored in Hive.

DATA SET

This topic takes the basketballplayer dataset as an example.

In this example, the data set has been stored in Hive. All vertexes and edges are stored in the player, team, follow, and serve tables. The following are some of the data for each table.



# Note

The Hive data type  $\ensuremath{\mathsf{bigint}}$  corresponds to the NebulaGraph  $\ensuremath{\mathsf{int}}$  .

ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- Hadoop: 2.9.2, pseudo-distributed deployment
- Hive: 2.3.7, Hive Metastore database is MySQL 8.0.22
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- The Hive Metastore database (MySQL in this example) has been started.

#### STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

For more information, see Quick start workflow.

Step 2: Use Spark SQL to confirm Hive SQL statements

After the Spark-shell environment is started, run the following statements to ensure that Spark can read data in Hive.

```
scala> sql("select playerid, age, name from basketball.player").show
scala> sql("select teamid, name from basketball.team").show
scala> sql("select src_player, dst_player, degree from basketball.follow").show
scala> sql("select playerid, teamid, start_year, end_year from basketball.serve").show
```

The following is the result read from the table basketball.player.

++-	+-	+
playerid	age	name
++-	+-	+
player100	42	Tim Duncan

player101	36	Tony Parker
player102	33 L	.aMarcus Aldridge
player103	32	Rudy Gay
player104	32	Marco Belinelli
++-	+-	+

Step 3: Modify configuration file

After Exchange is compiled, copy the conf file target/classes/application.conf to set Hive data source configuration. In this example, the copied file is called hive\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
    app: {
      name: NebulaGraph Exchange 3.6.1
    driver: {
      cores: 1
      maxResultSize: 1G
    cores: {
      max: 16
    }
  }
  # If Spark and Hive are deployed in different clusters, you need to configure the parameters for connecting to Hive. Otherwise, skip these configurations.
  #hive:
 #UNC. \
# waredir: "hdfs://NAMENODE_IP:9000/apps/svr/hive-xxx/warehouse/"
# connectionURL: "jdbc:mysql://your_ip:3306/hive_spark?characterEncoding=UTF-8"
# connectionDriverName: "com.mysql.jdbc.Driver"
# connectionUserName: "user"
  # connectionPassword: "password"
  #}
  # NebulaGraph configuration
  nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and all Meta services.
      # If there are multiple addresses, the format is "ip1:port", "ip2:port", "ip3:port".
      # Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
      # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
     # The account entered must have write permission for the NebulaGraph space.
    user: root
    pswd: nebula
     # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    execution: {
      retry: 3
    error: {
      max: 32
      output: /tmp/errors
    rate: {
      limit: 1024
      timeout: 1000
    }
  # Processing vertexes
  tags: [
    # Set the information about the Tag player.
    {
      # The Tag name in NebulaGraph.
      name: player
type: {
         # Specify the data source file format to Hive.
        source: hive
         # Specify how to import the data into NebulaGraph: Client or SST.
        sink: client
      }
      # Set the SQL statement to read the data of player table in basketball database.
      exec: "select playerid, age, name from basketball.player'
      # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
       # The sequence of fields and nebula.fields must correspond to each other.
      # If multiple column names need to be specified, separate them by commas.
      fields: [age,name]
      nebula.fields: [age,name]
       # Specify a column of data in the table as the source of vertex VID in the NebulaGraph
```
vertex:{ field:playerid # udf:{ separator." " # oldColNames:[field-0,field-1,field-2] newColName:new-field } # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1\_12345`. The underscore cannot be modified. # prefix:"tag1" # Performs hashing operations on VIDs of type string. # policy:hash # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT. #writeMode: INSERT # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'. #deleteEdge: false # The number of data written to NebulaGraph in a single batch. batch: 256 # The number of partitions to be created when the data is written to NebulaGraph. partition: 32 # Set the information about the Tag Team. name: team type: { source: hive sink: client } exec: "select teamid, name from basketball.team" fields: [name] nebula.fields: [name] vertex: { field: teamid batch: 256 partition: 32 } ] # Processing edges edges: [ # Set the information about the Edge Type follow. { # The corresponding Edge Type name in NebulaGraph. name: follow type: { # Specify the data source file format to Hive. source: hive # Specify how to import the Edge type data into NebulaGraph. # Specify how to import the data into NebulaGraph: Client or SST. sink: client } # Set the SOL statement to read the data of follow table in the basketball database. exec: "select src\_player, dst\_player, degree from basketball.follow" # Specify the column names in the follow table in Fields, and their corresponding values are specified as properties in the NebulaGraph. # The sequence of fields and nebula.fields must correspond to each other. # If multiple column names need to be specified, separate them by commas fields: [degree] nebula.fields: [degree] # In source, use a column in the follow table as the source of the edge's starting vertex. # In target, use a column in the follow table as the source of the edge's destination vertex. source: { field: src\_player # udf:{ separator:"\_" # oldColNames:[field-0,field-1,field-2] newColName:new-field } # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1\_12345`. The underscore cannot be modified. # prefix:"tag1" # Performs hashing operations on VIDs of type string. # policy:hash target: { field: dst player # udf:{ separator:" " # oldColNames:[field-0,field-1,field-2] newColName:new-field } # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tag1' will result in 'tag1\_12345'. The underscore cannot be modified. # prefix:"tag1" # Performs hashing operations on VIDs of type string.

```
# policy:hash
  # (Optional) Specify a column as the source of the rank.
  #ranking: rank
  # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
  #writeMode: INSERT
  # The number of data written to NebulaGraph in a single batch.
  batch: 256
  # The number of partitions to be created when the data is written to NebulaGraph.
  partition: 32
}
# Set the information about the Edge Type serve.
  name: serve
  type: {
    source: hive
    sink: client
  exec: "select playerid, teamid, start_year, end_year from basketball.serve"
  fields: [start_year,end_year]
  nebula.fields: [start_year,end_year]
  source: {
    field: playerid
  target: {
   field: teamid
  }
  # (Optional) Specify a column as the source of the rank.
  #ranking: rank
  batch: 256
  partition: 32
}
```

Step 4: Import data into NebulaGraph

}

Run the following command to import Hive data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <hive\_application.conf\_path> -h

# Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

## For example:

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-
exchange/nebula-exchange/target/classes/hive_application.conf -h
```

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 5: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 6: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

```
Last update: November 7, 2023
```

## Import data from MaxCompute

This topic provides an example of how to use Exchange to import NebulaGraph data stored in MaxCompute.

DATA SET

This topic takes the basketballplayer dataset as an example.

## ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- MaxCompute: Alibaba Cloud official version
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

## PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer. nebula> USE basketballplayer;
## Create the Tag player. nebula> CREATE TAG player(name string, age int);
## Create the Tag team. nebula> CREATE TAG team(name string);
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>

For more information, see Quick start workflow.

## Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set MaxCompute data source configuration. In this example, the copied file is called maxcompute\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
    app: {
    name: NebulaGraph Exchange 3.6.1
    driver: {
      cores: 1
      maxResultSize: 1G
    cores: {
      max: 16
    }
 }
 # NebulaGraph configuration
  nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and Meta services.
      # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
# Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
      # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
    # The account entered must have write permission for the NebulaGraph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
```

```
execution: {
    retry: 3
 error: {
    max: 32
   output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
 }
# Processing vertexes
tags: [
  # Set the information about the Tag player.
  {
    name: player
    type: {
      # Specify the data source file format to MaxCompute.
      source: maxcompute
      # Specify how to import the data into NebulaGraph: Client or SST.
     sink: client
    }
    # Table name of MaxCompute.
    table:player
    # Project name of MaxCompute.
    project:project
    # OdpsUrl and tunnelUrl for the MaxCompute service.
    # The address is https://help.aliyun.com/document_detail/34951.html.
    odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api
    tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"
    # AccessKeyId and accessKeySecret of the MaxCompute service.
    accessKeyId:xxx
    accessKeySecret:xxx
    \# Partition description of the MaxCompute table. This configuration is optional. partitionSpec:"dt='partition1'"
    # Ensure that the table name in the SQL statement is the same as the value of the table above. This configuration is optional.
    sentence:"select id, name, age, playerid from player where id < 10"
    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
# The sequence of fields and nebula.fields must correspond to each other.
    # If multiple column names need to be specified, separate them by commas.
    fields:[name, age]
nebula.fields:[name, age]
    # Specify a column of data in the table as the source of vertex VID in the NebulaGraph.
    vertex:{
     field: playerid
    # udf:{
                  separator:"_"
                 separator:"_"
oldColNames:[field-0,field-1,field-2]
newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when `writeMode` is `DELETE`.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  }
  # Set the information about the Tag Team.
    name: team
    type: {
      source: maxcompute
     sink: client
    table:team
    project:project
    odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
    tunnelUrl: "http://dt.cn-hangzhou.maxcompute.aliyun.com"
    accessKeyId:xxx
    accessKeySecret:xxx
    partitionSpec:"dt='partition1'"
sentence:"select id, name, teamid from team where id < 10"</pre>
```

```
fields:[name]
nebula.fields:[name]
    vertex:{
      field teamid
    batch: 256
   partition: 32
 }
# Processing edges
edges: [
    # Set the information about the Edge Type follow.
  {
    # The corresponding Edge Type name in NebulaGraph.
    name: follow
    type:{
      # Specify the data source file format to MaxCompute.
      source:maxcompute
      # Specify how to import the Edge type data into NebulaGraph.
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink:client
    }
    # Table name of MaxCompute.
    table:follow
    # Project name of MaxCompute.
    project:project
    # OdpsUrl and tunnelUrl for MaxCompute service.
    # The address is https://help.aliyun.com/document_detail/34951.html.
odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
    tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"
    # AccessKeyId and accessKeySecret of the MaxCompute service.
    accessKeyId:xxx
    accessKeySecret:xxx
    # Partition description of the MaxCompute table. This configuration is optional.
    partitionSpec:"dt='partition1''
    # Ensure that the table name in the SQL statement is the same as the value of the table above. This configuration is optional. sentence:"select * from follow"
    # Specify the column names in the follow table in Fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
    # If multiple column names need to be specified, separate them by commas.
    fields:[degree]
    nebula.fields:[degree]
    # In source, use a column in the follow table as the source of the edge's source vertex.
    source:{
     field: src plaver
    # udf:{
                  separator:" "
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
             3
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # In target, use a column in the follow table as the source of the edge's destination vertex.
    target:{
      field: dst_player
    # udf:{
                  separator:" "
                  oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
    #
             }
    #
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition:10
    # The number of data written to NebulaGraph in a single batch.
    batch:10
  }
```

1

```
# Set the information about the Edge Type serve.
       name: serve
      type:{
         source:maxcompute
        sink:client
      table:serve
      project:project
odpsUrl:"http://service.cn-hangzhou.maxcompute.aliyun.com/api"
       tunnelUrl:"http://dt.cn-hangzhou.maxcompute.aliyun.com"
       accessKeyId:xxx
      accessKeySecret:xxx
      partitionSpec:"dt='partition1'"
sentence:"select * from serve"
       fields:[start_year,end_year]
      nebula.fields:[start_year,end_year]
      source:{
         field: playerid
      target:{
        field: teamid
      }
      # (Optional) Specify a column as the source of the rank.
      #ranking: rank
       partition:10
      batch:10
}
```

Step 3: Import data into NebulaGraph

Run the following command to import MaxCompute data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula.exchange-3.6.1.jar\_path> -c <maxcompute\_application.conf\_path>

Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/nebula-exchange/target/classes/maxcompute\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

Last update: November 7, 2023

## Import data from Pulsar

This topic provides an example of how to use Exchange to import NebulaGraph data stored in Pulsar.

## ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

## PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- The Pulsar service has been installed and started.

PRECAUTIONS

- Only client mode is supported when importing Pulsar data, i.e. the value of parameters tags.type.sink and edges.type.sink is client.
- When importing Pulsar data, do not use Exchange version 3.4.0, which adds caching of imported data and does not support streaming data import. Use Exchange versions 3.0.0, 3.3.0, or 3.5.0.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	start_year int, end_year int

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space nebula&gt; CREATE SPACE basketballplayer \     (partition_num = 10, \     replica_factor = 1, \     vid_type = FIXED_STRING(30));</pre>
<pre>## Use the graph space basketballplayer nebula&gt; USE basketballplayer;</pre>
## Create the Tag player nebula> CREATE TAG player(name string, age int);
## Create the Tag team nebula> CREATE TAG team(name string);
<pre>## Create the Edge type follow nebula&gt; CREATE EDGE follow(degree int);</pre>
## Create the Edge type serve nebula> CREATE EDGE serve(start year int, end year int

For more information, see Quick start workflow.

Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set Pulsar data source configuration. In this example, the copied file is called pulsar\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
 spark: {
   app: {
      name: NebulaGraph Exchange 3.6.1
   driver: {
     cores: 1
      maxResultSize: 1G
   cores: {
     max: 16
   }
 }
 # NebulaGraph configuration
  nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and all Meta services.
      # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
      # Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
      # the address of any of the meta services.
# if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
    }
    # The account entered must have write permission for the NebulaGraph space.
   user: root
pswd: nebula
    # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
```

```
timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  rate: {
    limit: 1024
    timeout: 1000
  }
# Processing vertices
tags: [
  # Set the information about the Tag player.
  {
    # The corresponding Tag name in NebulaGraph.
    name: player
    type: {
      # Specify the data source file format to Pulsar.
      source: pulsar
      # Specify how to import the data into NebulaGraph. Only client is supported.
      sink: client
    # The address of the Pulsar server.
service: "pulsar://127.0.0.1:6650"
    # admin.url of pulsar.
    admin: "http://127.0.0.1:8081"
     # The Pulsar option can be configured from topic, topics or topicsPattern.
    options: {
topics: "topic1,topic2"
    }
    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other
    # If multiple column names need to be specified, separate them by commas.
    fields: [age,name]
    nebula.fields: [age,name]
    # Specify a column of data in the table as the source of VIDs in the NebulaGraph.
    vertex:{
        field:playerid
     # udf:{
                 separator:"_"
oldColNames:[field-0,field-1,field-2]
                  newColName:new-field
    #
             3
     # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
     # prefix:"tag1"
     # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
     # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 10
    \ensuremath{^\#} The number of partitions to be created when the data is written to NebulaGraph. partition: 10
     # The interval for message reading. Unit: second.
    interval.seconds: 10
   # Set the information about the Tag Team.
    name: team
    type: {
      source: pulsar
      sink: client
    service: "pulsar://127.0.0.1:6650"
     admin: "http://127.0.0.1:8081"
    options: {
      topics: "topic1,topic2"
    fields: [name]
    nebula.fields: [name]
    vertex:{
        field:teamid
    batch: 10
    partition: 10
    interval.seconds: 10
  }
]
```

# Processing edges edges: [ # Set the information about Edge Type follow { # The corresponding Edge Type name in NebulaGraph. name: follow type: { # Specify the data source file format to Pulsar. source: pulsar # Specify how to import the Edge type data into NebulaGraph. # Specify how to import the data into NebulaGraph. Only client is supported. sink: client } # The address of the Pulsar server. service: "pulsar://127.0.0.1:6650" # admin.url of pulsar admin: "http://127.0.0.1:8081" # The Pulsar option can be configured from topic, topics or topicsPattern. options: { topics: "topic1,topic2" } # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the NebulaGraph. # The sequence of fields and nebula.fields must correspond to each other. # If multiple column names need to be specified, separate them by commas fields: [degree] nebula.fields: [degree] # In source, use a column in the follow table as the source of the edge's source vertex. # In target, use a column in the follow table as the source of the edge's destination vertex. source:{ field:src\_player # udf:{ separator:"\_" oldColNames:[field-0,field-1,field-2] newColName:new-field } # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1\_12345`. The underscore cannot be modified. # prefix:"tag1" # Performs hashing operations on VIDs of type string. # policy:hash target:{ field:dst\_player # udf:{ separator." " oldColNames:[field-0,field-1,field-2] newColName:new-field } # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1\_12345`. The underscore cannot be modified. # prefix:"tag1" # Performs hashing operations on VIDs of type string. # policy:hash # (Optional) Specify a column as the source of the rank. #ranking: rank # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT. #writeMode: INSERT # The number of data written to NebulaGraph in a single batch. batch: 10 # The number of partitions to be created when the data is written to NebulaGraph. partition: 10 # The interval for message reading. Unit: second. interval.seconds: 10 1 # Set the information about the Edge Type serve name: serve type: { source: Pulsar sink: client service: "pulsar://127.0.0.1:6650" admin: "http://127.0.0.1:8081" options: {
 topics: "topic1,topic2" 1 fields: [start\_year,end\_year] nebula.fields: [start\_year,end\_year] source:{ field:playerid 1

```
target:{
    field:teamid
}
# (Optional) Specify a column as the source of the rank.
#ranking: rank
batch: 10
partition: 10
interval.seconds: 10
}
]
```

Step 3: Import data into NebulaGraph

Run the following command to import Pulsar data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula.exchange-3.6.1.jar\_path> -c <pulsar\_application.conf\_path>

Q Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

## For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/nebula-exchange/target/classes/pulsar\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

Last update: November 7, 2023

## Import data from Kafka

This topic provides a simple guide to importing Data stored on Kafka into NebulaGraph using Exchange.

# Empatibility

Please use Exchange 3.5.0/3.3.0/3.0.0 when importing Kafka data. In version 3.4.0, caching of imported data was added, and streaming data import is not supported.

ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

## PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- The following JAR files have been downloaded and placed in the directory SPARK\_HOME/jars of Spark:
- spark-streaming-kafka\_xxx.jar
- spark-sql-kafka-0-10\_xxx.jar
- kafka-clients-xxx.jar
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- The Kafka service has been installed and started.

## PRECAUTIONS

- Only client mode is supported when importing Kafka data, i.e. the value of parameters tags.type.sink and edges.type.sink is client.
- When importing Kafka data, do not use Exchange version 3.4.0, which adds caching of imported data and does not support streaming data import. Use Exchange versions 3.0.0, 3.3.0, or 3.5.0.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	start_year int, end_year int

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space. nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer. nebula> USE basketballplayer;
## Create the Tag player. nebula> CREATE TAG player(name string, age int);
## Create the Tag team. nebula> CREATE TAG team(name string);
<pre>## Create the Edge type follow. nebula&gt; CREATE EDGE follow(degree int);</pre>
<pre>## Create the Edge type serve. nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>

For more information, see Quick start workflow.

Step 2: Modify configuration files

# Note

If some data is stored in Kafka's value field, you need to modify the source code, get the value from Kafka, parse the value through the from\_JSON function, and return it as a Dataframe.

After Exchange is compiled, copy the conf file target/classes/application.conf to set Kafka data source configuration. In this example, the copied file is called kafka\_application.conf. For details on each configuration item, see Parameters in the configuration file.

#### Q Note

When importing Kafka data, a configuration file can only handle one tag or edge type. If there are multiple tag or edge types, you need to create multiple configuration files.

```
# Spark configuration
spark: {
    app: {
        name: NebulaGraph Exchange 3.6.1
    }
    driver: {
        cores: 1
        maxResultSize: 16
    }
    cores: {
        max: 16
    }
}
```

```
# NebulaGraph configuration
nebula: {
  address:{
    # Specify the IP addresses and ports for Graph and all Meta services.
    # If there are multiple addresses, the format is "ip1:port","ip2:port","ip3:port".
    # Addresses are separated by commas.
    graph:["127.0.0.1:9669"]
# the address of any of the meta services.
    # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
    meta:["127.0.0.1:9559"]
  # The account entered must have write permission for the NebulaGraph space.
  user: root
pswd: nebula
  .
# Fill in the name of the graph space you want to write data to in the NebulaGraph.
  space: basketballplaver
  connection: {
    timeout: 3000
    retry: 3
  execution: {
    retry: 3
  error: {
    max: 32
    output: /tmp/errors
  rate: {
    limit · 1024
    timeout: 1000
  }
# Processing vertexes
tags: [
  # Set the information about the Tag player.
  {
    # The corresponding Tag name in NebulaGraph.
    name: player
    type: {
      # Specify the data source file format to Kafka.
      source: kafka
      # Specify how to import the data into NebulaGraph. Only client is supported.
      sink: client
    # Kafka server address.
    service: "127.0.0.1:9092"
    # Message category.
topic: "topic_name1"
    # Kafka data has a fixed domain name: key, value, topic, partition, offset, timestamp, timestampType.
# If multiple fields need to be specified after Spark reads as DataFrame, separate them with commas.
# Specify the field name in fields. For example, use key for name in NebulaGraph and value for age in Nebula, as shown in the following.
fields: [key,value]
    nebula.fields: [name,age]
     # Specify a column of data in the table as the source of vertex VID in the NebulaGraph.
     # The key is the same as the value above, indicating that key is used as both VID and property name.
    vertex:{
        field:key
    # udf:{
                   separator:"_"
     #
                   oldColNames:[field-0,field-1,field-2]
                   newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'.
    #deleteEdge: false
    # The number of data written to NebulaGraph in a single batch.
    batch: 10
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 10
    # The interval for message reading. Unit: second.
    interval.seconds: 10
    # The consumer offsets. The default value is latest. Optional value are latest and earliest.
    startingOffsets: latest
    # Flow control, with a rate limit on the maximum offset processed per trigger interval, may not be configured.
    # maxOffsetsPerTrigger:10000
  }
```

```
# Processing edges
```

#edges: [ # Set the information about the Edge Type follow # # { # The corresponding Edge Type name in NebulaGraph. # name: follow # # type: { # Specify the data source file format to Kafka. source: kafka # # Specify how to import the Edge type data into NebulaGraph. # # # Specify how to import the data into NebulaGraph. Only client is supported. # sink: client # } # # Kafka server address # service: "127.0.0.1:9092" # Message category.
topic: "topic\_name3" # # Kafka data has a fixed domain name: key, value, topic, partition, offset, timestamp, timestampType # If multiple fields need to be specified after Spark reads as DataFrame, separate them with co # Specify the field name in fields. For example, use key for degree in Nebula, as shown in the following. # fields: [key] nebula.fields: [degree] # # # In source, use a column in the topic as the source of the edge's source vertex.  $\ensuremath{\texttt{\#}}$  In target, use a column in the topic as the source of the edge's destination vertex. # source:{ field:timestamp # udf:{ # separator:"\_" oldColNames:[field-0,field-1,field-2] newColName:new-field # # # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1\_12345`. The underscore cannot be modified. # prefix:"tag1" # # # Performs hashing operations on VIDs of type string. # # policy:hash # } # target:{ field:offset # udf:{ # # separator:"\_" oldColNames:[field-0,field-1,field-2] newColName:new-field # # # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tagl' will result in 'tag1\_12345'. The underscore cannot be modified. # # prefix:"tag1" # Performs hashing operations on VIDs of type string. # # policy:hash # # # (Optional) Specify a column as the source of the rank. # #ranking: rank # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT. #writeMode: INSERT # # # The number of data written to NebulaGraph in a single batch. # batch: 10 # The number of partitions to be created when the data is written to NebulaGraph. # # partition: 10 # # The interval for message reading. Unit: second. # interval.seconds: 10 # # The consumer offsets. The default value is latest. Optional value are latest and earliest. # startingOffsets: latest # Flow control, with a rate limit on the maximum offset processed per trigger interval, may not be configured. # # maxOffsetsPerTrigger:10000 # } #1 }

Step 3: Import data into NebulaGraph

## Run the following command to import Kafka data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange <nebula-exchange <nebula-exchange-3.6.1.jar\_path> -c <kafka\_application.conf\_path>

# Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebulaexchange/nebula-exchange/target/classes/kafka\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

Last update: November 7, 2023

## Import data from general JDBC

JDBC data refers to the data of various databases accessed through the JDBC interface. This topic provides an example of how to use Exchange to export MySQL data and import to NebulaGraph.

## DATA SET

This topic takes the basketballplayer dataset as an example.

In this example, the data set has been stored in MySQL. All vertexes and edges are stored in the player, team, follow, and serve tables. The following are some of the data for each table.

mysql> desc p	layer;	-+	++	+		+	+
Field	Гуре	N	ull	Key	Default	Exti	ra
playerid     age     name   ++-	int int varchar( <mark>30</mark> )	Y    Y    Y	ES   ES   ES	      +	NULL NULL NULL	     +	     +
mysql> desc t	eam;		+	+	+		.+
Field   Ty	pe	Nul	l   Ke	ey   D	efault	Extra	-+
teamid   in   name   va ++	t rchar(30)	YES YES	   +	N   N	ULL   ULL		   +
mysql> desc f	ollow;	+		+	_+	+	
Field	Туре	 	Null	Key	Defau	lt   Ex	ktra
src_player   dst_player   degree +	int   int   int +	     	YES YES YES	     +	NULL   NULL   NULL   NULL	     	
mysql> desc s	erve;						
+ Field			Null	+   Key	Defau	lt   Ex	ktra
playerid   teamid   start_year   end_year +	int   int   int   int	       	YES YES YES YES	       +	NULL   NULL   NULL   NULL -+	     	

## ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- MySQL: 8.0.23
- NebulaGraph: 3.6.0. Deploy NebulaGraph with Docker Compose.

#### PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- Learn about the Schema created in NebulaGraph, including names and properties of Tags and Edge types, and more.
- The Hadoop service has been installed and started.

#### PRECAUTIONS

nebula-exchange\_spark\_2.2 supports only single table queries, not multi-table queries.

## STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	start_year int, end_year int

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

```
## Create a graph space.
nebula> CREATE SPACE basketballplayer \
    (partition_num = 10, \
    replica_factor = 1, \
    vid_type = FIXED_STRING(30));
```

## Use the graph space basketballplayer.
nebula> USE basketballplayer;

## Create the Tag player.
nebula> CREATE TAG player(name string, age int);

## Create the Tag team.
nebula> CREATE TAG team(name string);

## Create the Edge type follow.
nebula> CREATE EDGE follow(degree int);

## Create the Edge type serve.
nebula> CREATE EDGE serve(start\_year int, end\_year int);

### For more information, see Quick start workflow.

# Step 2: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set JDBC data source configuration. In this case, the copied file is called jdbc\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
{

# Spark configuration

spark: {

app: {

name: NebulaGraph Exchange 3.6.1

}
```

```
driver: {
      cores: 1
      maxResultSize: 1G
    cores: {
      max: 16
   }
 }
 # NebulaGraph configuration
 nebula: {
    address:{
      # Specify the IP addresses and ports for Graph and Meta services.
      # If there are multiple addresses, the format is "ip1:port", "ip2:port", "ip3:port".
      # Addresses are separated by commas.
      graph:["127.0.0.1:9669"]
      # the address of any of the meta services.
      # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
      meta:["127.0.0.1:9559"]
    # The account entered must have write permission for the NebulaGraph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the NebulaGraph.
    space: basketballplayer
    connection: {
      timeout: 3000
      retry: 3
    execution: {
      retry: 3
    error: {
      max: 32
      output: /tmp/errors
    rate: {
      limit: 1024
      timeout: 1000
    }
  # Processing vertexes
  tags: [
    # Set the information about the Tag player.
    {
      # The Tag name in NebulaGraph.
      name: player
      type: {
        # Specify the data source file format to JDBC.
        source: jdbc
# Specify how to import the data into NebulaGraph: Client or SST.
        sink: client
      }
      # URL of the JDBC data source. The example is MySql database.
url:"jdbc:mysql://127.0.0.1:3306/basketball?useUnicode=true&characterEncoding=utf-8"
      # JDBC driver
      driver:"com.mysql.cj.jdbc.Driver"
      # Database user name and password
      user:"root"
      password:"12345"
      # Scanning a single table to read data.
      # nebula-exchange_spark_2.2 must configure this parameter, and can additionally configure sentence.
# nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as sentence.
      table: "basketball.player"
      # Use query statement to read data.
      # nebula-exchange_spark_2.2 can configure this parameter. Multi-table queries are not supported. Only the table name needs to be written after from. The form `db.table` is not
supported.
      # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as table. Multi-table queries are supported.
      # sentence:"select playerid, age, name from player, team order by playerid"
      # (optional)Multiple connections read parameters. See https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html
      partitionColumn:playerid # optional. Must be a numeric, date, or timestamp column from the table in question.
lowerBound:1 # optional
      upperBound:5
                                    # optional
      numPartitions:5
                                     # optional
                             # The JDBC fetch size, which determines how many rows to fetch per round trip.
      fetchSize:2
      # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the NebulaGraph.
# The sequence of fields and nebula.fields must correspond to each other.
      # If multiple column names need to be specified, separate them by commas.
      fields: [age.name]
      nebula.fields: [age,name]
      # Specify a column of data in the table as the source of VIDs in the NebulaGraph.
      vertex: {
        field:plaverid
      # udf:{
```

```
separator:" "
                  oldColNames:[field-0,field-1,field-2]
                   newColName:new-field
     #
     # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
      # prefix:"tag1"
     # Performs hashing operations on VIDs of type string.
      # policy:hash
     # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
     #writeMode: TNSERT
      # Whether or not to delete the related incoming and outgoing edges of the vertices when performing a batch delete operation. This parameter takes effect when 'writeMode' is 'DELETE'.
     #deleteEdge: false
     # The number of data written to NebulaGraph in a single batch.
     batch: 256
     # The number of partitions to be created when the data is written to NebulaGraph.
     partition: 32
   # Set the information about the Tag Team.
      name: team
     type: {
       source: jdbc
       sink: client
     }
     url:"jdbc:mysql://127.0.0.1:3306/basketball?useUnicode=true&characterEncoding=utf-8"
     driver:"com.mysql.cj.jdbc.Driver
     user:root
     password:"12345"
      table:team
     sentence:"select teamid, name from team order by teamid"
     partitionColumn:teamid
      lowerBound:1
     upperBound:5
     numPartitions:5
     fetchSize:2
     fields: [name]
     nebula.fields: [name]
     vertex: {
       field: teamid
     batch: 256
     partition: 32
   3
 ]
 # Processing edges
 edges: [
   # Set the information about the Edge Type follow.
   {
     # The corresponding Edge Type name in NebulaGraph.
     name: follow
     type: {
       # Specify the data source file format to JDBC.
       source: jdbc
       # Specify how to import the Edge type data into NebulaGraph.
       # Specify how to import the data into NebulaGraph: Client or SST.
       sink: client
     }
     url:"jdbc:mysql://127.0.0.1:3306/basketball?useUnicode=true&characterEncoding=utf-8"
     driver:"com.mysql.cj.jdbc.Driver"
     user:root
     password:"12345"
     # Scanning a single table to read data.
     # nebula-exchange_spark_2.2 must configure this parameter, and can additionally configure sentence.
     # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as sentence.
table:"basketball.follow"
     # Use query statement to read data.
     # nebula-exchange_spark_2.2 can configure this parameter. Multi-table queries are not supported. Only the table name needs to be written after from. The form `db.table` is not
supported.
     # nebula-exchange_spark_2.4 and nebula-exchange_spark_3.0 can configure this parameter, but not at the same time as table. Multi-table queries are supported.
     # sentence:"select src_player,dst_player,degree from follow order by src_player"
     partitionColumn:src player
      .
lowerBound:1
     upperBound:5
     numPartitions:5
     fetchSize:2
      # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the NebulaGraph.
     # The sequence of fields and nebula.fields must correspond to each other
     # If multiple column names need to be specified, separate them by commas.
```

```
fields: [degree]
    nebula.fields: [degree]
    # In source, use a column in the follow table as the source of the edge's source vertex.
# In target, use a column in the follow table as the source of the edge's destination vertex.
    source:
      field: src player
     # udf:{
                  separator:"_"
                  oldColNames:[field-0,field-1,field-2]
     #
                  newColName:new-field
              3
     # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
     # prefix:"tag1"
     # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
      field: dst_player
     # udf:{
                  separator:"_"
                  oldColNames:[field-0,field-1,field-2]
    #
                  newColName:new-field
    # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
# prefix:"tag1"
     # Performs hashing operations on VIDs of type string.
    # policy:hash
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
     # The number of data written to NebulaGraph in a single batch.
    batch: 256
     # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
  # Set the information about the Edge Type serve.
    name: serve
    type: {
      source: jdbc
      sink: client
    }
    url:"jdbc:mysql://127.0.0.1:3306/basketball?useUnicode=true&characterEncoding=utf-8"
    driver:"com.mysql.cj.jdbc.Driver"
    user:root
    password:"12345"
    table:serve
    sentence:"select playerid,teamid,start_year,end_year from serve order by playerid"
    partitionColumn:playerid
     .
lowerBound:1
    upperBound:5
    numPartitions:5
    fetchSize:2
     fields: [start_year,end_year]
    nebula.fields: [start_year,end_year]
    source: {
      field: playerid
    }
    target: {
      field: teamid
    batch: 256
    partition: 32
  }
]
```

Step 3: Import data into NebulaGraph

}

Run the following command to import general JDBC data into NebulaGraph. For a description of the parameters, see Options for import.

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar\_path> -c <jdbc\_application.conf\_path>

Q Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange-3.6.1.jar -c /root/nebula-exchange/nebula-exchange/target/classes/jdbc\_application.conf

You can search for batchSuccess.<tag\_name/edge\_name> in the command output to check the number of successes. For example, batchSuccess.follow: 300.

Step 4: (optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 5: (optional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

Last update: November 7, 2023

## Import data from SST files

This topic provides an example of how to generate the data from the data source into an SST (Sorted String Table) file and save it on HDFS, and then import it into NebulaGraph. The sample data source is a CSV file.

PRECAUTIONS

- The SST file can be imported only in Linux.
- The default value of the property is not supported.

BACKGROUND INFORMATION

Exchange supports two data import modes:

- Import the data from the data source directly into NebulaGraph as **nGQL** statements.
- Generate the SST file from the data source, and use Console to import the SST file into NebulaGraph.

The following describes the scenarios, implementation methods, prerequisites, and steps for generating an SST file and importing data.

SCENARIOS

• Suitable for online services, because the generation almost does not affect services (just reads the Schema), and the import speed is fast.

# Caution

Although the import speed is fast, write operations in the corresponding space are blocked during the import period (about 10 seconds). Therefore, you are advised to import data in off-peak hours.

• Suitable for scenarios with a large amount of data from data sources for its fast import speed.

IMPLEMENTATION METHODS

The underlying code in NebulaGraph uses RocksDB as the key-value storage engine. RocksDB is a storage engine based on the hard disk, providing a series of APIs for creating and importing SST files to help quickly import massive data.

The SST file is an internal file containing an arbitrarily long set of ordered key-value pairs for efficient storage of large amounts of key-value data. The entire process of generating SST files is mainly done by Exchange Reader, sstProcessor, and sstWriter. The whole data processing steps are as follows:

## 1. Reader reads data from the data source.

- 2. sstProcessor generates the SST file from the NebulaGraph's Schema information and uploads it to the HDFS. For details about the format of the SST file, see Data Storage Format.
- 3. sstWriter opens a file and inserts data. When generating SST files, keys must be written in sequence.
- 4. After the SST file is generated, RocksDB imports the SST file into NebulaGraph using the IngestExternalFile() method. For example:

When the IngestExternalFile() method is called, RocksDB copies the file to the data directory by default and blocks the RocksDB write operation. If the key range in the SST file overwrites the Memtable key range, flush the Memtable to the hard disk. After placing the SST file in an optimal location in the LSM tree, assign a global serial number to the file and turn on the write operation.

## DATA SET

This topic takes the basketballplayer dataset as an example.

## ENVIRONMENT

This example is done on MacOS. Here is the environment configuration information:

- Hardware specifications:
- CPU: 1.7 GHz Quad-Core Intel Core i7
- Memory: 16 GB
- Spark: 2.4.7, stand-alone
- Hadoop: 2.9.2, pseudo-distributed deployment
- NebulaGraph: 3.6.0.

## PREREQUISITES

Before importing data, you need to confirm the following information:

- NebulaGraph has been installed and deployed with the following information:
- IP addresses and ports of Graph and Meta services.
- The user name and password with write permission to NebulaGraph.
- --ws\_storage\_http\_port in the Meta service configuration file is the same as --ws\_http\_port in the Storage service configuration file. For example, 19779.
- --ws\_meta\_http\_port in the Graph service configuration file is the same as --ws\_http\_port in the Meta service configuration file. For example, 19559.
- The information about the Schema, including names and properties of Tags and Edge types, and more.
- Exchange has been compiled, or download the compiled .jar file directly.
- Spark has been installed.
- JDK 1.8 or the later version has been installed and the environment variable JAVA\_HOME has been configured.
- The Hadoop service has been installed and started.

#### Q Note

- To generate SST files of other data sources, see documents of the corresponding data source and check the prerequisites.
- To generate SST files only, users do not need to install the Hadoop service on the machine where the Storage service is deployed.
- To delete the SST file after the ingest (data import) operation, add the configuration -- move\_Files =true to the Storage Service configuration file.

STEPS

Step 1: Create the Schema in NebulaGraph

Analyze the data to create a Schema in NebulaGraph by following these steps:

1. Identify the Schema elements. The Schema elements in the NebulaGraph are shown in the following table.

Element	Name	Property
Tag	player	name string, age int
Tag	team	name string
Edge Type	follow	degree int
Edge Type	serve	<pre>start_year int, end_year int</pre>

2. Create a graph space **basketballplayer** in the NebulaGraph and create a Schema as shown below.

<pre>## Create a graph space nebula&gt; CREATE SPACE basketballplayer \ (partition_num = 10, \ replica_factor = 1, \ vid_type = FIXED_STRING(30));</pre>
## Use the graph space basketballplayer nebula> USE basketballplayer;
## Create the Tag player nebula> CREATE TAG player(name string, age int);
<pre>## Create the Tag team nebula&gt; CREATE TAG team(name string);</pre>
<pre>## Create the Edge type follow nebula&gt; CREATE EDGE follow(degree int);</pre>
<pre>## Create the Edge type serve nebula&gt; CREATE EDGE serve(start_year int, end_year int);</pre>

For more information, see Quick start workflow.

Step 2: Process CSV files

Confirm the following information:

1. Process CSV files to meet Schema requirements.

Note	
Exchange supports uploading CSV files with or without headers.	

## 2. Obtain the CSV file storage path.

Step 3: Modify configuration files

After Exchange is compiled, copy the conf file target/classes/application.conf to set SST data source configuration. In this example, the copied file is called sst\_application.conf. For details on each configuration item, see Parameters in the configuration file.

```
memory:1G
 }
  cores · {
   max: 16
  }
}
# NebulaGraph configuration
nebula: {
  address:{
    graph:["192.8.168.XXX:9669"]
    # the address of any of the meta services.
    # if your NebulaGraph server is in virtual network like k8s, please config the leader address of meta.
    meta:["192.8.168.XXX:9559"]
  user: root
  pswd: nebula
  space: basketballplayer
  # SST file configuration
  path:{
      # The local directory that temporarily stores generated SST files
      local:"/tmp"
      # The path for storing the SST file in the HDFS
      remote:"/sst"
      # The NameNode address of HDFS, for example, "hdfs://<ip/hostname>:<port>"
      hdfs.namenode: "hdfs://*.*.*:9000"
  }
  # The connection parameters of clients
  connection: {
    # The timeout duration of socket connection and execution. Unit: milliseconds.
    timeout: 30000
  3
  error: {
    # The maximum number of failures that will exit the application.
    max: 32
    # Failed import jobs are logged in the output path.
    output: /tmp/errors
  }
  # Use Google's RateLimiter to limit requests to NebulaGraph.
  rate: {
    # Steady throughput of RateLimiter.
    limit: 1024
     # Get the allowed timeout duration from RateLimiter. Unit: milliseconds.
    timeout: 1000
  }
}
# Processing vertices
tags: [
    # Set the information about the Tag player.
  {
    # Specify the Tag name defined in NebulaGraph.
    name: player
    type: {
    # Specify the data source file format to CSV.
     source: csv
      # Specify how to import the data into NebulaGraph: Client or SST.
      sink: sst
    }
    # Specify the path to the CSV file.
    # If the file is stored in HDFs, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://<ip/hostname>:port/xx/xx.csv".
path: "hdfs://*.*.*.*:9000/dataset/vertex_player.csv"
    # If the CSV file does not have a header, use [_c0, _c1, _c2, ..., _cn] to represent its header and indicate the columns as the source of the property values.
     # If the CSV file has a header, use the actual column name.
    fields: [_c1, _c2]
    # Specify the property name defined in NebulaGraph.
    # The sequence of fields and nebula.fields must correspond to each other.
    nebula.fields: [age, name]
    # Specify a column of data in the table as the source of VIDs in NebulaGraph.
     # The value of vertex must be consistent with the column name in the above fields or csv.fields.
    # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    vertex: {
      field:_c0
      # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # The delimiter specified. The default value is comma.
```

```
separator: ","
    # If the CSV file has a header, set the header to true.
    # If the CSV file does not have a header, set the header to false. The default value is false.
    header: false
    # Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
    #writeMode: INSERT
    # The number of data written to NebulaGraph in a single batch.
    batch: 256
    # The number of partitions to be created when the data is written to NebulaGraph.
    partition: 32
    # Whether to repartition data based on the number of partitions of graph spaces in NebulaGraph when generating the SST file.
    repartitionWithNebula: false
  3
  # Set the information about the Tag Team
    name: team
    type: {
     source: csv
     sink: sst
    }
    path: "hdfs://*.*.*.*:9000/dataset/vertex_team.csv"
    fields: [_c1]
nebula.fields: [name]
    vertex: {
     field: c0
    }
    separator: ","
   header: false
batch: 256
    partition: 32
    repartitionWithNebula: false
  # If more vertices need to be added, refer to the previous configuration to add them.
# Processing edges
edges: [
# Set the information about the Edge Type follow.
  {
    # The Edge Type name defined in NebulaGraph.
    name: follow
    type: {
    # Specify the data source file format to CSV.
     source: csv
      # Specify how to import the data into NebulaGraph: Client or SST.
     sink: sst
    }
   # Specify the path to the CSV file.
# If the file is stored in HDFS, use double quotation marks to enclose the file path, starting with hdfs://. For example, "hdfs://<ip/hostname>:port/xx/xx.csv".
    path: "hdfs://*.*.*:9000/dataset/edge_follow.csv"
    # If the CSV file does not have a header, use [_c0, _c1, _c2, ..., _cn] to represent its header and indicate the columns as the source of the property values.
    # If the CSV file has a header, use the actual column name
    fields: [ c2]
    # Specify the property name defined in NebulaGraph.
# The sequence of fields and nebula.fields must correspond to each other.
    nebula.fields: [degree]
    # Specify a column as the source for the source and destination vertices.
    # The value of vertex must be consistent with the column name in the above fields or csv.fields.
    # Currently, NebulaGraph 3.6.0 supports only strings or integers of VID.
    source: {
     field: _c0
      # Add the specified prefix to the VID. For example, if the VID is `12345`, adding the prefix `tag1` will result in `tag1_12345`. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    target: {
     field: c1
      # Add the specified prefix to the VID. For example, if the VID is '12345', adding the prefix 'tagl' will result in 'tagl_12345'. The underscore cannot be modified.
    # prefix:"tag1"
    # Performs hashing operations on VIDs of type string.
    # policy:hash
    # The delimiter specified. The default value is comma.
    separator: ",
    # (Optional) Specify a column as the source of the rank.
    #ranking: rank
    # If the CSV file has a header, set the header to true.
    # If the CSV file does not have a header, set the header to false. The default value is false.
    header: false
```

```
# Batch operation types, including INSERT, UPDATE, and DELETE. defaults to INSERT.
      #writeMode: INSERT
      # The number of data written to NebulaGraph in a single batch.
      batch: 256
      # The number of partitions to be created when the data is written to NebulaGraph.
      partition: 32
      # Whether to repartition data based on the number of partitions of graph spaces in NebulaGraph when generating the SST file.
      repartitionWithNebula: false
    # Set the information about the Edge Type serve.
      name: serve
      type: {
        source: csv
       sink: sst
      path: "hdfs://*.*.*:9000/dataset/edge_serve.csv"
      fields: [_c2,_c3]
nebula.fields: [start_year, end_year]
      source: {
   field: _c0
      target: {
field: _c1
      separator: ",
      header: false
      batch: 256
      partition: 32
      .
repartitionWithNebula: false
  # If more edges need to be added, refer to the previous configuration to add them.
}
```

Step 4: Generate the SST file

Run the following command to generate the SST file from the CSV source file. For a description of the parameters, see Options for import.

```
${SPARK_HOME}/bin/spark-submit --master "local" --conf spark.sql.shuffle.partition=<shuffle_concurrency> --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-3.6.1.jar_path> -c <sst_application.conf_path>
```

#### Q Note

When generating SST files, the shuffle operation of Spark will be involved. Note that the configuration of spark.sql.shuffle.partition should be added when you submit the command.

# Note

JAR packages are available in two ways: compiled them yourself, or download the compiled .jar file directly.

## For example:

\${SPARK\_HOME}/bin/spark-submit --master "local" --conf spark.sql.shuffle.partition=200 --class com.vesoft.nebula.exchange.Exchange /root/nebula-exchange/nebula-exchange/target/nebula-exchange/arget/nebula-exchange/target/classes/sst\_application.conf

After the task is complete, you can view the generated SST file in the /sst directory (specified by the nebula.path.remote parameter) on HDFS.

#### Q Note

If you modify the Schema, such as rebuilding the graph space, modifying the Tag, or modifying the Edge type, you need to regenerate the SST file because the SST file verifies the space ID, Tag ID, and Edge ID.

Step 5: Import the SST file

Note		
Confirm the following information before importing:		
$\bullet$ Confirm that the Hadoop service has been deployed on all the $\tt HADOOP\_HOME$ and <code>JAVA_HOME</code> .	machines where the Storage service is deployed, and configure	
• Thews_storage_http_port in the Meta service configuration file ( the Storage service configuration file. For example, both are 1	add it manually if it does not exist) is the same as the $\ws_http_port$ in 9779 .	
• Thews_meta_http_port in the Graph service configuration file (a the Meta service configuration file. For example, both are 1955	dd it manually if it does not exist) is the same as the <code>ws_http_port</code> in <code>9</code> .	
Connect to the NebulaGraph database using the client tool a	nd import the SST file as follows:	
Run the following command to select the graph space you created earlier.		

nebula> USE basketballplayer;

2. Run the following command to download the SST file:

nebula> SUBMIT JOB DOWNLOAD HDFS "hdfs://<hadoop\_address>:<hadoop\_port>/<sst\_file\_path>";

For example:

nebula> SUBMIT JOB DOWNLOAD HDFS "hdfs://\*.\*.\*:9000/sst";

3. Run the following command to import the SST file:

nebula> SUBMIT JOB INGEST;

# Note

• To download the SST file again, delete the download folder in the space ID in the data/storage/nebula directory in the NebulaGraph installation path, and then download the SST file again. If the space has multiple copies, the download folder needs to be deleted on all machines where the copies are saved.

• If there is a problem with the import and re-importing is required, re-execute SUBMIT JOB INGEST; .

Step 6: (Optional) Validate data

Users can verify that data has been imported by executing a query in the NebulaGraph client (for example, NebulaGraph Studio). For example:

LOOKUP ON player YIELD id(vertex);

Users can also run the SHOW STATS command to view statistics.

Step 7: (Conditional) Rebuild indexes in NebulaGraph

With the data imported, users can recreate and rebuild indexes in NebulaGraph. For details, see Index overview.

```
Last update: November 7, 2023
```

# 11.3.5 Exchange FAQ

## Compilation

Q: SOME PACKAGES NOT IN CENTRAL REPOSITORY FAILED TO DOWNLOAD, ERROR: COULD NOT RESOLVE DEPENDENCIES FOR PROJECT XXX

Please check the mirror part of Maven installation directory libexec/conf/settings.xml:

Check whether the value of mirrorOf is configured to \*. If it is, change it to central or \*,!SparkPackagesRepo,!bintray-streamnative-maven.

**Reason**: There are two dependency packages in Exchange's pom.xml that are not in Maven's central repository. pom.xml configures the repository address for these two dependencies. If the mirrorOf value for the mirror address configured in Maven is \*, all dependencies will be downloaded from the Central repository, causing the download to fail.

Q: UNABLE TO DOWNLOAD SNAPSHOT PACKAGES WHEN COMPILING EXCHANGE

Problem description: The system reports Could not find artifact com.vesoft:client:jar:xxx-SNAPSHOT when compiling.

Cause: There is no local Maven repository for storing or downloading SNAPSHOT packages. The default central repository in Maven only stores official releases, not development versions (SNAPSHOT).

Solution: Add the following configuration in the profiles scope of Maven's setting.xml file:

```
<profile>
<activation>
<activeByDefault>true</activeByDefault>
</activation>
</activation>
</activation>
</repositories>
</id>
<activation>
</url>
</snapshots>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</repositors>
</reposito
```

### Execution

Q: ERROR: JAVA.LANG.CLASSNOTFOUNDEXCEPTION: COM.VESOFT.NEBULA.EXCHANGE.EXCHANGE

To submit a task in Yarn-Cluster mode, run the following command, especially the two '--conf' commands in the example.

```
$SPARK_HOME/bin/spark-submit --class com.vesoft.nebula.exchange.Exchange \
--master yarn-cluster \
--files application.conf \
--conf spark.driver.extraClassPath=./ \
--conf spark.executor.extraClassPath=./ \
nebula-exchange-3.0.0.jar \
-c application.conf
```

Q: ERROR: METHOD NAME XXX NOT FOUND

Generally, the port configuration is incorrect. Check the port configuration of the Meta service, Graph service, and Storage service.

Q: ERROR: NOSUCHMETHOD, METHODNOTFOUND (EXCEPTION IN THREAD "MAIN" JAVA.LANG.NOSUCHMETHODERROR, ETC)

Most errors are caused by JAR package conflicts or version conflicts. Check whether the version of the error reporting service is the same as that used in Exchange, especially Spark, Scala, and Hive.

Q: WHEN EXCHANGE IMPORTS HIVE DATA, ERROR: EXCEPTION IN THREAD "MAIN" ORG.APACHE.SPARK.SQL.ANALYSISEXCEPTION: TABLE OR VIEW NOT FOUND

Check whether the -h parameter is omitted in the command for submitting the Exchange task and whether the table and database are correct, and run the user-configured exec statement in spark-SQL to verify the correctness of the exec statement.

Q: RUN ERROR: COM.FACEBOOK.THRIFT.PROTOCOL.TPROTOCOLEXCEPTION: EXPECTED PROTOCOL ID XXX

Check that the NebulaGraph service port is configured correctly.

- For source, RPM, or DEB installations, configure the port number corresponding to --port in the configuration file for each service.
- For docker installation, configure the docker mapped port number as follows:

Execute docker-compose ps in the nebula-docker-compose directory, for example:

\$ docker-compose ps Name	Command	State	Ports
nebula-docker-compose_graphd_1	/usr/local/nebula/bin/nebu	Up (healthy)	0.0.0.0:33205->19669/tcp, 0.0.0.0:33204->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1	./bin/nebula-metadflagf	Up (healthy)	0.0.0.0:33165->19559/tcp, 0.0.0.0:33162->19560/tcp, 0.0.0.0:33167->9559/tcp, 9560/tcp
nebula-docker-compose_metad1_1	./bin/nebula-metadflagf	Up (healthy)	0.0.0.0:33166->19559/tcp, 0.0.0.0:33163->19560/tcp, 0.0.0.0:33168->9559/tcp, 9560/tcp
nebula-docker-compose_metad2_1	./bin/nebula-metadflagf	Up (healthy)	0.0.0.0:33161->19559/tcp, 0.0.0.0:33160->19560/tcp, 0.0.0.0:33164->9559/tcp, 9560/tcp
nebula-docker-compose_storaged0_1	./bin/nebula-storagedfl	Up (healthy)	0.0.0.0:33180->19779/tcp, 0.0.0.0:33178->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33183->9779/tcp, 9780/
tcp			
nebula-docker-compose_storaged1_1	./bin/nebula-storagedfl	Up (healthy)	0.0.0.0:33175->19779/tcp, 0.0.0.0:33172->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33177->9779/tcp, 9780/
tcp			
nebula-docker-compose_storaged2_1	./bin/nebula-storagedfl	Up (healthy)	0.0.0.0:33184->19779/tcp, 0.0.0.0:33181->19780/tcp, 9777/tcp, 9778/tcp, 0.0.0.0:33185->9779/tcp, 9780/
tcp			

Check the Ports column to find the docker mapped port number, for example:

- The port number available for Graph service is 9669.
- The port number for Meta service are 33167, 33168, 33164.
- The port number for Storage service are 33183, 33177, 33185.

Q: ERROR: EXCEPTION IN THREAD "MAIN" COM.FACEBOOK.THRIFT.PROTOCOL.TPROTOCOLEXCEPTION: THE FIELD 'CODE' HAS BEEN ASSIGNED THE INVALID VALUE -4

Check whether the version of Exchange is the same as that of NebulaGraph. For more information, see Limitations.

Q: HOW TO CORRECT THE ENCODING ERROR WHEN IMPORTING DATA IN A SPARK ENVIRONMENT?

It may happen if the property value of the data contains Chinese characters. The solution is to add the following options before the JAR package path in the import command:

--conf spark.driver.extraJavaOptions=-Dfile.encoding=utf-8 --conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8

#### Namely:

```
<spark_install_path>/bin/spark-submit --master "local" \
--conf spark.driver.extraJava0ptions=-Dfile.encoding=utf-8 \
--conf spark.executor.extraJava0ptions=-Dfile.encoding=utf-8 \
--class com.vesoft.nebula.exchange \
<nebula-exchange-3.x.y.jar_path> -c <application.conf_path>
```

## In YARN, use the following command:

```
<spark_install_path>/bin/spark-submit \
--class com.vesoft.nebula.exchange.Exchange \
--master yarn-cluster \
--files <application.conf_path> \
--conf spark.driver.extraClassPath=./ \
--conf spark.executor.extraClassPath=./ \
--conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8 \
<-nebula-exchange-3.x.y.jar_path> \
-c application.conf
```

Q: ORG.ROCKSDB.ROCKSDBEXCEPTION: WHILE OPEN A FILE FOR APPENDING: /PATH/SST/1-XXX.SST: NO SUCH FILE OR DIRECTORY

Solution:

- 1. Check if /path exists. If not, or if the path is set incorrectly, create or correct it.
- 2. Check if Spark's current user on each machine has the operation permission on /path. If not, grant the permission.

## Configuration

Q: WHICH CONFIGURATION FIELDS WILL AFFECT IMPORT PERFORMANCE?

- batch: The number of data contained in each nGQL statement sent to the NebulaGraph service.
- partition: The number of partitions to be created when the data is written to NebulaGraph, indicating the number of concurrent data imports.
- nebula.rate: Get a token from the token bucket before sending a request to NebulaGraph.
  - limit: Represents the size of the token bucket.
  - timeout: Represents the timeout period for obtaining the token.

The values of these four parameters can be adjusted appropriately according to the machine performance. If the leader of the Storage service changes during the import process, you can adjust the values of these four parameters to reduce the import speed.

## Others

Q: WHICH VERSIONS OF NEBULAGRAPH ARE SUPPORTED BY EXCHANGE?

```
See Limitations.
```

```
Q: WHAT IS THE RELATIONSHIP BETWEEN EXCHANGE AND SPARK WRITER?
```

Exchange is the Spark application developed based on Spark Writer. Both are suitable for bulk migration of cluster data to NebulaGraph in a distributed environment, but later maintenance work will be focused on Exchange. Compared with Spark Writer, Exchange has the following improvements:

- It supports more abundant data sources, such as MySQL, Neo4j, Hive, HBase, Kafka, Pulsar, etc.
- It fixed some problems of Spark Writer. For example, when Spark reads data from HDFS, the default source data is String, which may be different from the NebulaGraph's Schema. So Exchange adds automatic data type matching and type conversion. When the data type in the NebulaGraph's Schema is non-String (e.g. double), Exchange converts the source data of String type to the corresponding type.

Last update: December 18, 2023

# 12. Connectors

# 12.1 NebulaGraph Spark Connector

NebulaGraph Spark Connector is a Spark connector application for reading and writing NebulaGraph data in Spark standard format. NebulaGraph Spark Connector consists of two parts: Reader and Writer.

# • Reader

Provides a Spark SQL interface. This interface can be used to read NebulaGraph data. It reads one vertex or edge type data at a time and assemble the result into a Spark DataFrame.

# • Writer

Provides a Spark SQL interface. This interface can be used to write DataFrames into NebulaGraph in a row-by-row or batchimport way.

For more information, see NebulaGraph Spark Connector.

# 12.1.1 Version compatibility

The correspondence between the NebulaGraph Spark Connector version, the NebulaGraph core version and the Spark version is as follows.

Spark Connector version	NebulaGraph version	Spark version
nebula-spark-connector_3.0-3.0-SNAPSHOT.jar	nightly	3.x
nebula-spark-connector_2.2-3.0-SNAPSHOT.jar	nightly	2.2.x
nebula-spark-connector-3.0-SNAPSHOT.jar	nightly	2.4.x
nebula-spark-connector_3.0-3.6.0.jar	3.x	3.x
nebula-spark-connector_2.2-3.6.0.jar	3.x	2.2.x
nebula-spark-connector-3.6.0.jar	3.x	2.4.x
nebula-spark-connector_2.2-3.4.0.jar	3.x	2.2.x
nebula-spark-connector-3.4.0.jar	3.x	2.4.x
nebula-spark-connector_2.2-3.3.0.jar	3.x	2.2.x
nebula-spark-connector-3.3.0.jar	3.x	2.4.x
nebula-spark-connector-3.0.0.jar	3.x	2.4.x
nebula-spark-connector-2.6.1.jar	2.6.0, 2.6.1	2.4.x
nebula-spark-connector-2.6.0.jar	2.6.0, 2.6.1	2.4.x
nebula-spark-connector-2.5.1.jar	2.5.0, 2.5.1	2.4.x
nebula-spark-connector-2.5.0.jar	2.5.0, 2.5.1	2.4.x
nebula-spark-connector-2.1.0.jar	2.0.0, 2.0.1	2.4.x
nebula-spark-connector-2.0.1.jar	2.0.0, 2.0.1	2.4.x
nebula-spark-connector-2.0.0.jar	2.0.0, 2.0.1	2.4.x

## 12.1.2 Use cases

NebulaGraph Spark Connector applies to the following scenarios:

- Read data from NebulaGraph for analysis and computation.
- Write data back to NebulaGraph after analysis and computation.
- Migrate the data of NebulaGraph.
- Graph computing with NebulaGraph Algorithm.

## 12.1.3 Benefits

The features of NebulaGraph Spark Connector 3.6.0 are as follows:

- Supports multiple connection settings, such as timeout period, number of connection retries, number of execution retries, etc.
- Supports multiple settings for data writing, such as setting the corresponding column as vertex ID, starting vertex ID, destination vertex ID or attributes.
- Supports non-attribute reading and full attribute reading.
- Supports reading NebulaGraph data into VertexRDD and EdgeRDD, and supports non-Long vertex IDs.
- Unifies the extended data source of SparkSQL, and uses DataSourceV2 to extend NebulaGraph data.
- Three write modes, insert, update and delete, are supported. insert mode will insert (overwrite) data, update mode will only update existing data, and delete mode will only delete data.

## 12.1.4 Release note

Release

# 12.1.5 Get NebulaGraph Spark Connector

## Compile and package

1. Clone repository nebula-spark-connector.

\$ git clone -b release-3.6 https://github.com/vesoft-inc/nebula-spark-connector.git

- 2. Enter the nebula-spark-connector directory.
- 3. Compile and package. The procedure varies with Spark versions.

# Note

Spark of the corresponding version has been installed.

## - Spark 2.4

```
```bash
$ mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true -pl nebula-spark-connector -am -Pscala-2.11 -Pspark-2.4
```

# - Spark 2.2

```bash \$ mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true -pl nebula-spark-connector\_2.2 -am -Pscala-2.11 -Pspark-2.2

# - Spark 3.x

```bash
\$ mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true -pl nebula-spark-connector\_3.0 -am -Pscala-2.12 -Pspark-3.0

After compilation, a file similar to nebula-spark-connector-3.6.0-SHANPSHOT.jar is generated in the directory target of the folder.

#### Download maven remote repository

#### Download

## 12.1.6 How to use

When using NebulaGraph Spark Connector to reading and writing NebulaGraph data, You can refer to the following code.

```
# Read vertex and edge data from NebulaGraph.
spark.read.nebula().loadVerticesToDF()
spark.read.nebula().loadEdgesToDF()
# Write dataframe data into NebulaGraph as vertex and edges
```

dataframe.write.nebula().writeVertices()
dataframe.write.nebula().writeEdges()

nebula() receives two configuration parameters, including connection configuration and read-write configuration.

# Note

If the value of the properties contains Chinese characters, the encoding error may appear. Please add the following options when submitting the Spark task:

--conf spark.driver.extraJavaOptions=-Dfile.encoding=utf-8
--conf spark.executor.extraJavaOptions=-Dfile.encoding=utf-8

## Reading data from NebulaGraph

```
val config = NebulaConnectionConfig
  .builder()
  .withMetaAddress("127.0.0.1:9559")
  .withConenctionRetry(2)
  .withExecuteRetry(2)
  .withTimeout(6000)
  .build()
val nebulaReadVertexConfig: ReadNebulaConfig = ReadNebulaConfig
  .builder()
  .withSpace("test")
  .withLabel("person")
  .withNoColumn(false)
  .withReturnCols(List("birthday"))
  .withLimit(10)
  .withPartitionNum(10)
  .build()
val vertex = spark.read.nebula(config, nebulaReadVertexConfig).loadVerticesToDF()
val nebulaReadEdgeConfig: ReadNebulaConfig = ReadNebulaConfig
  .builder()
  .withSpace("test")
  .withLabel("knows")
  .withNoColumn(false)
  .withReturnCols(List("degree"))
  .withLimit(10)
  .withPartitionNum(10)
```
.build()

val edge = spark.read.nebula(config, nebulaReadEdgeConfig).loadEdgesToDF()

• NebulaConnectionConfig is the configuration for connecting to NebulaGraph, as described below.

| Parameter           | Required | Description                                                                                                                                                                                                             |
|---------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| withMetaAddress     | Yes      | Specifies the IP addresses and ports of all Meta Services. Separate multiple addresses with commas. The format is <code>ip1:port1,ip2:port2,</code> . Read data is no need to configure <code>withGraphAddress</code> . |
| withConnectionRetry | No       | The number of retries that the NebulaGraph Java Client connected to NebulaGraph. The default value is 1.                                                                                                                |
| withExecuteRetry    | No       | The number of retries that the NebulaGraph Java Client executed query statements. The default value is 1.                                                                                                               |
| withTimeout         | No       | The timeout for the NebulaGraph Java Client request response. The default value is 6000 , Unit: ms.                                                                                                                     |

• ReadNebulaConfig is the configuration to read NebulaGraph data, as described below.

| Parameter        | Required | Description                                                                                                                                                                                                        |
|------------------|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| withSpace        | Yes      | NebulaGraph space name.                                                                                                                                                                                            |
| withLabel        | Yes      | The Tag or Edge type name within the NebulaGraph space.                                                                                                                                                            |
| withNoColumn     | No       | Whether the property is not read. The default value is <code>false</code> , read property. If the value is <code>true</code> , the property is not read, the <code>withReturnCols</code> configuration is invalid. |
| withReturnCols   | No       | Configures the set of properties for vertex or edges to read. the format is List(property1,property2,) , The default value is List() , indicating that all properties are read.                                    |
| withLimit        | No       | Configure the number of rows of data read from the server by the NebulaGraph Java Storage Client at a time. The default value is 1000.                                                                             |
| withPartitionNum | No       | Configures the number of Spark partitions to read the NebulaGraph data. The default value is 100. This value should not exceed the number of slices in the graph space (partition_num).                            |

### Write data into NebulaGraph

### Note

The values of columns in a dataframe are automatically written to NebulaGraph as property values.

```
val config = NebulaConnectionConfig
.builder()
.withKetaAddress("127.0.0.1:9559")
.withCraphAddress("127.0.0.1:9669")
.withConenctionRetry(2)
.build()
val nebulaWriteVertexConfig: WriteNebulaVertexConfig = WriteNebulaVertexConfig
.builder()
.withSpace("test")
.withTag("person")
.withVidField("id")
.withVidField("id")
.withVidField("id")
.withVidField("id")
.withVidAsProp(true)
.withWidAsProp(true)
.withBach(1000)
.build()
df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()
```

val nebulaWriteEdgeConfig: WriteNebulaEdgeConfig = WriteNebulaEdgeConfig
.builder()

.withSpace("test") .withSque("friend") .withSrcDdField("src") .withSrcDoLicy(null) .withDstDdField("dst") .withDstNeField("degree") .withSrcAsProperty(true) .withBackFroperty(true) .withBackProperty(true) .withBackPro

The default write mode is insert, which can be changed to update or delete via withWriteMode configuration:

```
val config = NebulaConnectionConfig
.builder()
.withMetaAddress("127.0.0.1:9559")
.withGraphAddress("127.0.0.1:9669")
.build()
val nebulaWriteVertexConfig = WriteNebulaVertexConfig
.builder()
.withSpace("test")
.withTag("person")
.withNidfield("id")
.withNidfield("id")
.withNidfield("id")
.withWriteWode(WriteNode.UPDATE)
```

.build() df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()

| • | NebulaConnectionConfig | is the configuration for | connecting to the nebula | graph, as described below. |
|---|------------------------|--------------------------|--------------------------|----------------------------|
|   |                        |                          |                          |                            |

| Param    | eter         | Required | Description                                                                                                                                           |
|----------|--------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| withMeta | Address      | Yes      | Specifies the IP addresses and ports of all Meta Services. Separate multiple addresses with commas. The format is <code>ip1:port1,ip2:port2,</code> . |
| withGrap | bhAddress    | Yes      | Specifies the IP addresses and ports of Graph Services. Separate multiple addresses with commas. The format is <code>ip1:port1,ip2:port2,</code> .    |
| withConn | nectionRetry | No       | Number of retries that the NebulaGraph Java Client connected to NebulaGraph.<br>The default value is 1.                                               |

• WriteNebulaVertexConfig is the configuration of the write vertex, as described below.

| Parameter      | Required | Description                                                                                                                                                                                     |
|----------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| withSpace      | Yes      | NebulaGraph space name.                                                                                                                                                                         |
| withTag        | Yes      | The Tag name that needs to be associated when a vertex is written.                                                                                                                              |
| withVidField   | Yes      | The column in the DataFrame as the vertex ID.                                                                                                                                                   |
| withVidPolicy  | No       | When writing the vertex ID, NebulaGraph use mapping function, supports HASH only.<br>No mapping is performed by default.                                                                        |
| withVidAsProp  | No       | Whether the column in the DataFrame that is the vertex ID is also written as an property. The default value is false. If set to true, make sure the Tag has the same property name as VidField. |
| withUser       | No       | NebulaGraph username. If authentication is disabled, you do not need to configure the username and password.                                                                                    |
| withPasswd     | No       | The password for the NebulaGraph username.                                                                                                                                                      |
| withBatch      | Yes      | The number of rows of data written at a time. The default value is 1000.                                                                                                                        |
| withWriteMode  | No       | Write mode. The optional values are $% \left( {{\left[ {{{\left[ {{{_{{\rm{mod}}}}} \right]}_{\rm{mod}}}} \right]}_{\rm{mod}}} \right)}$ delete . The default value is insert .                 |
| withDeleteEdge | No       | Whether to delete the related edges synchronously when deleting a vertex. The default value is false. It takes effect when withWriteMode is delete.                                             |

• WriteNebulaEdgeConfig is the configuration of the write edge, as described below.

| Parameter          | Required | Description                                                                                                                                                                                                                                   |
|--------------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| withSpace          | Yes      | NebulaGraph space name.                                                                                                                                                                                                                       |
| withEdge           | Yes      | The Edge type name that needs to be associated when a edge is written.                                                                                                                                                                        |
| withSrcIdField     | Yes      | The column in the DataFrame as the vertex ID.                                                                                                                                                                                                 |
| withSrcPolicy      | No       | When writing the starting vertex ID, NebulaGraph use mapping function, supports HASH only. No mapping is performed by default.                                                                                                                |
| withDstIdField     | Yes      | The column in the DataFrame that serves as the destination vertex.                                                                                                                                                                            |
| withDstPolicy      | No       | When writing the destination vertex ID, NebulaGraph use mapping function, supports HASH only. No mapping is performed by default.                                                                                                             |
| withRankField      | No       | The column in the DataFrame as the rank. Rank is not written by default.                                                                                                                                                                      |
| withSrcAsProperty  | No       | Whether the column in the DataFrame that is the starting vertex is also written as an property. The default value is false. If set to true, make sure Edge type has the same property name as SrcIdField.                                     |
| withDstAsProperty  | No       | Whether column that are destination vertex in the DataFrame are also written as property. The default value is <code>false</code> . If set to <code>true</code> , make sure Edge type has the same property name as <code>DstIdField</code> . |
| withRankAsProperty | No       | Whether column in the DataFrame that is the rank is also written as property. The default value is false. If set to true, make sure Edge type has the same property name as RankField.                                                        |
| withUser           | No       | NebulaGraph username. If authentication is disabled, you do not need to configure the username and password.                                                                                                                                  |
| withPasswd         | No       | The password for the NebulaGraph username.                                                                                                                                                                                                    |
| withBatch          | Yes      | The number of rows of data written at a time. The default value is $\ensuremath{1000}$ .                                                                                                                                                      |
| withWriteMode      | No       | Write mode. The optional values are $% \left( {{\left[ {{n_{\rm{s}}} \right]}_{\rm{s}}}} \right)$ insert , update and delete . The default value is insert .                                                                                  |

Last update: April 10, 2024

## 12.2 NebulaGraph Flink Connector

NebulaGraph Flink Connector is a connector that helps Flink users quickly access NebulaGraph. NebulaGraph Flink Connector supports reading data from the NebulaGraph database or writing other external data to the NebulaGraph database.

For more information, see NebulaGraph Flink Connector.

### 12.2.1 Use cases

NebulaGraph Flink Connector applies to the following scenarios:

- Read data from NebulaGraph for analysis and computation.
- Write data back to NebulaGraph after analysis and computation.
- Migrate the data of NebulaGraph.

### 12.2.2 Release note

### Release

### 12.2.3 Version compatibility

The correspondence between the NebulaGraph Flink Connector version and the NebulaGraph core version is as follows.

| Flink Connector version | NebulaGraph version |
|-------------------------|---------------------|
| 3.0-SNAPSHOT            | nightly             |
| 3.5.0                   | 3.x.x               |
| 3.3.0                   | 3.x.x               |
| 3.0.0                   | 3.x.x               |
| 2.6.1                   | 2.6.0, 2.6.1        |
| 2.6.0                   | 2.6.0, 2.6.1        |
| 2.5.0                   | 2.5.0, 2.5.1        |
| 2.0.0                   | 2.0.0, 2.0.1        |

### 12.2.4 Prerequisites

• Java 8 or later is installed.

• Flink 1.11.x is installed.

### 12.2.5 Get NebulaGraph Flink Connector

### **Configure Maven dependency**

Add the following dependency to the Maven configuration file pom.xml to automatically obtain the Flink Connector.

```
<dependency>
    <groupId>com.vesoft</groupId>
    <artifactId>nebula-flink-connector</artifactId>
    <version>3.5.0</version>
</dependency>
```

#### Compile and package

Follow the steps below to compile and package the Flink Connector.

1. Clone repository nebula-flink-connector.

\$ git clone -b release-3.5 https://github.com/vesoft-inc/nebula-flink-connector.git

- 2. Enter the nebula-flink-connector directory.
- 3. Compile and package.

\$ mvn clean package -Dmaven.test.skip=true

After compilation, a file similar to nebula-flink-connector-3.5.0.jar is generated in the directory connector/target of the folder.

#### 12.2.6 How to use

#### Write data into NebulaGraph

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
NebulaClientOptions nebulaClientOptions = new NebulaClientOptions.NebulaClientOptionsBuilder()
                 .setGraphAddress("127.0.0.1:9669")
                  .setMetaAddress("127.0.0.1:9559")
                 .build();
NebulaGraphConnectionProvider graphConnectionProvider = new NebulaGraphConnectionProvider(nebulaClientOptions);
NebulaMetaConnectionProvider metaConnectionProvider = new NebulaMetaConnectionProvider(nebulaClientOptions);
VertexExecutionOptions executionOptions = new VertexExecutionOptions.ExecutionOptionBuilder()
                 .setGraphSpace("flinkSink")
                 .setTag("player")
.setIdIndex(0)
                 .setFields(Arrays.asList("name", "age"))
                 .setPositions(Arrays.asList(1, 2))
                 .setBatchSize(2)
                 .build();
NebulaVertexBatchOutputFormat outputFormat = new NebulaVertexBatchOutputFormat(
                 {\tt graph Connection Provider, meta Connection Provider, execution Options);}
NebulaSinkFunction<Row> nebulaSinkFunction = new NebulaSinkFunction<>(outputFormat);
DataStream<Row> dataStream = playerSource.map(row ->
            Row record = new org.apache.flink.types.Row(row.size());
for (int i = 0; i < row.size(); i++) {</pre>
                 record.setField(i, row.get(i));
            return record;
dataStream.addSink(nebulaSinkFunction);
env.execute("write nebula")
```

#### Read data from NebulaGraph

```
NebulaClientOptions nebulaClientOptions = new NebulaClientOptions.NebulaClientOptionsBuilder()
         setMetaAddress("127.0.0.1:9559")
         build();
storageConnectionProvider = new NebulaStorageConnectionProvider(nebulaClientOptions);
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.setParallelism(1);
VertexExecutionOptions vertexExecutionOptions = new VertexExecutionOptions.ExecutionOptionBuilder()
         setGraphSpace("flinkSource")
         .setTag("person")
.setNoColumn(false)
         .setFields(Arrays.asList())
         .setLimit(100)
         .build():
NebulaSourceFunction sourceFunction = new NebulaSourceFunction(storageConnectionProvider)
         .setExecutionOptions(vertexExecutionOptions);
DataStreamSource<BaseTableRow> dataStreamSource = env.addSource(sourceFunction):
dataStreamSource.map(row ->
    List<ValueWrapper> values = row.getValues();
    Row record = new Row(15):
    record.setField(0, values.get(0).asLong());
    record.setField(1, values.get(1).asString());
    record.setField(2, values.get(2).asString());
    record.setField(3, values.get(3).asLong());
    record.setField(4, values.get(4).asLong());
   record.setField(5, values.get(5).asLong());
record.setField(6, values.get(6).asLong());
    record.setField(7, values.get(7).asDate());
```

```
record.setField(8, values.get(8).asDateTime().getUTCDateTimeStr());
record.setField(9, values.get(9).asLong());
record.setField(10, values.get(10).asBoolean());
record.setField(11, values.get(11).asDouble());
record.setField(12, values.get(12).asDouble());
record.setField(13, values.get(13).asTime().getUTCTimeStr());
return record;
}).print();
env.execute("NebulaStreamSource");
```

### Parameter descriptions

• NebulaClientOptions is the configuration for connecting to NebulaGraph, as described below.

Parameter	Туре	Required	Description
setGraphAddress	String	Yes	The Graph service address of NebulaGraph.
setMetaAddress	String	Yes	The Meta service address of NebulaGraph.

• VertexExecutionOptions is the configuration for reading vertices from and writing vertices to NebulaGraph, as described below.

Parameter	Туре	Required	Description
setGraphSpace	String	Yes	The graph space name.
setTag	String	Yes	The tag name.
setIdIndex	Int	Yes	The subscript of the stream data field that is used as the VID when writing data to NebulaGraph.
setFields	List	Yes	A collection of the property names of a tag. It is used to write data to or read data from NebulaGraph. Make sure the setNoColumn is false when reading data; otherwise, the configuration is invalid. If this parameter is empty, all properties are read when reading data from NebulaGraph.
setPositions	List	Yes	A collection of the subscripts of the stream data fields. It indicates that the corresponding field values are written to NebulaGraph as property values. This parameter needs to correspond to setFields.
setBatchSize	String	No	The maximum number of data records to write to NebulaGraph at a time. The default value is 2000.
setNoColumn	String	No	The properties are not to be read if set to true when reading data. The default value is false.
setLimit	String	No	The maximum number of data records to pull at a time when reading data. The default value is 2000.

. EdgeExecutionOptions is the configuration for reading edges from and writing edges to NebulaGraph, as described below.

Parameter	Туре	Required	Description
setGraphSpace	String	Yes	The graph space name.
setEdge	String	Yes	The edge type name.
setSrcIndex	Int	Yes	The subscript of the stream data field that is used as the VID of the source vertex when writing data to NebulaGraph.
setDstIndex	Int	Yes	The subscript of the stream data field that is used as the VID of the destination vertex when writing data to NebulaGraph.
setRankIndex	Int	Yes	The subscript of the stream data field that is used as the rank of the edge when writing data to NebulaGraph.
setFields	List	Yes	A collection of the property names of an edge type. It is used to write data to or read data from NebulaGraph. Make sure the setNoColumn is false when reading data; otherwise, the configuration is invalid. If this parameter is empty, all properties are read when reading data from NebulaGraph.
setPositions	List	Yes	A collection of the subscripts of the stream data fields. It indicates that the corresponding field values are written to NebulaGraph as property values. This parameter needs to correspond to setFields.
setBatchSize	String	No	The maximum number of data records to write to NebulaGraph at a time. The default value is 2000 .
setNoColumn	String	No	The properties are not to be read if set to true when reading data. The default value is $\ensuremath{false}$ .
setLimit	String	No	The maximum number of data records to pull at a time when reading data. The default value is 2000 .

### 12.2.7 Example

### 1. Create a graph space.

```
"127.0.0.1:9559",
"127.0.0.1:9669");
.build();
TableEnvironment tableEnv = TableEnvironment.create(settings);
tableEnv.registerCatalog(CATALOG_NAME, nebulaCatalog);
tableEnv.useCatalog(CATALOG_NAME);
```

tableEnv.executeSql(createDataBase);

### 2. Create a tag.

+ " col7 DATE," + " col8 TIMESTAMP," + " col9 BIGINT," + " col10 BOOLEAN," + " col11 DOUBLE," + " col12 DOUBLE," + " col13 TIME," + " col14 STRING" + ") WITH (" + " 'connector' = 'nebula'," + " 'meta-address' = '127.0.0.1:9559'," + " 'graph-address' = '127.0.0.1:9669'," + " 'username' = 'root'," + " 'graph-space' = 'flink\_test'," + " 'label-name' = 'person'" + ")"

```
);
```

### 3. Create an edge type.

4. Queries the data of an edge type and inserts it into another edge type.

```
Table table = tableEnvironment.sqlQuery("SELECT * FROM `friend`");
table.executeInsert("`friend_sink`").await();
```

Last update: April 10, 2024

# 13. Best practices

### 13.1 Compaction

This topic gives some information about compaction.

In NebulaGraph, Compaction is the most important background process and has an important effect on performance.

Compaction reads the data that is written on the hard disk, then re-organizes the data structure and the indexes, and then writes back to the hard disk. The read performance can increase by times after compaction. Thus, to get high read performance, trigger compaction (full compaction) manually when writing a large amount of data into Nebula Graph.

### Note

Note that compaction leads to long-time hard disk IO. We suggest that users do compaction during off-peak hours (for example, early morning).

NebulaGraph has two types of compaction : automatic compaction and full compaction .

### 13.1.1 Automatic compaction

Automatic compaction is automatically triggered when the system reads data, writes data, or the system restarts. The read performance can increase in a short time. Automatic compaction is enabled by default. But once triggered during peak hours, it can cause unexpected IO occupancy that has an unwanted effect on the performance.

### 13.1.2 Full compaction

Q Note

Full compaction enables large-scale background operations for a graph space such as merging files, deleting the data expired by TTL. This operation needs to be initiated manually. Use the following statements to enable full compaction :

We recommend you to do the full compaction during off-peak hours because full compaction has a lot of IO operations.

nebula> USE <your\_graph\_space>;
nebula> SUBMIT JOB COMPACT;

The preceding statement returns the job ID. To show the compaction progress, use the following statement:

nebula> SHOW JOB <job\_id>;

### 13.1.3 Operation suggestions

These are some operation suggestions to keep Nebula Graph performing well.

- After data import is done, run SUBMIT JOB COMPACT.
- Run SUBMIT JOB COMPACT periodically during off-peak hours (e.g. early morning).
- To control the write traffic limitation for compactions, set the following parameter in the nebula-storaged.conf configuration file.

# Note

This parameter limits the rate of all writes including normal writes and compaction writes.

# Limit the write rate to 20MB/s.
--rocksdb\_rate\_limit=20 (in MB/s)

### 13.1.4 FAQ

### "Where are the logs related to Compaction stored?"

By default, the logs are stored under the LOG file in the /usr/local/nebula/data/storage/nebula/{1}/data/ directory, or similar to LOG.old. 1625797988509303. You can find the following content.

** Compa	** Compaction Stats [default] **																	
Level	Files	Size	Score	Read(GB)	Rn(GB)	Rnp1(GB)	Write(GB)	Wnew(GB)	Moved(GB)	W-Amp	Rd(MB/s)	Wr(MB/s)	Comp(sec)	CompMergeCPU(sec)	Comp(cnt)	Avg(sec)	KeyIn K	eyDrop
L0	2/0	2.46 KB	0.5	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.53	0.51	2	0.264	0	0
Sum	2/0	2.46 KB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.53	0.51	2	0.264	0	0
Int	0/0	0.00 KB	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00	0.00	0	0.000	0	0

If the number of L0 files is large, the read performance will be greatly affected and compaction can be triggered.

### "Can I do full compactions for multiple graph spaces at the same time?"

Yes, you can. But the IO is much larger at this time and the efficiency may be affected.

### "How much time does it take for full compactions ?"

When rocksdb\_rate\_limit is set to 20, you can estimate the full compaction time by dividing the hard disk usage by the rocksdb\_rate\_limit. If you do not set the rocksdb\_rate\_limit value, the empirical value is around 50 MB/s.

### "Can I modify --rocksdb\_rate\_limit dynamically?"

No, you cannot.

### "Can I stop a full compaction after it starts?"

No, you cannot. When you start a full compaction, you have to wait till it is done. This is the limitation of RocksDB.

Last update: November 3, 2023

### 13.2 Storage load balance

You can use the SUBMIT JOB BALANCE statement to balance the distribution of partitions and Raft leaders, or clear some Storage servers for easy maintenance. For details, see SUBMIT JOB BALANCE.



### 13.2.1 Balance leader distribution

To balance the raft leaders, run SUBMIT JOB BALANCE LEADER. It will start a job to balance the distribution of all the storage leaders in all graph spaces.

### Example

nebula> SUBMIT JOB BALANCE LEADER;

Run SHOW HOSTS to check the balance result.

nebula> SHOW HOSTS;				+
Host   Port   Stat	tus   Leader count	Leader distribution	Partition distribution	Version
	LINE"   8     LINE"   3     LINE"   0     LINE"   0     LINE"   0	"basketballplayer:3" "basketballplayer:3" "basketballplayer:2" "basketballplayer:2" "basketballplayer:2"	"basketballplayer:8" "basketballplayer:8" "basketballplayer:7" "basketballplayer:7"	"3.6.0"   "3.6.0"   "3.6.0"   "3.6.0"   "3.6.0"
++++++++				+

# Caution

During leader partition replica switching in NebulaGraph, the leader replicas will be temporarily prohibited from being written to until the switch is completed. If there are a large number of write requests during the switching period, it will result in a request error (Storage Error  $E_RPC_FAILURE$ ). See FAQ for error handling methods.

You can set the value of raft\_heartbeat\_interval\_secs in the Storage configuration file to control the timeout period for leader replica switching. For more information on the configuration file, see Storage configuration file.

### 13.3 Graph data modeling suggestions

This topic provides general suggestions for modeling data in NebulaGraph.

### Note

The following suggestions may not apply to some special scenarios. In these cases, find help in the NebulaGraph community.

### 13.3.1 Model for performance

There is no perfect method to model in Nebula Graph. Graph modeling depends on the questions that you want to know from the data. Your data drives your graph model. Graph data modeling is intuitive and convenient. Create your data model based on your business model. Test your model and gradually optimize it to fit your business. To get better performance, you can change or redesign your model multiple times.

### Design and evaluate the most important queries

Usually, various types of queries are validated in test scenarios to assess the overall capabilities of the system. However, in most production scenarios, there are not many types of frequently used queries. You can optimize the data model based on key queries selected according to the Pareto (80/20) principle.

### Full-graph scanning avoidance

Graph traversal can be performed after one or more vertices/edges are located through property indexes or VIDs. But for some query patterns, such as subgraph and path query patterns, the source vertex or edge of the traversal cannot be located through property indexes or VIDs. These queries find all the subgraphs that satisfy the query pattern by scanning the whole graph space which will have poor query performance. NebulaGraph does not implement indexing for the graph structures of subgraphs or paths.

### No predefined bonds between Tags and Edge types

Define the bonds between Tags and Edge types in the application, not NebulaGraph. There are no statements that could get the bonds between Tags and Edge types.

### Tags/Edge types predefine a set of properties

While creating Tags or Edge types, you need to define a set of properties. Properties are part of the NebulaGraph Schema.

### Control changes in the business model and the data model

Changes here refer to changes in business models and data models (meta-information), not changes in the data itself.

Some graph databases are designed to be Schema-free, so their data modeling, including the modeling of the graph topology and properties, can be very flexible. Properties can be re-modeled to graph topology, and vice versa. Such systems are often specifically optimized for graph topology access.

NebulaGraph 3.6.0 is a strong-Schema (row storage) system, which means that the business model should not change frequently. For example, the property Schema should not change. It is similar to avoiding ALTER TABLE in MySQL.

On the contrary, vertices and their edges can be added or deleted at low costs. Thus, the easy-to-change part of the business model should be transformed to vertices or edges, rather than properties.

For example, in a business model, people have relatively fixed properties such as age, gender, and name. But their contact, place of visit, trade account, and login device are often changing. The former is suitable for modeling as properties and the latter as vertices or edges.

#### Set temporary properties through self-loop edges

As a strong Schema system, NebulaGraph does not support List-type properties. And using ALTER TAG costs too much. If you need to add some temporary properties or List-type properties to a vertex, you can first create an edge type with the required properties, and then insert one or more edges that direct to the vertex itself. The figure is as follows.



To retrieve temporary properties of vertices, fetch from self-loop edges. For example:

```
//Create the edge type and insert the loop property.
nebula> CREATE EDGE IF NOT EXISTS temp(tmp int);
nebula> INSERT EDGE temp(tmp) VALUES "player100"->"player100"@2:(2);
nebula> INSERT EDGE temp(tmp) VALUES "player100"->"player100"@3:(3);
//After the data is inserted, you can query the loop property by general query statements, for example:
nebula> GO FROM "player100" OVER temp YIELD properties(edge).tmp;
+------+
| properties(EDGE).tmp |
+-----+
| 1 | |
2 | |
3 | |
+-----+
//If you want the results to be returned in the form of a List, you can use a function, for example:
nebula> MATCH (v1:player)-[e:temp]->() return collect(e.tmp);
+------+
| collect(e.tmp) |
+-----+
| [1, 2, 3] |
```

Operations on loops are not encapsulated with any syntactic sugars and you can use them just like those on normal edges.

#### About dangling edges

A dangling edge is an edge that only connects to a single vertex and only one part of the edge connects to the vertex.

In NebulaGraph 3.6.0, dangling edges may appear in the following two cases.

- 1. Insert edges with INSERT EDGE statement before the source vertex or the destination vertex exists.
- 2. Delete vertices with DELETE VERTEX statement and the WITH EDGE option is not used. At this time, the system does not delete the related outgoing and incoming edges of the vertices. There will be dangling edges by default.

Dangling edges may appear in NebulaGraph 3.6.0 as the design allow it to exist. And there is no MERGE statement like openCypher has. The existence of dangling edges depends entirely on the application level. You can use GO and LOOKUP statements to find a dangling edge, but cannot use the MATCH statement to find a dangling edge.

#### Examples:

// Insert an edge that connects two vertices which do not exist in the graph. The source vertex's ID is '11'. The destination vertex's ID is'13'.

nebula> CREATE EDGE IF NOT EXISTS el (name string, age int); nebula> INSERT EDGE el (name, age) VALUES "11"->"13":("n1", 1);

// Query using the GO statement
<pre>nebula&gt; 60 FROM "11" over el YIELD properties(edge); ++   properties(EDGE)   ++ {age: 1, name: "n1"}   ++</pre>
// Query using the <code>`LOOKUP`</code> statement
nebula> LOOKUP ON el YIELD EDGE AS r; ++   r   ++
[:e2 "11"->"13" @0 {age: 1, name: "n1"}]   ++
// Query using the <code>`MATCH`</code> statement
<pre>nebula&gt; MATCH ()-[e:e1]-&gt;() RETURN e; ++</pre>
c   ++ ++
Empty set (time spent 3153/3573 us)

### Breadth-first traversal over depth-first traversal

- NebulaGraph has lower performance for depth-first traversal based on the Graph topology, and better performance for breadth-first traversal and obtaining properties. For example, if model A contains properties "name", "age", and "eye color", it is recommended to create a tag person and add properties name, age, and eye\_color to it. If you create a tag eye\_color and an edge type has, and then create an edge to represent the eye color owned by the person, the traversal performance will not be high.
- The performance of finding an edge by an edge property is close to that of finding a vertex by a vertex property. For some databases, it is recommended to re-model edge properties as those of the intermediate vertices. For example, model the pattern (src)-[edge {P1, P2}]->(dst) as (src)-[edge1]->(i\_node {P1, P2})-[edge2]->(dst). With NebulaGraph 3.6.0, you can use (src)-[edge {P1, P2}]->(dst) directly to decrease the depth of the traversal and increase the performance.

#### Edge directions

To query in the opposite direction of an edge, use the following syntax:

(dst)<-[edge]-(src) Or GO FROM dst REVERSELY.

If you do not care about the directions or want to query against both directions, use the following syntax:

(src)-[edge]-(dst) or GO FROM src BIDIRECT.

Therefore, there is no need to insert the same edge redundantly in the reversed direction.

#### Set tag properties appropriately

Put a group of properties that are on the same level into the same tag. Different groups represent different concepts.

#### Use indexes correctly

Using property indexes helps find VIDs through properties, but can lead to great performance reduction. Only use an index when you need to find vertices or edges through their properties.

### Design VIDs appropriately

See VID.

### Long texts

Do not use long texts to create edge properties. Edge properties are stored twice and long texts lead to greater write amplification. For how edges properties are stored, see Storage architecture. It is recommended to store long texts in HBase or Elasticsearch and store its address in NebulaGraph.

### 13.3.2 Dynamic graphs (sequence graphs) are not supported

In some scenarios, graphs need to have the time information to describe how the structure of the entire graph changes over time. $^1$ 

The Rank field on Edges in NebulaGraph 3.6.0 can be used to store time in int64, but no field on vertices can do this because if you store the time information as property values, it will be covered by new insertion. Thus NebulaGraph does not support sequence graphs.



### 13.3.3 Free graph data modeling tools

#### arrows.app

1. https://blog.twitter.com/engineering/en\_us/topics/insights/2021/temporal-graph-networks ←

# 13.4 System design suggestions

### 13.4.1 QPS or low-latency first

- NebulaGraph 3.6.0 is good at handling small requests with high concurrency. In such scenarios, the whole graph is huge, containing maybe trillions of vertices or edges, but the subgraphs accessed by each request are not large (containing millions of vertices or edges), and the latency of a single request is low. The concurrent number of such requests, i.e., the QPS, can be huge.
- On the other hand, in interactive analysis scenarios, the request concurrency is usually not high, but the subgraphs accessed by each request are large, with thousands of millions of vertices or edges. To lower the latency of big requests in such scenarios, you can split big requests into multiple small requests in the application, and concurrently send them to multiple graphd processes. This can decrease the memory used by each graphd process as well. Besides, you can use NebulaGraph Algorithm for such scenarios.

### 13.4.2 Data transmission and optimization

- Read/write balance. NebulaGraph fits into OLTP scenarios with balanced read/write, i.e., concurrent write and read. It is not suitable for OLAP scenarios that usually need to write once and read many times.
- Select different write methods. For large batches of data writing, use SST files. For small batches of data writing, use INSERT.
- Run COMPACTION and BALANCE jobs to optimize data format and storage distribution at the right time.
- NebulaGraph 3.6.0 does not support transactions and isolation in the relational database and is closer to NoSQL.

### 13.4.3 Query preheating and data preheating

Preheat on the application side:

- The Grapd process does not support pre-compiling queries and generating corresponding query plans, nor can it cache previous query results.
- The Storagd process does not support preheating data. Only the LSM-Tree and BloomFilter of RocksDB are loaded into memory at startup.
- Once accessed, vertices and edges are cached respectively in two types of LRU cache of the Storage Service.

# 13.5 Execution plan

NebulaGraph 3.6.0 applies rule-based execution plans. Users cannot change execution plans, pre-compile queries (and corresponding plan cache), or accelerate queries by specifying indexes.

To view the execution plan and executive summary, see  $\ensuremath{\mathsf{EXPLAIN}}$  and  $\ensuremath{\mathsf{PROFILE}}.$ 

## 13.6 Processing super vertices

### 13.6.1 Principle introduction

In graph theory, a super vertex, also known as a dense vertex, is a vertex with an extremely high number of adjacent edges. The edges can be outgoing or incoming.

Super vertices are very common because of the power-law distribution. For example, popular leaders in social networks (Internet celebrities), top stocks in the stock market, Big Four in the banking system, hubs in transportation networks, websites with high clicking rates on the Internet, and best sellers in E-commerce.

In NebulaGraph 3.6.0, a vertex and its properties form a key-value pair, with its VID and other meta information as the key. Its Out-Edge Key-Value and In-Edge Key-Value are stored in the same partition in the form of LSM-trees in hard disks and caches.

Therefore, directed traversals from this vertex and directed traversals ending at this vertex both involve either a large number of sequential IO scans (ideally, after Compaction or a large number of random IO (frequent writes to the vertex and its ingoing and outgoing edges).

As a rule of thumb, a vertex is considered dense when the number of its edges exceeds 10,000. Some special cases require additional consideration.

#### Q Note

In NebulaGraph 3.6.0, there is not any data structure to store the out/in degree for each vertex. Therefore, there is no direct method to know whether it is a super vertex or not. You can try to use Spark to count the degrees periodically.

### Indexes for duplicate properties

In a property graph, there is another class of cases similar to super vertices: **a property has a very high duplication rate**, i.e., many vertices with the same tag but different VIDs have identical property and property values.

Property indexes in NebulaGraph 3.6.0 are designed to reuse the functionality of RocksDB in the Storage Service, in which case indexes are modeled as keys with the same prefix. If the lookup of a property fails to hit the cache, it is processed as a random seek and a sequential prefix scan on the hard disk to find the corresponding VID. After that, the graph is usually traversed from this vertex, so that another random read and sequential scan for the corresponding key-value of this vertex will be triggered. The higher the duplication rate, the larger the scan range.

For more information about property indexes, see How indexing works in NebulaGraph.

Usually, special design and processing are required when the number of duplicate property values exceeds 10,000.

### Suggested solutions

SOLUTIONS AT THE DATABASE END

1. Truncation: Only return a certain number (a threshold) of edges, and do not return other edges exceeding this threshold.

2. Compact: Reorganize the order of data in RocksDB to reduce random reads and increase sequential reads.

#### SOLUTIONS AT THE APPLICATION END

Break up some of the super vertices according to their business significance:

• Delete multiple edges and merge them into one.

For example, in the transfer scenario  $(Account_A)-[TRANSFER]->(Account_B)$ , each transfer record is modeled as an edge between account A and account B, then there may be tens of thousands of transfer records between  $(Account_A)$  and  $(Account_B)$ .

In such scenarios, merge obsolete transfer details on a daily, weekly, or monthly basis. That is, batch-delete old edges and replace them with a small number of edges representing monthly total and times. And keep the transfer details of the latest month.

• Split an edge into multiple edges of different types.

For example, in the (Airport)<-[DEPART]-(Flight) scenario, the departure of each flight is modeled as an edge between a flight and an airport. Departures from a big airport might be enormous.

According to different airlines, divide the DEPART edge type into finer edge types, such as DEPART\_CEAIR, DEPART\_CSAIR, etc. Specify the departing airline in queries (graph traversal).

• Split vertices.

For example, in the loan network (person)-[BORROW]->(bank), large bank A will have a very large number of loans and borrowers. In such scenarios, you can split the large vertex A into connected sub-vertices A1, A2, and A3.

(Person1)-[BORROW]->(BankA1), (Person2)-[BORROW]->(BankA2), (Person2)-[BORROW]->(BankA3); (BankA1)-[BELONGS\_T0]->(BankA), (BankA2)-[BELONGS\_T0]->(BankA3), (BankA3)-[BELONGS\_T0]->(BankA)

A1, A2, and A3 can either be three real branches of bank A, such as Beijing branch, Shanghai branch, and Zhejiang branch, or three virtual branches set up according to certain rules, such as A1: 1-1000, A2: 1001-10000 and A3: 10000+ according to the number of loans. In this way, any operation on A is converted into three separate operations on A1, A2, and A3.

Last update: November 3, 2023

### 13.7 Enable AutoFDO for NebulaGraph

The AutoFDO can analyze the performance of an optimized program and use the program's performance information to guide the compiler to re-optimize the program. This document will help you to enable the AutoFDO for NebulaGraph.

More information about the AutoFDO, please refer AutoFDO Wiki.

### 13.7.1 Resource Preparations

### Install Dependencies

• Install perf

```
sudo apt-get update
sudo apt-get install -y linux-tools-common \
linux-tools-generic \
linux-tools-`uname -r`
```

### Install autofdo tool

```
sudo apt-get update
sudo apt-get install -y autofdo
```

Or you can compile the *autofdo tool* from source.

### NebulaGraph Binary with Debug Version

For how to build NebulaGraph from source, please refer to the official document: Install NebulaGraph by compiling the source code. In the configure step, replace CMAKE\_BUILD\_TYPE=Release with CMAKE\_BUILD\_TYPE=RelWithDebInfo as below:

\$ cmake -DCMAKE\_INSTALL\_PREFIX=/usr/local/nebula -DENABLE\_TESTING=OFF -DCMAKE\_BUILD\_TYPE=RelWithDebInfo ...

### 13.7.2 Prepare Test Data

In our test environment, we use NebulaGraph Bench to prepare the test data and collect the profile data by running the *FindShortestPath*, *Go1Step*, *Go2Step*, *Go3Step*, *InsertPersonScenario* 5 scenarios.

### Note

You can use your *TopN* queries in your production environment to collect the profile data, the performance can gain more in your environment.

### 13.7.3 Prepare Profile Data

### Collect Perf Data For AutoFdo Tool

1. After the test data preparation work done. Collect the perf data for different scenarios. Get the pid of storaged, graphd, metad.

```
$ nebula.service status all
[INF0] nebula-metad: Running as 305422, Listening on 9559
[INF0] nebula-graphd: Running as 305516, Listening on 9669
[INF0] nebula-storaged: Running as 305707, Listening on 9779
```

2. Start the *perf record* for *nebula-graphd* and *nebula-storaged*.

perf record -p 305516,305707 -b -e br\_inst\_retired.near\_taken:pp -o ~/FindShortestPath.data

#### Q Note

Because the nebula-metad service contribution percent is small compared with nebula-graphd and nebula-storaged services. To reduce effort, we didn't collect the perf data for nebula-metad service.

3. Start the benchmark test for *FindShortestPath* scenario.

cd NebulaGraph-Bench python3 run.py stress run -s benchmark -scenario find\_path.FindShortestPath -a localhost:9669 --args='-u 100 -i 100000'

- 4. After the benchmark finished, end the *perf record* by *Ctrl* + *c*.
- 5. Repeat above steps to collect corresponding profile data for the rest *Go1Step*, *Go2Step*, *Go3Step* and *InsertPersonScenario* scenarios.

### **Create Gcov File**

```
create_gcov --binary=$NEBULA_HOME/bin/nebula-storaged \
--profile=~/FindShortestPath.data \
--gcov=~/FindShortestPath-storaged.gcov \
-gcov_version=1
create_gcov --binary=$NEBULA_HOME/bin/nebula-graphd \
--profile=~/FindShortestPath.data \
--gcov=version=1
```

### Repeat for Go1Step, Go2Step, Go3Step and InsertPersonScenario scenarios.

### Merge the Profile Data

```
profile_merger ~/FindShortestPath-graphd.gcov \
~/FindShortestPath-storaged.gcov \
~/golstep-storaged.gcov \
~/golstep-storaged.gcov \
~/golstep-graphd.gcov \
~/golstep-storaged.gcov \
~/golstep-storaged.gcov \
~/golstep-storaged.gcov \
~/golstep-storaged.gcov \
~/golstep-storaged.gcov \
~/golstep-storaged.gcov \
~/InsertPersonScenario-storaged.gcov \
~/InsertPersonScenario-graphd.gcov
```

You will get a merged profile which is named fbdata.afdo after that.

### 13.7.4 Recompile GraphNebula Binary with the Merged Profile

Recompile the GraphNebula Binary by passing the profile with compile option -fauto-profile.

```
diff --git a/cmake/nebula/GeneralCompilerConfig.cmake b/cmake/nebula/GeneralCompilerConfig.cmake
@@ -20,6 +20,8 @@ add_compile_options(-Wshadow)
add_compile_options(-Mon-virtual-dtor)
add_compile_options(-Wignored-qualifiers)
+add_compile_options(-Fauto-profile=~/fbdata.afdo)
```

#### Q Note

When you use multiple fbdata.afdo to compile multiple times, please remember to make clean before re-compile, baucase only change the fbdata.afdo will not trigger re-compile.

### 13.7.5 Performance Test Result

### Hardware & Software Environment

Key	Value
CPU Processor#	2
Sockets	2
NUMA	2
СРИ Туре	Intel(R) Xeon(R) Platinum 8380 CPU @ 2.30GHz
Cores per Processor	40C80T
Cache	L1 data: 48KB L1 i: 32KB L2: 1.25MB per physical core L3: shared 60MB per processor
Memory	Micron DDR4 3200MT/s 16GB16Micron DDR4 3200MT/s 16GB16
SSD Disk	INTEL SSDPE2KE016T8
SSD R/W Sequential	3200 MB/s (read) / 2100 MB/s(write)
Nebula Version	master with commit id 51d84a4ed7d2a032a337e3b996c927e3bc5d1415
Kernel	4.18.0-408.el8.x86_64

Test Results

Scenario	Average Latency(LiB)	Default Binary	Optimized Binary with AutoFDO	P95 Latency (LiB)	Default Binary	Optimized Binary with AutoFDO
FindShortestPath	1	8072.52	7260.10	1	22102.00	19108.00
	2	8034.32	7218.59	2	22060.85	19006.00
	3	8079.27	7257.24	3	22147.00	19053.00
	4	8087.66	7221.39	4	22143.00	19050.00
	5	8044.77	7239.85	5	22181.00	19055.00
	STDDEVP	20.57	17.34	STDDEVP	41.41	32.36
	Mean	8063.71	7239.43	Mean	22126.77	19054.40
	STDDEVP/ Mean	0.26%	0.24%	STDDEVP/ Mean	0.19%	0.17%
	Opt/Default	100.00%	10.22%	Opt/ Default	100.00%	13.89%
Go1Step	1	422.53	418.37	1	838.00	850.00
	2	432.37	402.44	2	866.00	815.00
	3	437.45	407.98	3	874.00	836.00
	4	429.16	408.38	4	858.00	838.00
	5	446.38	411.32	5	901.00	837.00
	STDDEVP	8.02	5.20	STDDEVP	20.63	11.30
	Mean	433.58	409.70	Mean	867.40	835.20
	STDDEVP/ Mean	1.85%	1.27%	STDDEVP/ Mean	2.38%	1.35%
	Opt/Default	100.00%	5.51%	Opt/ Default	100.00%	3.71%
Go2Step	1	2989.93	2824.29	1	10202.00	9656.95
	2	2957.22	2834.55	2	10129.00	9632.40
	3	2962.74	2818.62	3	10168.40	9624.70
	4	2992.39	2817.27	4	10285.10	9647.50
	5	2934.85	2834.91	5	10025.00	9699.65
	STDDEVP	21.53	7.57	STDDEVP	85.62	26.25
	Mean	2967.43	2825.93	Mean	10161.90	9652.24
	STDDEVP/ Mean	0.73%	0.27%	STDDEVP/ Mean	0.84%	0.27%
	Opt/Default	100.00%	4.77%	Opt/ Default	100.00%	5.02%
Go3Step	1	93551.97	89406.96	1	371359.55	345433.50
	2	92418.43	89977.25	2	368868.00	352375.20
	3	92587.67	90339.25	3	365390.15	356198.55

Scenario	Average Latency(LiB)	Default Binary	Optimized Binary with AutoFDO	P95 Latency (LiB)	Default Binary	Optimized Binary with AutoFDO
	4	93371.64	92458.95	4	373578.15	365177.75
	5	94046.05	89943.44	5	373392.25	352576.00
	STDDEVP	609.07	1059.54	STDDEVP	3077.38	6437.52
	Mean	93195.15	90425.17	Mean	370517.62	354352.20
	STDDEVP/ Mean	0.65%	1.17%	STDDEVP/ Mean	0.83%	1.82%
	Opt/Default	100.00%	2.97%	Opt/ Default	100.00%	4.36%
InsertPerson	1	2022.86	1937.36	1	2689.00	2633.45
	2	1966.05	1935.41	2	2620.45	2555.00
	3	1985.25	1953.58	3	2546.00	2593.00
	4	2026.73	1887.28	4	2564.00	2394.00
	5	2007.55	1964.41	5	2676.00	2581.00
	STDDEVP	23.02	26.42	STDDEVP	57.45	82.62
	Mean	2001.69	1935.61	Mean	2619.09	2551.29
	STDDEVP/ Mean	1.15%	1.37%	STDDEVP/ Mean	2.19%	3.24%
	Opt/Default	100.00%	3.30%	Opt/ Default	100.00%	2.59%

### 13.8 Best practices

NebulaGraph is used in a variety of industries. This topic presents a few best practices for using NebulaGraph. For more best practices, see Blog.

### 13.8.1 Scenarios

- Use cases
- User review
- Performance

### 13.8.2 Kernel

- What is a graph database and what are its use cases Definition, examples & trends
- NebulaGraph Source Code Explained: Variable-Length Pattern Matching
- Adding a Test Case for NebulaGraph
- BDD-Based Integration Testing Framework for NebulaGraph: Part I
- BDD-Based Integration Testing Framework for NebulaGraph: Part II
- Understanding Subgraph in NebulaGraph
- Full-Text Indexing in NebulaGraph

### 13.8.3 Ecosystem tool

- Validating Import Performance of NebulaGraph Importer
- Ecosystem Tools: NebulaGraph Dashboard for Monitoring
- Visualizing Graph Data with NebulaGraph Explorer

# 14. Clients

### 14.1 Clients overview

NebulaGraph supports multiple types of clients for users to connect to and manage the NebulaGraph database.

- NebulaGraph Console: the native CLI client
- NebulaGraph CPP: the NebulaGraph client for C++
- NebulaGraph Java: the NebulaGraph client for Java
- NebulaGraph Python: the NebulaGraph client for Python
- NebulaGraph Go: the NebulaGraph client for Golang

#### Q Note

Only the following classes are thread-safe:

- NebulaPool and SessionPool in NebulaGraph Java
- ConnectionPool and SessionPool in NebulaGraph Go

Last update: April 15, 2024

# 14.2 NebulaGraph Console

NebulaGraph Console is a native CLI client for NebulaGraph. It can be used to connect a NebulaGraph cluster and execute queries. It also supports special commands to manage parameters, export query results, import test datasets, etc.

### 14.2.1 Compatibility with NebulaGraph

See github.

### 14.2.2 Obtain NebulaGraph Console

You can obtain NebulaGraph Console in the following ways:

- Download the binary file from the GitHub releases page.
- Compile the source code to obtain the binary file. For more information, see Install from source code.

### 14.2.3 NebulaGraph Console functions

### Connect to NebulaGraph

To connect to NebulaGraph with the nebula-console file, use the following syntax:

<path\_of\_console> -addr <ip> -port <port> -u <username> -p <password>

- path\_of\_console indicates the storage path of the NebulaGraph Console binary file.
- When two-way authentication is required after SSL encryption is enabled, you need to specify SSL-related parameters when connecting.

### For example:

• Direct link to NebulaGraph

./nebula-console -addr 192.168.8.100 -port 9669 -u root -p nebula

• Enable SSL encryption and require two-way authentication

./nebula-console -addr 192.168.8.100 -port 9669 -u root -p nebula -enable\_ssl -ssl\_root\_ca\_path /home/xxx/cert/root.crt -ssl\_cert\_path /home/xxx/cert/client.crt -ssl\_private\_key\_path / home/xxx/cert/client.key

### Parameter descriptions are as follows:

Parameter	Description
-h/-help	Shows the help menu.
-addr/-address	Sets the IP or hostname of the Graph service. The default address is 127.0.0.1.
-P/-port	Sets the port number of the graphd service. The default port number is 9669.
-u/-user	Sets the username of your NebulaGraph account. Before enabling authentication, you can use any existing username. The default username is $root$ .
-p/-password	Sets the password of your NebulaGraph account. Before enabling authentication, you can use any characters as the password.
-t/-timeout	Sets an integer-type timeout threshold of the connection. The unit is millisecond. The default value is 120.
-e/-eval	Sets a string-type nGQL statement. The nGQL statement is executed once the connection succeeds. The connection stops after the result is returned.
-f/-file	Sets the path of an nGQL file. The nGQL statements in the file are executed once the connection succeeds. The result will be returned and the connection stops then.
-enable_ssl	Enables SSL encryption when connecting to NebulaGraph.
-ssl_root_ca_path	Sets the storage path of the certification authority file.
-ssl_cert_path	Sets the storage path of the certificate file.
-ssl_private_key_path	Sets the storage path of the private key file.
- ssl_insecure_skip_verify	Specifies whether the client skips verifying the server's certificate chain and hostname. The default is false. If set to true, any certificate chain and hostname provided by the server is accepted.

For information on more parameters, see the project repository.

### Manage parameters

You can save parameters for parameterized queries.

#### Q Note

• Setting a parameter as a VID in a query is not supported.

- Parameters are not supported in SAMPLE clauses.
- Parameters are deleted when their sessions are released.

• The command to save a parameter is as follows:

nebula> :param <param\_name> => <param\_value>;

The example is as follows:

nebula> :param p1 ⇒ "Tim Duncan"; nebula> MATCH (v:player{name:\$p1})-[:follow]->(n)   RETURN v,n;				
+	.++			
V V	n			
+	.++			
<pre>("player100" :player{age: 42, name: "Tim Duncan"})</pre>	("player125" :player{age: 41, name: "Manu Ginobili"})			
("player100" :player{age: 42, name: "Tim Duncan"})	("player101" :player{age: 36, name: "Tony Parker"})			
+	.++			
nebula> :param p2 ⇒ {"a":3,"b":false,"c":"Tim Duncan"}; nebula> RETURN \$p2.b AS b:				
++				
b				
++				
false				
++				

• The command to view the saved parameters is as follows:

nebula> :params;

• The command to view the specified parameters is as follows:

nebula> :params <param\_name>;

• The command to delete a specified parameter is as follows:

nebula> :param <param\_name> =>;

### Export query results

Export query results, which can be saved as a CSV file, DOT file, and a format of Profile or Explain.

#### Q Note

• The exported file is stored in the working directory, i.e., what the linux command pwd shows.

• This command only works for the next query statement.

• You can copy the contents of the DOT file and paste them in GraphvizOnline to generate a visualized execution plan.

### • The command to export a csv file is as follows:

nebula> :CSV <file\_name.csv>;

#### • The command to export a DOT file is as follows:

nebula> :dot <file\_name.dot>

The example is as follows:

nebula> :dot a.dot
nebula> PROFILE FORMAT="dot" GO FROM "player100" OVER follow;

• The command to export a PROFILE or EXPLAIN format is as follows:

nebula> :profile <file\_name>;

or

nebula> :explain <file\_name>;

### Note

The text file output by the above command is the preferred way to report issues in GitHub and execution plans in forums, and for graph query tuning because it has more information and is more readable than a screenshot or CSV file in Studio.

### The example is as follows:

```
nebula> :profile profile.log
nebula> PROFILE GO FROM "player102" OVER serve YIELD dst(edge);
nebula> :profile profile.dot
nebula> PROFILE FORMAT="dot" GO FROM "player102" OVER serve YIELD dst(edge);
nebula> :explain explain.log
nebula> EXPLAIN GO FROM "player102" OVER serve YIELD dst(edge);
```

#### Import a testing dataset

The testing dataset is named basketballplayer. To view details about the schema and data, use the corresponding SHOW command.

The command to import a testing dataset is as follows:

nebula> :play basketballplayer

#### Run a command multiple times

To run a command multiple times, use the following command:

nebula> :repeat N

The example is as follows:

```
nebula> :repeat 3
nebula> G0 FROM "player100" OVER follow YIELD dst(edge);
| dst(EDGE)
| "player101"
 "player125"
Got 2 rows (time spent 2602/3214 us)
Fri, 20 Aug 2021 06:36:05 UTC
+--
           ---+
| dst(EDGE) |
| "player101"
 "player125"
Got 2 rows (time spent 583/849 us)
Fri, 20 Aug 2021 06:36:05 UTC
dst(EDGE)
| "player101"
| "player125"
Got 2 rows (time spent 496/671 us)
Fri, 20 Aug 2021 06:36:05 UTC
Executed 3 times, (total time spent 3681/4734 us), (average time spent 1227/1578 us)
```

### Sleep

This command will make NebulaGraph Console sleep for N seconds. The schema is altered in an async way and takes effect in the next heartbeat cycle. Therefore, this command is usually used when altering schema. The command is as follows:

nebula> :sleep N

### Disconnect NebulaGraph Console from NebulaGraph

You can use :EXIT or :QUIT to disconnect from NebulaGraph. For convenience, NebulaGraph Console supports using these commands in lower case without the colon (":"), such as quit.

The example is as follows:

nebula> :QUIT Bye root!

Last update: November 22, 2023

### 14.3 NebulaGraph CPP

NebulaGraph CPP is a C++ client for connecting to and managing the NebulaGraph database.

### 14.3.1 Prerequisites

You have installed C++ and GCC 4.8 or later versions.

### 14.3.2 Compatibility with NebulaGraph

See github.

### 14.3.3 Install NebulaGraph CPP

This document describes how to install NebulaGraph CPP with the source code.

### Prerequisites

- You have prepared the correct resources.
- You have installed C++ and GCC version is: {10.1.0 | 9.3.0 | 9.2.0 | 9.1.0 | 8.3.0 | 7.5.0 | 7.1.0}. For details, see the gcc\_preset\_versions parameter.

### Steps

- 1. Clone the NebulaGraph CPP source code to the host.
- (Recommended) To install a specific version of NebulaGraph CPP, use the Git option --branch to specify the branch. For example, to install v3.4.0, run the following command:

\$ git clone --branch release-3.4 https://github.com/vesoft-inc/nebula-cpp.git

• To install the daily development version, run the following command to download the source code from the master branch:

\$ git clone https://github.com/vesoft-inc/nebula-cpp.git

2. Change the working directory to nebula-cpp.

\$ cd nebula-cpp

3. Create a directory named build and change the working directory to it.

\$ mkdir build && cd build

4. Generate the makefile file with CMake.

#### O Note

The default installation path is /usr/local/nebula. To modify it, add the -DCMAKE\_INSTALL\_PREFIX=<installation\_path> option while running the following command.

\$ cmake -DCMAKE\_BUILD\_TYPE=Release ...

#### Q Note

If G++ does not support C++ 11, add the option -DDISABLE\_CXX11\_ABI=ON .

### 5. Compile NebulaGraph CPP.

To speed up the compiling, use the -j option to set a concurrent number N. It should be  $(\min(\text{CPU core number}), \frac{(\text{CPU core number})}{(2)})$ .

\$ make -j{N}

### 6. Install NebulaGraph CPP.

\$ sudo make install

7. Update the dynamic link library.

\$ sudo ldconfig

### 14.3.4 Use NebulaGraph CPP

Compile the CPP file to an executable file, then you can use it. The following steps take using SessionExample.cpp for example.

- 1. Use the example code to create the SessionExample.cpp file.
- 2. Run the following command to compile the file.

\$ LIBRARY\_PATH=<library\_folder\_path>:\$LIBRARY\_PATH g++ -std=c++11 SessionExample.cpp -I<include\_folder\_path> -lnebula\_graph\_client -o session\_example

- library\_folder\_path : The storage path of the NebulaGraph dynamic libraries. The default path is /usr/local/nebula/lib64 .
- include\_folder\_path : The storage of the NebulaGraph header files. The default path is /usr/local/nebula/include .

### For example:

\$ LIBRARY\_PATH=/usr/local/nebula/lib64:\$LIBRARY\_PATH g++ -std=c++11 SessionExample.cpp -I/usr/local/nebula/include -lnebula\_graph\_client -o session\_example

### 14.3.5 API reference

Click here to check the classes and functions provided by the CPP Client.

### 14.3.6 Core of the example code

Nebula CPP clients provide both Session Pool and Connection Pool methods to connect to NebulaGraph. Using the Connection Pool method requires users to manage session instances by themselves.

Session Pool

For more details about all the code, see SessionPoolExample.

• Connection Pool

For more details about all the code, see SessionExample.

Last update: January 22, 2024
## 14.4 NebulaGraph Java

NebulaGraph Java is a Java client for connecting to and managing the NebulaGraph database.

#### 14.4.1 Prerequisites

You have installed Java 8.0 or later versions.

#### 14.4.2 Compatibility with NebulaGraph

See github.

#### 14.4.3 Download NebulaGraph Java

• (Recommended) To install a specific version of NebulaGraph Java, use the Git option --branch to specify the branch. For example, to install v3.6.1, run the following command:

\$ git clone --branch release-3.6 https://github.com/vesoft-inc/nebula-java.git

• To install the daily development version, run the following command to download the source code from the master branch:

\$ git clone https://github.com/vesoft-inc/nebula-java.git

## 14.4.4 Use NebulaGraph Java

#### Q Note

We recommend that each thread use one session. If multiple threads use the same session, the performance will be reduced.

When importing a Maven project with tools such as IDEA, set the following dependency in pom.xml.

#### Q Note

3.0.0-SNAPSHOT indicates the daily development version that may have unknown issues. We recommend that you replace 3.0.0-SNAPSHOT with a released version number to use a table version.

```
<dependency>
  <groupId>com.vesoft</groupId>
  <artifactId>client</artifactId>
  <version>3.0.0-SNAPSHOT</version>
</dependency>
```

If you cannot download the dependency for the daily development version, set the following content in pom.xml. Released versions have no such issue.

```
<repositories
<repositorys
<id>snapshots</id>
<url>https://oss.sonatype.org/content/repositories/snapshots/</url>
</repositorys
</repositories>
```

If there is no Maven to manage the project, manually download the JAR file to install NebulaGraph Java.

## 14.4.5 API reference

Click here to check the classes and functions provided by the Java Client.

## 14.4.6 Core of the example code

The NebulaGraph Java client provides both Connection Pool and Session Pool modes, using Connection Pool requires the user to manage session instances.

• Session Pool

For all the code, see GraphSessionPoolExample.

• Connection Pool

For all the code, see GraphClientExample.

Last update: January 9, 2024

# 14.5 NebulaGraph Python

NebulaGraph Python is a Python client for connecting to and managing the NebulaGraph database.

## 14.5.1 Prerequisites

You have installed Python 3.6 or later versions.

## 14.5.2 Compatibility with NebulaGraph

See github.

### 14.5.3 Install NebulaGraph Python

#### Install NebulaGraph Python with pip

\$ pip install nebula3-python==<version>

#### Install NebulaGraph Python from the source code

1. Clone the NebulaGraph Python source code to the host.

• (Recommended) To install a specific version of NebulaGraph Python, use the Git option --branch to specify the branch. For example, to install v3.4.0, run the following command:

\$ git clone --branch release-3.4 https://github.com/vesoft-inc/nebula-python.git

• To install the daily development version, run the following command to download the source code from the master branch:

\$ git clone https://github.com/vesoft-inc/nebula-python.git

2. Change the working directory to nebula-python.

\$ cd nebula-python

3. Run the following command to install NebulaGraph Python.

\$ pip install .

### 14.5.4 API reference

Click here to check the classes and functions provided by the Python Client.

### 14.5.5 Core of the example code

NebulaGraph Python clients provides Connection Pool and Session Pool methods to connect to NebulaGraph. Using the Connection Pool method requires users to manage sessions by themselves.

• Session Pool

For details about all the code, see SessinPoolExample.py.

For limitations of using the Session Pool method, see Example of using session pool.

Connection Pool

For details about all the code, see Example.

Last update: January 9, 2024

## 14.6 NebulaGraph Go

NebulaGraph Go is a Golang client for connecting to and managing the NebulaGraph database.

#### 14.6.1 Prerequisites

You have installed Golang 1.13 or later versions.

#### 14.6.2 Compatibility with NebulaGraph

See github.

#### 14.6.3 Download NebulaGraph Go

• (Recommended) To install a specific version of NebulaGraph Go, use the Git option --branch to specify the branch. For example, to install v3.7.0, run the following command:

\$ git clone --branch release-3.7 https://github.com/vesoft-inc/nebula-go.git

• To install the daily development version, run the following command to download the source code from the master branch:

\$ git clone https://github.com/vesoft-inc/nebula-go.git

## 14.6.4 Install or update

Run the following command to install or update NebulaGraph Go:

\$ go get -u -v github.com/vesoft-inc/nebula-go/v3@v3.7.0

## 14.6.5 API reference

Click here to check the functions and types provided by the GO Client.

## 14.6.6 Core of the example code

The NebulaGraph GO client provides both Connection Pool and Session Pool, using Connection Pool requires the user to manage the session instances.

• Session Pool

For details about all the code, see <a href="mailto:session\_pool\_example.go">session\_pool\_example.go</a>.

For limitations of using Session Pool, see Usage example.

• Connection Pool

For all the code, see graph\_client\_basic\_example and graph\_client\_goroutines\_example.

Last update: February 1, 2024

# 14.7 Community contributed clients

You can use the following clients developed by community users to connect to and manage NebulaGraph:

- NebulaGraph Rust
- NebulaGraph PHP
- NebulaGraph Node
- NebulaGraph .NET

Last update: April 15, 2024

# 15. Studio

## 15.1 About NebulaGraph Studio

## 15.1.1 What is NebulaGraph Studio

NebulaGraph Studio (Studio in short) is a browser-based visualization tool to manage NebulaGraph. It provides you with a graphical user interface to manipulate graph schemas, import data, and run nGQL statements to retrieve data. With Studio, you can quickly become a graph exploration expert from scratch. You can view the latest source code in the NebulaGraph GitHub repository, see nebula-studio for details.

# Note

You can also try some functions online in Studio.

#### Deployment

In addition to deploying Studio with RPM-based, DEB-based, or Tar-based packages, or with Docker, you can also deploy Studio with Helm in the Kubernetes cluster. For more information, see Deploy Studio.

The functions of the above four deployment methods are the same and may be restricted when using Studio. For more information, see Limitations.

#### Features

Studio can easily manage NebulaGraph data, with the following functions:

- On the **Schema** page, you can use the graphical user interface to create the space, Tag, Edge Type, Index, and view the statistics on the graph. It helps you quickly get started with NebulaGraph.
- On the Import page, you can operate batch import of vertex and edge data with clicks, and view a real-time import log.
- On the **Console** page, you can run nGQL statements and read the results in a human-friendly way.

## Scenarios

You can use Studio in one of these scenarios:

- You have a dataset, and you want to explore and analyze data in a visualized way. You can use Docker Compose to deploy NebulaGraph and then use Studio to explore or analyze data in a visualized way.
- You are a beginner of nGQL (NebulaGraph Query Language) and you prefer to use a GUI rather than a command-line interface (CLI) to learn the language.

#### Authentication

Authentication is not enabled in NebulaGraph by default. Users can log into Studio with the root account and any password.

When NebulaGraph enables authentication, users can only sign into Studio with the specified account. For more information, see Authentication.

### Version compatibility

#### Q Note

The Studio version is released independently of the NebulaGraph core. The correspondence between the versions of Studio and the NebulaGraph core, as shown in the table below.

NebulaGraph version	Studio version
3.6.0	3.8.0, 3.7.0
3.5.0	3.7.0
3.4.0 ~ 3.4.1	3.7.0 \ 3.6.0 \ 3.5.1 \ 3.5.0
3.3.0	3.5.1 \ 3.5.0
$3.0.0 \sim 3.2.0$	3.4.1 \ 3.4.0
3.1.0	3.3.2
3.0.0	3.2.x
2.6.x	3.1.x
2.6.x	3.1.x
2.0 & 2.0.1	2.x
1.x	1.x

## Check updates

Studio is in development. Users can view the latest releases features through Changelog.

To view the Changelog, on the upper-right corner of the page, click the version and then **New version**.



Last update: November 3, 2023

## 15.1.2 Limitations

This topic introduces the limitations of Studio.

#### Architecture

For now, Studio v3.x supports x86\_64 architecture only.

#### Upload data

Only CSV files without headers can be uploaded, but no limitations are applied to the size and store period for a single file. The maximum data volume depends on the storage capacity of your machine.

## Data backup

For now, only supports exporting query results in CSV format on **Console**, and other data backup methods are not supported.

#### nGQL statements

On the **Console** page of Docker-based and RPM-based Studio v3.x, all the nGQL syntaxes except these are supported:

- USE <space\_name>: You cannot run such a statement on the **Console** page to choose a graph space. As an alternative, you can click a graph space name in the drop-down list of **Current Graph Space**.
- You cannot use line breaks (\). As an alternative, you can use the Enter key to split a line.

#### Browser

We recommend that you use the latest version of Chrome to get access to Studio. Otherwise, some features may not work properly.

Last update: October 25, 2023

# 15.2 Deploy and connect

## 15.2.1 Deploy Studio

This topic describes how to deploy Studio locally by RPM, DEB, tar package and Docker.

#### **RPM-based Studio**

PREREQUISITES

Before you deploy RPM-based Studio, you must confirm that:

- The NebulaGraph services are deployed and started. For more information, see NebulaGraph Database Manual.
- The Linux distribution is CentOS, install Lsof.
- Before the installation starts, the following ports are not occupied.

Port	Description
7001	Web service provided by Studio.

INSTALL

1. Select and download the RPM package according to your needs. It is recommended to select the latest version. Common links are as follows:

Installation package	Checksum	NebulaGraph version
nebula-graph-studio-3.8.0.x86_64.rpm	nebula-graph-studio-3.8.0.x86_64.rpm.sha256	3.6.0

2. Use sudo rpm -i <rpm\_name> to install RPM package.

For example, install Studio 3.8.0, use the following command. The default installation path is /usr/local/nebula-graph-studio.

\$ sudo rpm -i nebula-graph-studio-3.8.0.x86\_64.rpm

You can also install it to the specified path using the following command:

\$ sudo rpm -i nebula-graph-studio-3.8.0.x86\_64.rpm --prefix=<path>

When the screen returns the following message, it means that the PRM-based Studio has been successfully started.

Start installing NebulaGraph Studio now... NebulaGraph Studio has been installed. NebulaGraph Studio started automatically.

3. When Studio is started, use <a href="http://<ip-address>:7001">http://<ip-address>:7001</a> to get access to Studio.

If you can see the **Config Server** page on the browser, Studio is started successfully.



#### UNINSTALL

You can uninstall Studio using the following command:

\$ sudo rpm -e nebula-graph-studio-3.8.0.x86\_64

If these lines are returned, PRM-based Studio has been uninstalled.

NebulaGraph Studio removed, bye~

#### EXCEPTION HANDLING

If the automatic start fails during the installation process or you want to manually start or stop the service, use the following command:

• Start the service manually

\$ bash /usr/local/nebula-graph-studio/scripts/rpm/start.sh

• Stop the service manually

```
$ bash /usr/local/nebula-graph-studio/scripts/rpm/stop.sh
```

If you encounter an error bind EADDRINUSE 0.0.0.0:7001 when starting the service, you can use the following command to check port 7001 usage.

\$ lsof -i:7001

If the port is occupied and the process on that port cannot be terminated, you can modify the startup port within the studio configuration and restart the service.

```
//Modify the studio service configuration. The default path to the configuration file is `/usr/local/nebula-graph-studio`.
$ vi etc/studio-api.yam
//Modify this port number and change it to any
Port: 7001
//Restart service
$ systemctl restart nebula-graph-studio.service
```

#### **DEB-based Studio**

```
PREREQUISITES
```

Before you deploy DEB-based Studio, you must do a check of these:

- The NebulaGraph services are deployed and started. For more information, see NebulaGraph Database Manual.
- The Linux distribution is Ubuntu.
- Before the installation starts, the following ports are not occupied.

Port	Description
7001	Web service provided by Studio

• The path /usr/lib/systemd/system exists in the system. If not, create it manually.

INSTALL

1. Select and download the DEB package according to your needs. It is recommended to select the latest version. Common links are as follows:

Installation package	Checksum	NebulaGraph version
nebula-graph-studio-3.8.0.x86_64.deb	nebula-graph-studio-3.8.0.x86_64.deb.sha256	3.6.0
	,	

2. Use sudo dpkg -i <deb\_name> to install DEB package.

For example, install Studio 3.8.0, use the following command:

\$ sudo dpkg -i nebula-graph-studio-3.8.0.x86\_64.deb

3. When Studio is started, use <a href="http://sip address>:7001">http://sip address>:7001</a> to get access to Studio.

If you can see the **Config Server** page on the browser, Studio is started successfully.



UNINSTALL

You can uninstall Studio using the following command:

\$ sudo dpkg -r nebula-graph-studio

## tar-based Studio

PREREQUISITES

Before you deploy tar-based Studio, you must do a check of these:

- The NebulaGraph services are deployed and started. For more information, see NebulaGraph Database Manual.
- Before the installation starts, the following ports are not occupied.

Port	Description
7001	Web service provided by Studio

INSTALL AND DEPLOY

1. Select and download the tar package according to your needs. It is recommended to select the latest version. Common links are as follows:

Installation package	Studio version
nebula-graph-studio-3.8.0.x86_64.tar.gz	3.8.0

2. Use tar -xvf to decompress the tar package.

\$ tar -xvf nebula-graph-studio-3.8.0.x86\_64.tar.gz

#### 3. Deploy and start nebula-graph-studio.

\$ cd nebula-graph-studio
\$ ./server

4. When Studio is started, use <a href="http://sip address>:7001">http://sip address>:7001</a> to get access to Studio.

If you can see the **Config Server** page on the browser, Studio is started successfully.



STOP SERVICE

You can use kill pid to stop the service:

\$ kill \$(lsof -t -i :7001) #stop nebula-graph-studio

#### Docker-based Studio

PREREQUISITES

Before you deploy Docker-based Studio, you must do a check of these:

- The NebulaGraph services are deployed and started. For more information, see NebulaGraph Database Manual.
- On the machine where Studio will run, Docker Compose is installed and started. For more information, see Docker Compose Documentation.
- Before the installation starts, the following ports are not occupied.

Port	Description
7001	Web service provided by Studio

PROCEDURE

To deploy and start Docker-based Studio, run the following commands. Here we use NebulaGraph v3.6.0 for demonstration:

1. Download the configuration files for the deployment.

### Installation package NebulaGraph version

nebula-graph-studio-3.8.0.tar.gz 3.6.0

#### 2. Create the nebula-graph-studio-3.8.0 directory and decompress the installation package to the directory.

\$ mkdir nebula-graph-studio-3.8.0 -zxvf nebula-graph-studio-3.8.0.gz -C nebula-graph-studio-3.8.0

3. Change to the nebula-graph-studio-3.8.0 directory.

\$ cd nebula-graph-studio-3.8.0

4. Pull the Docker image of Studio.

\$ docker-compose pull

5. Build and start Docker-based Studio. In this command, I d is to run the containers in the background.

\$ docker-compose up -d

If these lines are returned, Docker-based Studio v3.x is deployed and started.

Creating docker\_web\_1 ... done

6. When Docker-based Studio is started, use http://<ip address>:7001 to get access to Studio.

## Note

Run ifconfig or ipconfig to get the IP address of the machine where Docker-based Studio is running. On the machine running Docker-based Studio, you can use <a href="http://localhost:7001">http://localhost:7001</a> to get access to Studio.

If you can see the **Config Server** page on the browser, Docker-based Studio is started successfully.



### Helm-based Studio

This section describes how to deploy Studio with Helm.

## PREREQUISITES

Before installing Studio, you need to install the following software and ensure the correct version of the software:

Software	Requirement
Kubernetes	>= 1.14
Helm	>= 3.2.0

INSTALL

1. Use Git to clone the source code of Studio to the host.

\$ git clone https://github.com/vesoft-inc/nebula-studio.git

#### 2. Make the nebula-studio directory the current working directory.

bash

#### \$ cd nebula-studio

3. Assume using release name: my-studio, installed Studio in Helm Chart.

\$ helm upgrade --install my-studio --set service.type=NodePort --set service.port=30070 deployment/helm

The configuration parameters of the Helm Chart are described below.

Parameter	Default value	Description
replicaCount	0	The number of replicas for Deployment.
image.nebulaStudio.name	vesoft/nebula-graph- studio	The image name of nebula-graph-studio.
image.nebulaStudio.version	v3.8.0	The image version of nebula-graph-studio.
service.type	ClusterIP	The service type, which should be one of $\ensuremath{NodePort}$ , $\ensuremath{ClusterIP}$ , and $\ensuremath{LoadBalancer}$ .
service.port	7001	The expose port for nebula-graph-studio's web.
service.nodePort	32701	The proxy port for accessing nebula-studio outside kubernetes cluster.
resources.nebulaStudio	{}	The resource limits/requests for nebula-studio.
persistent.storageClassName	ш	The name of storageClass. The default value will be used if not specified.
persistent.size	5Gi	The persistent volume size.

4. When Studio is started, use  $http://<node_address>:30070/$  to get access to Studio.

If you can see the **Config Server** page on the browser, Studio is started successfully.



#### UNINSTALL

\$ helm uninstall my-studio

## Next to do

On the **Config Server** page, connect Docker-based Studio to NebulaGraph. For more information, see Connect to NebulaGraph.

Last update: February 27, 2024

## 15.2.2 Connect to NebulaGraph

After successfully launching Studio, you need to configure to connect to NebulaGraph. This topic describes how Studio connects to the NebulaGraph database.

#### Prerequisites

Before connecting to the NebulaGraph database, you need to confirm the following information:

- The NebulaGraph services and Studio are started. For more information, see Deploy Studio.
- You have the local IP address and the port used by the Graph service of NebulaGraph. The default port is 9669.
- You have a NebulaGraph account and its password.

### Procedure

To connect Studio to NebulaGraph, follow these steps:

 Type http://<ip\_address>:7001 in the address bar of your browser. The following login page shows that Studio starts successfully.



- 2. On the **Config Server** page of Studio, configure these fields:
- Graphd IP address: Enter the IP address of the Graph service of NebulaGraph. For example, 192.168.10.100.

#### Q Note

- When NebulaGraph and Studio are deployed on the same machine, you must enter the IP address of the machine, instead of 127.0.0.1 or localhost.
- When connecting to a NebulaGraph database on a new browser tab, a new session will overwrite the sessions of the old tab. If you need to log in to multiple NebulaGraph databases simultaneously, you can use a different browser or non-trace mode.
- Port: The port of the Graph service. The default port is 9669.
- Username and Password: Fill in the log in account according to the authentication settings of NebulaGraph.
- If authentication is not enabled, you can use root and any password as the username and its password.
- If authentication is enabled and no account information has been created, you can only log in as GOD role and use root and nebula as the username and its password.
- If authentication is enabled and different users are created and assigned roles, users in different roles log in with their accounts and passwords.
- 3. After the configuration, click the **Connect** button.

## Note

One session continues for up to 30 minutes. If you do not operate Studio within 30 minutes, the active session will time out and you must connect to NebulaGraph again.

A welcome page is displayed on the first login, showing the relevant functions according to the usage process, and the test datasets can be automatically downloaded and imported.

To visit the welcome page, click 🕐.

### Next to do

When Studio is successfully connected to NebulaGraph, you can do these operations:

- Create a schema on the **Console** page or on the **Schema** page.
- Batch import data on the Import page.
- Execute nGQL statements on the **Console** page.
- Design the schema visually on the **Schema drafting** page.

## Note

The permissions of an account determine the operations that can be performed. For details, see Roles and privileges.

#### LOG OUT

If you want to reconnect to NebulaGraph, you can log out and reconfigure the database.

Click the user profile picture in the upper right corner, and choose **Log out**.

Last update: October 25, 2023

# 15.3 Quick start

## 15.3.1 Design a schema

To manipulate graph data in NebulaGraph with Studio, you must have a graph schema. This article introduces how to design a graph schema for NebulaGraph.

A graph schema for NebulaGraph must have these essential elements:

- Tags (namely vertex types) and their properties.
- Edge types and their properties.

In this article, you can install the sample data set basketballplayer and use it to explore a pre-designed schema.

This table gives all the essential elements of the schema.

Element	Name	Property name (Data type)	Description
Tag	player	<ul><li>name (string)</li><li>age (int)</li></ul>	Represents the player.
Tag	team	- name (string)	Represents the team.
Edge type	serve	<ul><li>start_year ( int )</li><li>end_year ( int )</li></ul>	Represent the players behavior. This behavior connects the player to the team, and the direction is from player to team.
Edge type	follow	- degree ( int )	Represent the players behavior. This behavior connects the player to the player, and the direction is from a player to a player.

This figure shows the relationship  $({\it serve/follow})$  between a player and a team.



Last update: October 25, 2023

## 15.3.2 Create a schema

To batch import data into NebulaGraph, you must have a graph schema. You can create a schema on the **Console** page or on the **Schema** page of Studio.

## Note

• Users can use nebula-console to create a schema. For more information, see NebulaGraph Manual and Get started with NebulaGraph.

• Users can use the Schema drafting function to design schema visually. For more information, see Schema drafting.

#### Prerequisites

To create a graph schema on Studio, you must do a check of these:

- Studio is connected to NebulaGraph.
- Your account has the privilege of GOD, ADMIN, or DBA.
- The schema is designed.
- A graph space is created.

#### Q Note

If no graph space exists and your account has the GOD privilege, you can create a graph space on the **Console** page. For more information, see CREATE SPACE.

#### Create a schema with Schema

- 1. Create tags. For more information, see Operate tags.
- 2. Create edge types. For more information, see Operate edge types.

#### Create a schema with Console

- 1. In the toolbar, click the  ${\bf Console}$  tab.
- 2. In the Current Graph Space field, choose a graph space name. In this example, basketballplayer is used.

Nebula Studio	🗄 Schema	ப் Import	⊵ Console	
baske	tballplayer		~ (?)	
Ne	ebula Console			
1	SHOW SPACES;			
	\$			

3. In the input box, enter these statements one by one and click the button **Run**.



If the preceding statements are executed successfully, the schema is created. You can run the statements as follows to view the schema.

```
// To list all the tags in the current graph space
nebula> SHOW TAGS;
// To list all the edge types in the current graph space
nebula> SHOW EDGES;
// To view the definition of the tags and edge types
DESCRIBE TAG player;
DESCRIBE TAG team;
DESCRIBE EDGE follow;
DESCRIBE EDGE follow;
```

If the schema is created successfully, in the result window, you can see the definition of the tags and edge types.

#### Next to do

When a schema is created, you can import data.

```
Last update: October 25, 2023
```

## 15.3.3 Import data

Studio supports importing data in CSV format into NebulaGraph through an interface.

#### Prerequisites

To batch import data, do a check of these:

- The schema has been created in NebulaGraph.
- The CSV files meet the demands of the schema.

.

• The account has GOD, ADMIN, or DBA permissions. For details, see Built-in Roles.

#### Entry

In the top navigation bar, click	Œ.
----------------------------------	----

Importing data is divided into 2 parts, creating a new data source and creating an import task, which will be described in detail next.

## Create a new data source

Click **New Data Source** in the upper right corner of the page to set the data source and its related settings. Currently, 3 types of data sources are supported.

Type of data source	Description
Cloud storage	Add cloud storage as the CSV file source, which only supports cloud services compatible with the Amazon S3 interface.
SFTP	Add SFTP as the CSV file source.
Local file	Upload a local CSV file. The file size can not exceed 200 MB, please put the files exceeding the limit into other types of data sources.

#### Q Note

• When uploading a local CSV file, you can select more than one CSV file at one time.

• After adding a data source, you can click **Data Source Management** at the top of the page and switch tabs to view the details of different types of data sources, and you can also edit or delete data sources.

#### Create an import task

1. Click **New Import** at the top left corner of the page to complete the following settings:

# Caution

Users can also click **Import Template** to download the sample configuration file example.yaml, configure it and then upload the configuration file. Configure in the same way as NebulaGraph Importer.

- **Space**: The name of the graph space where the data needs to be imported.
- Task Name: automatically generated by default, can be modified.
- (optional)**More configuration**: You can customize the concurrency, batch size, retry times, read concurrency, and import concurrency.
- Map Tags:
- a. Click Add Tag, and then select the tag within the added tags below.
- b. Click Add source file, select Data Source Type and File Path in Data source file, find the file you need to import, and then click Add.
- c. In the preview page, set the file separator and whether to carry the table header, and then click **Confirm**.
- d. Select the corresponding column for VID in **VID Columns**. You can select multiple columns to be merged into a VID, and you can also add a prefix or suffix to the VID.
- e. Select the corresponding column for the attribute in the properties box. For properties that can be NULL or have DEFAULT set, you can leave the corresponding column unspecified.
- f. Repeat steps 2 to 5 to import all the data files of the Tag selected.
- g. Repeat steps 1 to 6 to import all Tag data.
- Map Edges: Same operation as map tags.

Task List / New	Import						
ace				* Task Name			
asketballplayer	~			task-1693385507784			
			✓ Expand n	nore configurations			
ap Tags				* Map Edges			
+ Add Tag				+ Add Edge Typ	e		
v *Tag: p	layer V		×	✓ *Edge Type:	- serve - V		×
* Data source file	e: player.csv		×	* Data source	file: serve.csv		х
File Path: /play	ver.csv			File Path: /s	erve.csv		
> *VID colur	nn: Column 0		0	> *Source	e VID column: Column 0		0
Prop	CSV Index	Туре		> *Destin	ation VID column: Column 1		0
name	Column 1	string		Prop	CSV Index	Туре	
age	Column 2	int		start_year	Column 1	int	
	+ Add source file			end_year	Column 2	int	
				rank	Column 3	int	
	Cancel		s	Save Draft	Import		

2. After completing the settings, click **Import**, enter the password for the NebulaGraph account, and confirm.

After the import task is created, you can view the progress of the import task in the **Import Data** tab, which supports operations such as filtering tasks based on graph space, editing the task, viewing logs, downloading logs, reimporting, downloading configuration files, and deleting tasks.

#### Next

After completing the data import, users can access the Console page.

Last update: November 3, 2023

## 15.3.4 Console

Studio console interface is shown as follows.

Nebula St	udio 🖼 Schema 🕁 Import	☑ Console			0 (*)	0 🕓
2 basketballp	olayer v (	0				
Nebula 7 p1 8 1 MAT	Console CH (v:player) RETURN v LIMIT 3;			3 公	4 5 ℃ □ ^ Cyph	6 Run
9 \$ MA	TCH (v:player) RETURN v LIMIT 3				10 11 ☆	12 13 1 ~ ×
14 ⊞ Table 15 % Graph						
		player115	pie	yer106		<
			player102			+ -
Executio	n Time 0.002752 (s)					
\$ MA	TCH (v:player) RETURN v LIMIT 3				☆ .	1. <b>^</b> ×
⊞ Table	v					Å
ہے Graph	("player102" :player{age: 33, name: "La	Marcus Aldridge"})				
	("player106" :player{age: 25, name: "Ky	le Anderson"})				
	("player115" :player{age: 40, name: "Ko	be Bryant"})				

The following table lists various functions on the console interface.

number	function	descriptions
1	toolbar	Click the <b>Console</b> tab to enter the console page.
2	select a space	Select a space in the Current Graph Space list. <b>descriptions</b> : Studio does not support running the USE <space_name> statements directly in the input box.</space_name>
3	favorites	Click the button to expand the favorites, click one of the statements, and the input box will automatically enter the statement.
4	history list	Click C button representing the statement record. In the statement running record list, click one of the statements, and the statement will be automatically entered in the input box. The list provides the record of the last 15 statements.
5	clean input box	Click $\mathbf{\hat{D}}$ button to clear the content entered in the input box.
6	run	After inputting the nGQL statement in the input box, click button to indicate the operation to start running the statement.
7	custom parameters display	Click the button to expand the custom parameters for parameterized query. For details, see Manage parameters.
8	input box	After inputting the nGQL statements, click the button to run the statement. You can input multiple statements and run them at the same time by using the separator ; , and also use the symbol // to add comments.
9	statement running status	After running the nGQL statement, the statement running status is displayed. If the statement runs successfully, the statement is displayed in green. If the statement fails, the statement is displayed in red.
10	add to favorites	Click the button to save the statement as a favorite, the button for the favorite statement is colored in yellow exhibit.
11	export CSV file or PNG file	After running the nGQL statement to return the result, when the result is in <b>Table</b> window, click the button to export as a CSV file. Switch to the <b>Graph</b> window and click the button to save the results as a CSV file or PNG image export.
12	expand/hide execution results	Click the ^ button to hide the result or click  button to expand the result.
13	close execution results	Click the $ imes$ button to close the result returned by this nGQL statement.
14	Table window	Display the result from running nGQL statement. If the statement returns results, the window displays the results in a table.
15	Graph window	Display the result from running nGQL statement. If the statement returns the complete vertex-edge result, the window displays the result as a graph . Click the button on the right to view the overview panel.

Last update: October 25, 2023

## 15.3.5 Use Schema

#### Manage graph spaces

When Studio is connected to NebulaGraph, you can create or delete a graph space. You can use the **Console** page or the **Schema** page to do these operations. This article only introduces how to use the **Schema** page to operate graph spaces in NebulaGraph.

#### PREREQUISITES

To operate a graph space on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to NebulaGraph.
- Your account has the authority of GOD. It means that:
- If the authentication is enabled in NebulaGraph, you can use root and any password to sign in to Studio.
- If the authentication is disabled in NebulaGraph, you must use root and its password to sign in to Studio.

CREATE A GRAPH SPACE

1. In the toolbar, click the **Schema** tab.

- 2. In the Graph Space List page, click Create Space, do these settings:
- Name: Specify a name to the new graph space. In this example, basketballplayer is used. The name must be unique in the database.
- Vid Type: The data types of VIDs are restricted to FIXED\_STRING(<N>) or INT64. A graph space can only select one VID type. In this example, FIXED\_STRING(32) is used. For more information, see VID.
- **Comment**: Enter the description for graph space. The maximum length is 256 bytes. By default, there will be no comments on a space. But in this example, Statistics of basketball players is used.
- **Optional Parameters**: Set the values of partition\_num and replica\_factor respectively. In this example, these parameters are set to 100 and 1 respectively. For more information, see CREATE SPACE syntax.

In the **Equivalent to the following nGQL statement** panel, you can see the statement equivalent to the preceding settings.

CREATE SPACE basketballplayer (partition\_num = 100, replica\_factor = 1, vid\_type = FIXED\_STRING(32)) COMMENT = "Statistics of basketball players"

3. Confirm the settings and then click the + **Create** button. If the graph space is created successfully, you can see it on the graph space list.

Nebula Studio t= Schema du Import E	Console			0	۲	0	R
← Graph Space List / Create Space	3						
* Name	*	Vid Type	* Length				
Comment		PIAED_STRING V	32				
Statistics of basketball players							
Partition_num:(Optional)	R	eplica_factor:(Optional)					
100		1					
✓ View nGQL							
1 CREATE SPACE `basketballplayer` ketball players"	<pre>(partition_num = 100, replica_factor)</pre>	or = 1, vid_type = FIXED_STRING	:(32)) COMMENT = "Stat	istics o	of bas		
	Cancel	Create					

DELETE A GRAPH SPACE



- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List, find the space you want to be deleted, and click Delete Graph Space in the Operation column.

Nebula Studio 🚦 Schema 👜 Import 🖸 Console	۲	0	R
Graph Space List			
+ Create Space			
No Name Partition Replica Charset Collate Vid Type Atomic Group Comment Operations Number Factor Edge			
1 basketball			
2 hello_test 100 1 utf8 utf8_bin INT64 false _EMPTY_ Schema	Clo	ne Graph	Space
3 test 15 1 utf8 utf8_bin FIXED_ST RING(30) EMPTY_ Schema	0 0		
<	1	>	

3. On the dialog box, confirm the information and then click  $\mathbf{OK}$ .

NEXT TO DO

After a graph space is created, you can create or edit a schema, including:

- Operate tags
- Operate edge types
- Operate indexes

Last update: October 25, 2023

#### Manage tags

After a graph space is created in NebulaGraph, you can create tags. With Studio, you can use the **Console** page or the **Schema** page to create, retrieve, update, or delete tags. This topic introduces how to use the **Schema** page to operate tags in a graph space only.

#### PREREQUISITES

To operate a tag on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to NebulaGraph.
- A graph space is created.
- Your account has the authority of GOD, ADMIN, or DBA.

CREATE A TAG

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4. Click the **Tag** tab and click the **+ Create** button.
- 5. On the **Create** page, do these settings:
- Name: Specify an appropriate name for the tag. In this example, course is specified.
- **Comment** (Optional): Enter the description for tag.
- Define Properties (Optional): If necessary, click + Add Property to do these settings:
- Enter a property name.
- Select a data type.
- Select whether to allow null values..
- (Optional) Enter the default value.
- (Optional) Enter the description.
- Set TTL (Time To Live) (Optional): If no index is set for the tag, you can set the TTL configuration: In the upper left corner of the Set TTL panel, click the check box to expand the panel, and configure TTL\_COL and TTL\_ DURATION (in seconds). For more information about both parameters, see TTL configuration.
- 6. When the preceding settings are completed, in the **Equivalent to the following nGQL statement** panel, you can see the nGQL statement equivalent to these settings.

Nebula Stud	lio 👍	Schema	≰ Import	🔈 Console								0	۲	0	R
	← Gra	aph Space Li	ist / bask	etballplayer Tag	List / Create T	ag			Curre	nt Graph Space:	basketba	llplayer	V		
	* Name player						Comm	ent							
	Define	Properties													
		+ Add Prope	rty												
	•	Property Name		* Data Type		Allow Null		Defaults		Comment					
		age		int	~							Delete			
	Ē	name		fixed_string	~ 64							Delete			
	Set TT	TL (Time To Live	P)					JRATION							
				V			Pleas	se enter the time (in sec	conds)						
	V View	w nGQL EATE tag `p]	layer` (`a	ge` int NULL ,	`name` fixed_s	string(64) N	ULL )								
													1		
								<b>0</b>		1					

7. Confirm the settings and then click the **+ Create** button.

When the tag is created successfully, the **Define Properties** panel shows all its properties on the list.

EDIT A TAG

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4.

Click the **Tag** tab, find a tag and then click the button



in the **Operations** column.

- 5. On the **Edit** page, do these operations:
- To edit a Comment: Click **Edit** on the right of Comment.
- To edit a property: On the **Define Properties** panel, find a property, click **Edit**, and then change the data type or the default value.
- To delete a property: On the **Define Properties** panel, find a property, click **Delete**.
- To add more properties: On the Define Properties panel, click the Add Property button to add a new property.
- To set the TTL configuration: In the upper left corner of the **Set TTL** panel, click the check box and then set TTL.
- To delete the TTL configuration: When the **Set TTL** panel is expanded, in the upper left corner of the panel, click the check box to delete the configuration.
- To edit the TTL configuration: On the **Set TTL** panel, click **Edit** and then change the configuration of TTL\_COL and TTL\_DURATION (in seconds).

## Note

For the problem of the coexistence of TTL and index, see TTL.

#### DELETE A TAG

**D**anger Confirm the impact before deleting the tag. The deleted data cannot be restored if it is not backup.

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4. Click the **Tag** tab, find an tag and then click the button **(D)** in the **Operations** column.
- 5. Click **OK** to confirm delete a tag in the pop-up dialog box.

#### NEXT TO DO

After the tag is created, you can use the **Console** page to insert vertex data one by one manually or use the **Import** page to bulk import vertex data.

Last update: April 7, 2024

#### Manage edge types

After a graph space is created in NebulaGraph, you can create edge types. With Studio, you can choose to use the **Console** page or the **Schema** page to create, retrieve, update, or delete edge types. This topic introduces how to use the **Schema** page to operate edge types in a graph space only.

#### PREREQUISITES

To operate an edge type on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to NebulaGraph.
- A graph space is created.
- Your account has the authority of GOD, ADMIN, or DBA.

CREATE AN EDGE TYPE

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4. Click the **Edge Type** tab and click the **+ Create** button.
- 5. On the **Create Edge Type** page, do these settings:
- Name: Specify an appropriate name for the edge type. In this example, serve is used.
- Comment (Optional): Enter the description for edge type.
- Define Properties (Optional): If necessary, click + Add Property to do these settings:
- Enter a property name.
- Select a data type.
- Select whether to allow null values..
- (Optional) Enter the default value.
- (Optional) Enter the description.
- Set TTL (Time To Live) (Optional): If no index is set for the edge type, you can set the TTL configuration: In the upper left corner of the Set TTL panel, click the check box to expand the panel, and configure TTL\_COL and TTL\_DURATION (in seconds). For more information about both parameters, see TTL configuration.
- 6. When the preceding settings are completed, in the **Equivalent to the following nGQL statement** panel, you can see the nGQL statement equivalent to these settings.
| Nebula Studio 🛯 🔁 Schema | 占 Import   🖸 Console                      |                               |                                             | <b>?</b>         | ? R |
|--------------------------|-------------------------------------------|-------------------------------|---------------------------------------------|------------------|-----|
| ← Graph Space List       | / Edge Type List / Create Edge            | Туре                          | Current Graph Space:                        | basketballplayer | ~   |
| * Name                   |                                           | Comment                       |                                             |                  |     |
| Seive                    |                                           |                               |                                             |                  |     |
| ✓ Define Properties      |                                           |                               |                                             |                  |     |
| + Add Property           |                                           |                               |                                             |                  |     |
| * Property Name          | * Data Type                               | Allow Null Default            | s Comment                                   |                  |     |
| start_year               | int v                                     |                               |                                             | Delete           |     |
| end_year                 | int v                                     |                               |                                             | Delete           |     |
| teamID                   | string ~                                  |                               |                                             | Delete           |     |
| playerID                 | string $\lor$                             |                               |                                             | Delete           |     |
|                          |                                           |                               |                                             |                  |     |
| Set TTL (Time To Live)   |                                           |                               |                                             |                  |     |
| TTL_COL                  |                                           | TTL_DURATION                  | N                                           |                  |     |
|                          | V                                         | Please enter                  | the time (in seconds)                       |                  |     |
|                          |                                           |                               |                                             |                  |     |
| ✓ View nGQL              |                                           |                               |                                             |                  |     |
| 1 CREATE edge `ser       | <pre>ve` (`start_year` int NULL , `</pre> | `end_year` int NULL , `teamID | <pre>`string NULL , `playerID` string</pre> | ng NULL )        |     |
|                          |                                           | Cancel                        | Create                                      |                  |     |

7. Confirm the settings and then click the **+ Create** button.

When the edge type is created successfully, the **Define Properties** panel shows all its properties on the list.

EDIT AN EDGE TYPE

- 1. In the toolbar, click the  ${\bf Schema}$  tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.

4.



Click the Edge Type tab, find an edge type and then click the button

in the **Operations** column.

- 5. On the **Edit** page, do these operations:
- $\bullet$  To edit a comment: Click  ${\bf Edit}$  on the right of  ${\tt Comment}$  .
- To edit a property: On the **Define Properties** panel, find a property, click **Edit**, and then change the data type or the default value.
- To delete a property: On the **Define Properties** panel, find a property, click **Delete**.
- To add more properties: On the Define Properties panel, click the Add Property button to add a new property.
- To set the TTL configuration: In the upper left corner of the **Set TTL** panel, click the check box and then set TTL.
- To delete the TTL configuration: When the **Set TTL** panel is expanded, in the upper left corner of the panel, click the check box to delete the configuration.
- To edit the TTL configuration: On the **Set TTL** panel, click **Edit** and then change the configuration of TTL\_COL and TTL\_DURATION (in seconds).

### Note

For information about the coexistence problem of TTL and index, see TTL.

DELETE AN EDGE TYPE

**Panger** Confirm the impact before deleting the Edge type. The deleted data cannot be restored if it is not backup.

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.

4. Click the **Edge Type** tab, find an edge type and then click the button in the **Operations** column.

5. Click  $\boldsymbol{OK}$  to confirm in the pop-up dialog box.

#### NEXT TO DO

After the edge type is created, you can use the **Console** page to insert edge data one by one manually or use the **Import** page to bulk import edge data.

Last update: April 7, 2024

#### Manage indexes

You can create an index for a Tag and/or an Edge type. An index lets traversal start from vertices or edges with the same property and it can make a query more efficient. With Studio, you can use the **Console** page or the **Schema** page to create, retrieve, and delete indexes. This topic introduces how to use the **Schema** page to operate an index only.

# Note

You can create an index when a Tag or an Edge Type is created. But an index can decrease the write speed during data import. We recommend that you import data firstly and then create and rebuild an index. For more information, see Index overview.

PREREQUISITES

To operate an index on the Schema page of Studio, you must do a check of these:

- Studio is connected to NebulaGraph.
- A graph Space, Tags, and Edge Types are created.
- Your account has the authority of GOD, ADMIN, or DBA.

#### CREATE AN INDEX

1. In the toolbar, click the **Schema** tab.

- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4. Click the **Index** tab and then click the **+ Create** button.
- 5. On the **Create** page, do these settings:
- Index Type: Choose to create an index for a tag or for an edge type. In this example, Edge Type is chosen.
- Associated tag name: Choose a tag name or an edge type name. In this example, follow is chosen.
- Index Name: Specify a name for the new index. In this example, follow\_index is used.
- Comment (Optional): Enter the description for index.
- Indexed Properties (Optional): Click Add property, and then, in the dialog box, choose a property. If necessary, repeat this step to choose more properties. You can drag the properties to sort them. In this example, degree is chosen.

#### Q Note

The order of the indexed properties has an effect on the result of the LOOKUP statement. For more information, see nGQL Manual.

6. When the settings are done, the **Equivalent to the following nGQL statement** panel shows the statement equivalent to the settings.

Nebula Studio 陆 Schema പ്ര Import 🖂 🕻	sole	0	•	2) R
← Graph Space List / Index List / Cre	e Index Curre	nt Graph Space: basketballplayer	~	
* Index Type	* Associated edge name			
* Index Name	Comment			
follow_index	follow_index			
Indexed Properties(Drag to Sort) + Add Property				
✓ View nGQL				
1 CREATE EDGE INDEX `follow_index` c	`follow`() COMMENT "follow_index"			
			- 1/	
	Cancel Create			

7. Confirm the settings and then click the **+ Create** button. When an index is created, the index list shows the new index.

VIEW INDEXES

- 1. In the toolbar, click the  ${\bf Schema}$  tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4. Click the Index tab, in the upper left corner, choose an index type, Tag or Edge Type.
- 5. In the list, find an index and click its row. All its details are shown in the expanded row.

REBUILD INDEXES

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4. Click the Index tab, in the upper left corner, choose an index type, Tag or Edge Type.
- 5. Click the Index tab, find an index and then click the button **Rebuild** in the **Operations** column.

#### Q Note

For more Information, see **REBUILD INDEX**.

DELETE AN INDEX

To delete an index on **Schema**, follow these steps:

- 1. In the toolbar, click the **Schema** tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
- 4.
  - . Click the **Index** tab, find an index and then click the button 🔟 in the **Operations** column.
- 5. Click **OK** to confirm in the pop-up dialog box.

Last update: October 25, 2023

#### View Schema

Users can visually view schemas in NebulaGraph Studio.

STEPS

- 1. In the toolbar, click the  ${\bf Schema}$  tab.
- 2. In the Graph Space List page, find a graph space and then click its name or click Schema in the Operations column.
- 3. Click  $\ensuremath{\textit{View Schema}}$  tab and click the  $\ensuremath{\textit{Get Schema}}$  button.

OTHER OPERATIONS

In the Graph Space List page, find a graph space and then perform the following operations in the Operations column:

- View Schema DDL: Displays schema creation statements for the graph space, including graph spaces, tags, edge types, and indexes.
- Clone Graph Space: Clones the schema of the graph space to a new graph space.
- Delete Graph pace: Deletes the graph space, including the schema and all vertices and edges.

#### 15.3.6 Schema drafting

Studio supports the schema drafting function. Users can design their schemas on the canvas to visually display the relationships between vertices and edges, and apply the schema to a specified graph space after the design is completed.

#### Features

- Design schema visually.
- Applies schema to a specified graph space.
- Export the schema as a PNG image.

#### Entry

At the top navigation bar, click 😵 .

#### Design schema

The following steps take designing the schema of the basketballplayer dataset as an example to demonstrate how to use the schema drafting function.

- 1. At the upper left corner of the page, click New.
- 2. Create a tag by selecting the appropriate color tag under the canvas. You can hold down the left button and drag the tag into the canvas.
- 3. Click the tag. On the right side of the page, you need to fill in the name of the tag as player, and add two properties name and age.
- 4. Create a tag again. The name of the tag is team, and the property is name.
- 5. Connect from the anchor point of the tag player to the anchor point of the tag team. Click the generated edge, fill in the name of the edge type as serve, and add two properties start\_year and end\_year.
- 6. Connect from an anchor point of the tag player to another one of its own. Click the generated edge, fill in the name of the edge type as follow, and add a property degree.
- 7.
- After the design is complete, click 🗹 at the top of the page to change the name of the draft, and then click 🛱 at the top right corner to save the draft.

basketballplayer 🖄	Apply to Space
	Copy
	Tag
	* Tag Name
	player
	Comment
player	
	Properties + Add Property
	Property Name Data Type
	name string v ×
	age int64 $\vee$ $\times$

#### Apply schema

- 1. Select the draft that you want to import from the **Draft list** on the left side of the page, and then click **Apply to Space** at the upper right corner.
- 2. Import the schema to a new or existing space, and click Confirm.

# Note

- For more information about the parameters for creating a graph space, see CREATE SPACE.
- If the same schema exists in the graph space, the import operation fails, and the system prompts you to modify the name or change the graph space.

#### Modify schema

Select the schema draft that you want to modify from the **Draft list** on the left side of the page. Click  $\square$  at the upper right corner after the modification.

#### Q Note

The graph space to which the schema has been applied will not be modified synchronously.

#### Delete schema

Select the schema draft that you want to delete from the **Draft list** on the left side of the page, click X at the upper right corner of the thumbnail, and confirm to delete it.

# Export Schema

Click  $\hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \hfill \$ 

# 15.4 Troubleshooting

#### 15.4.1 Connecting to the database error

#### **Problem description**

According to the connect Studio operation, it prompts failed.

#### Possible causes and solutions

You can troubleshoot the problem by following the steps below.

STEP1: CONFIRM THAT THE FORMAT OF THE HOST FIELD IS CORRECT

You must fill in the IP address (graph\_server\_ip) and port of the NebulaGraph database Graph service. If no changes are made, the port defaults to 9669. Even if NebulaGraph and Studio are deployed on the current machine, you must use the local IP address instead of 127.0.0.1, localhost or 0.0.0.0.

STEP2: CONFIRM THAT THE USERNAME AND PASSWORD ARE CORRECT

If authentication is not enabled, you can use root and any password as the username and its password.

If authentication is enabled and different users are created and assigned roles, users in different roles log in with their accounts and passwords.

STEP3: CONFIRM THAT NEBULAGRAPH SERVICE IS NORMAL

Check NebulaGraph service status. Regarding the operation of viewing services:

- If you compile and deploy NebulaGraph on a Linux server, refer to the NebulaGraph service.
- If you use NebulaGraph deployed by Docker Compose and RPM, refer to the NebulaGraph service status and ports.

If the NebulaGraph service is normal, proceed to Step 4 to continue troubleshooting. Otherwise, please restart NebulaGraph service.

### Note

If you used docker-compose up -d to satrt NebulaGraph before, you must run the docker-compose down to stop NebulaGraph.

STEP4: CONFIRM THE NETWORK CONNECTION OF THE GRAPH SERVICE IS NORMAL

Run a command (for example, telnet 9669) on the Studio machine to confirm whether NebulaGraph's Graph service network connection is normal.

If the connection fails, check according to the following steps:

- If Studio and NebulaGraph are on the same machine, check if the port is exposed.
- If Studio and NebulaGraph are not on the same machine, check the network configuration of the NebulaGraph server, such as firewall, gateway, and port.

If you cannot connect to the NebulaGraph service after troubleshooting with the above steps, please go to the NebulaGraph forum for consultation.

Last update: November 3, 2023

#### 15.4.2 Cannot access to Studio

#### **Problem description**

I follow the document description and visit 127.0.0.1:7001 or 0.0.0.0:7001 after starting Studio, why can't I open the page?

#### Possible causes and solutions

You can troubleshoot the problem by following the steps below.

STEP1: CONFIRM SYSTEM ARCHITECTURE

It is necessary to confirm whether the machine where the Studio service is deployed is of  $x86_64$  architecture. Currently, Studio only supports  $x86_64$  architecture.

STEP2: CHECK IF THE STUDIO SERVICE STARTS NORMALLY

- For Studio deployed with RPM or DEB packages, use systemctl status nebula-graph-studio to see the running status.
- For Studio deployed with tar package, use sudo lsof -i:7001 to check port status.
- For Studio deployed with docker, use docker-compose ps to see the running status. Run docker-compose ps to check if the service has started normally.

If the service is normal, the return result is as follows. Among them, the State column should all be displayed as Up.

Name	Command	State		Ports
nebula-web-docker_client_1	./nebula-go-api		Up	0.0.0:32782->8080/tcp
nebula-web-docker_importer_1	nebula-importerport=	569	Up	0.0.0:32783->5699/tcp
nebula-web-docker_nginx_1	/docker-entrypoint.sh n	gin	Up	0.0.0.0:7001->7001/tcp, 80/tcp
nebula-web-docker_web_1	docker-entrypoint.sh np	m r	Up	0.0.0:32784->7001/tcp

If the above result is not returned, stop Studio and restart it first. For details, refer to Deploy Studio.

#### !!! note

If you used `docker-compose up -d` to satrt NebulaGraph before, you must run the `docker-compose down` to stop NebulaGraph.

STEP3: CONFIRM ADDRESS

If Studio and the browser are on the same machine, users can use localhost:7001, 127.0.0.1:7001 or 0.0.0.0:7001 in the browser to access Studio.

If Studio and the browser are not on the same machine, you must enter <studio\_server\_ip>:7001 in the browser. Among them, studio\_server\_ip refers to the IP address of the machine where the Studio service is deployed.

STEP4: CONFIRM NETWORK CONNECTION

Run curl <studio\_server\_ip>:7001 -I to confirm if it is normal. If it returns HTTP/1.1 200 OK, it means that the network is connected normally.

If the connection is refused, check according to the following steps:

If the connection fails, check according to the following steps:

- If Studio and NebulaGraph are on the same machine, check if the port is exposed.
- If Studio and NebulaGraph are not on the same machine, check the network configuration of the NebulaGraph server, such as firewall, gateway, and port.

If you cannot connect to the NebulaGraph service after troubleshooting with the above steps, please go to the NebulaGraph forum for consultation.

# 15.4.3 FAQ

# Why can't I use a function?

If you find that a function cannot be used, it is recommended to troubleshoot the problem according to the following steps:

. Confirm that NebulaGraph is the latest version. If you use Docker Compose to deploy the NebulaGraph database, it is recommended to run docker-compose pull & docker-compose up -d to pull the latest Docker image and start the container.

2. Confirm that Studio is the latest version. For more information, refer to check updates.

3. Search the nebula forum, nebula and nebula-studio projects on the GitHub to confirm if there are already similar problems.

4. If none of the above steps solve the problem, you can submit a problem on the forum.

# 16. Dashboard (Community)

# 16.1 What is NebulaGraph Dashboard Community Edition

NebulaGraph Dashboard Community Edition (Dashboard for short) is a visualization tool that monitors the status of machines and services in NebulaGraph clusters.

# Sterpriseonly

Dashboard Enterprise Edition adds features such as visual cluster creation, batch import of clusters, fast scaling, etc. For more information, see Pricing.

#### 16.1.1 Features

Dashboard monitors:

- The status of all the machines in clusters, including CPU, memory, load, disk, and network.
- The information of all the services in clusters, including the IP addresses, versions, and monitoring metrics (such as the number of queries, the latency of queries, the latency of heartbeats, and so on).
- The information of clusters, including the information of services, partitions, configurations, and long-term tasks.
- Set how often the metrics page refreshes.

#### 16.1.2 Scenarios

You can use Dashboard in one of the following scenarios:

- You want to monitor key metrics conveniently and quickly, and present multiple key information of the business to ensure the business operates normally.
- You want to monitor clusters from multiple dimensions (such as the time, aggregate rules, and metrics).
- After a failure occurs, you need to review it and confirm its occurrence time and unexpected phenomena.

#### 16.1.3 Precautions

The monitoring data will be retained for 14 days by default, that is, only the monitoring data within the last 14 days can be queried.

# Note

The monitoring service is supported by Prometheus. The update frequency and retention intervals can be modified. For details, see Prometheus.

# 16.1.4 Version compatibility

The version correspondence between NebulaGraph and Dashboard Community Edition is as follows.

NebulaGraph version	Dashboard version
3.6.0	3.4.0
3.5.x	3.4.0
3.4.0 ~ 3.4.1	3.4.0 \ 3.2.0
3.3.0	3.2.0
2.5.0 ~ 3.2.0	3.1.0
2.5.x ~ 3.1.0	1.1.1
2.0.1~2.5.1	1.0.2
2.0.1~2.5.1	1.0.1

#### 16.1.5 Release note

Release

Last update: November 6, 2023

# 16.2 Deploy Dashboard Community Edition

This topic will describe how to deploy NebulaGraph Dashboard in detail.

To download and compile the latest source code of Dashboard, follow the instructions on the nebula dashboard GitHub page.

#### 16.2.1 Prerequisites

Before you deploy Dashboard, you must confirm that:

- The NebulaGraph services are deployed and started. For more information, see NebulaGraph Database Manual.
- Before the installation starts, the following ports are not occupied.
- 9200
- 9100
- 9090
- 8090
- 7003

• The node-exporter is installed on the machines to be monitored. For details on installation, see Prometheus document.

#### 16.2.2 Steps

- 1. Download the tar packagenebula-dashboard-3.4.0.x86\_64.tar.gz as needed.
- 2. Run tar -xvf nebula-dashboard-3.4.0.x86\_64.tar.gz to decompress the installation package.
- 3. Modify the config.yaml file in nebula-dashboard.

The configuration file contains the configurations of four dependent services and configurations of clusters. The descriptions of the dependent services are as follows.

Service	Default port	Description
nebula-http- gateway	8090	Provides HTTP ports for cluster services to execute nGQL statements to interact with the NebulaGraph database.
nebula-stats- exporter	9200	Collects the performance metrics in the cluster, including the IP addresses, versions, and monitoring metrics (such as the number of queries, the latency of queries, the latency of heartbeats, and so on).
node-exporter	9100	Collects the source information of nodes in the cluster, including the CPU, memory, load, disk, and network.
prometheus	9090	The time series database that stores monitoring data.

#### The descriptions of the configuration file are as follows.

```
# port: 9100
# https: false
 prometheus
   ip: hostIP # The IP of the machine where the Dashboard is deployed.
   prometheusPort: 9090
   https: false # Whether to enable HTTPS.
   scrape_interval: 5s # The interval for collecting the monitoring data, which is 1 minute by default.
evaluation_interval: 5s # The interval for running alert rules, which is 1 minute by default.
  # Cluster node info
 nebula-cluster:
   name: 'default' # Cluster name
   metad:
      - name: metad0
        endpointIP: nebulaMetadIP # The IP of the machine where the Meta service is deployed.
        port: 9559
        endpointPort: 19559
    # - name: metadl
    # endpointIP: nebulaMetadIP
       port: 9559
       endpointPort: 19559
   graphd:
       name: graphd0
        endpointIP: nebulaGraphdIP # The IP of the machine where the Graph service is deployed.
        port: 9669
        endpointPort: 19669
    # - name: graphd1
# endpointIP: nebulaGraphdIP
       port: 9669
endpointPort: 19669
    #
    storaged:
       - name: storaged0
        endpointIP: nebulaStoragedIP # The IP of the machine where the Storage service is deployed.
        .
port: 9779
        endpointPort: 19779
    # - name: storaged1
    # endpointIP: nebulaStoragedIP
        port: 9779
    #
        endpointPort: 19779
```

4. Run ./dashboard.service start all to start the services.

#### Deploy Dashboard with Docker Compose

If you are deploying Dashboard using docker, you should also modify the configuration file config.yaml, and then run docker-compose up -d to start the container.

# Note

If you change the port number in config.yaml, the port number in docker-compose.yaml needs to be consistent as well.

Run docker-compose stop to stop the container.

#### 16.2.3 Manage services in Dashboard

You can use the dashboard.service script to start, restart, stop, and check the Dashboard services.

sudo <dashboard\_path>/dashboard.service
[-v] [-h]
<start|restart|stop|status> <prometheus|webserver|exporter|gateway|all>

Parameter	Description
dashboard_path	Dashboard installation path.
- V	Display detailed debugging information.
-h	Display help information.
start	Start the target services.
restart	Restart the target services.
stop	Stop the target services.
status	Check the status of the target services.
prometheus	Set the prometheus service as the target service.
webserver	Set the webserver Service as the target service.
exporter	Set the exporter Service as the target service.
gateway	Set the gateway Service as the target service.
all	Set all the Dashboard services as the target services.

# Q Note

To view the Dashboard version, run the command ./dashboard.service -version.

#### 16.2.4 Next to do

#### Connect to Dashboard

# 16.3 Connect Dashboard

After Dashboard is deployed, you can log in and use Dashboard on the browser.

#### 16.3.1 Prerequisites

- The Dashboard services are started. For more information, see Deploy Dashboard.
- We recommend you to use the Chrome browser of the version above 89. Otherwise, there may be compatibility issues.

#### 16.3.2 Procedures

- 1. Confirm the IP address of the machine where the Dashboard service is installed. Enter <IP>:7003 in the browser to open the login page.
- 2. Enter the username and the passwords of the NebulaGraph database.
- If authentication is enabled, you can log in with the created accounts.
- If authentication is not enabled, you can only log in using root as the username and random characters as the password. To enable authentication, see Authentication.
- $3. \ Select \ the \ NebulaGraph \ version \ to \ be \ used.$
- 4. Click Login.

# 16.4 Dashboard

NebulaGraph Dashboard consists of three parts: Machine, Service, and Management. This topic will describe them in detail.

# 16.4.1 Overview

1 Hour	6 Hours	12 Hours	1 Day	3 Days	7 Days	14 Days	2023-09-7	12 15:	46 2023-0	9-12 16:46	instance	all ×		
refresh fre	equency:	C <sup>•</sup> Of	if ∨											
CPU cpu	_utilization						Ľ	0	Memory mer	nory_used_perce	ntage			C (
100% —									100%					
80%									80%					
60%									60%					
40%									40%					
20%									20%					
0% -	16:10		16:20		16:30		16:40	_	0%	16:10	16:20	16:30	16:40	
			<b>—</b> 192	.168.8.111:	9100						<b>—</b> 192.168.	8.111:9100		
oad load	l_15s						C	0	Disk disk_used	d_percentage		AI	I Instances 🔻	<
6									instance	Disk Na	ame	Disk Used		
5												91.82GB/99.99GB		
2										/dev/sda	1			92
0									102 168 8 111	.0100				

# 16.4.2 Machine

Click **Machine->Overview** to enter the machine overview page.

On this page, you can view the variation of CPU, Memory, Load, Disk, and Network In/Out quickly.

- By default, you can view the monitoring data for a maximum of 14 days. You can also select a time range or quickly select the latest 1 hour, 6 hours, 12 hours, 1 day, 3 days, 7 days, or 14 days.
- By default, you can view the monitoring data of all the instances in clusters. You can select the instances you want to view in the **instance** box.
- By default, the monitoring information page will not be updated automatically. You can set the update frequency of the

monitoring information page globally or click the C<sup>\*</sup> button to update the page manually.



To set a base line, click the

**—** 



To view the detailed monitoring information, click the follows.

button. In th	s example,	select	Load	for	details.	The	figure	is	as
	o onampio,	001000		101	ao oano.		ngaro	10	ao

1 Hour	6 Hours	12 Hours	1 Day	3 Days	7 Days	14 Days	2023-09-7	11 16:45	→ 2023-09	9-12 16:45	instance:	all $\times$	
efresh fre	equency:	C Of	f 🗸 N	/letric:	PI	ease Searc	h Metric						
			load	_15s 🕕		•	💿 🗹 🛛 Base Lin	e			load_5m	1 (1)	💿 🗹 🛛 Base Lii
								- 6	3				
								- 5	5				
								- 4					
								- 3	3				
								- 2	2				
								- 1					
								- 0	)				
30			- 192.1	68.8.111:9	100			1	6:30		- 192.168.8.1	111:9100	

- You can set the monitoring time range, instance, update frequency and base line.
- You can search for or select the target metric. For details about monitoring metrics, see Metrics.
- You can temporarily hide nodes that you do not need to view.



ວຸ

button to view the detailed monitoring information.

#### 16.4.3 Service

Click **Service->Overview** to enter the service overview page.

On this page, you can view the information of Graph, Meta, and Storage services quickly. In the upper right corner, the number of normal services and abnormal services will be displayed.

Note

In the Service page, only two monitoring metrics can be set for each service, which can be adjusted by clicking the Set up button.

- By default, you can view the monitoring data for a maximum of 14 days. You can also select a time range or quickly select the latest 1 hour, 6 hours, 12 hours, 1 day, 3 days, 7 days, or 14 days.
- By default, you can view the monitoring data of all the instances in clusters. You can select the instances you want to view in the **instance** box.
- By default, the monitoring information page will not be updated automatically. You can set the update frequency of the

monitoring information page globally or click the U button to update the page manually.

- You can view the status of all the services in a cluster.
- $\odot$

button. In this example, select Graph for details. The figure is as

To view the detailed monitoring information, click the follows.

1 Hour 6 Hours	12 Hours 1 Day 3 Days	7 Days 14 Days	2023-09-11	16:42 - 2023-0	09-12 16:42 📋 instance : all ×	
fresh frequency :	C Off ∨ Metric:	Please Sear	ch Metric	Spaces ?:	All	
	context_switches_t	otal 🕕	I Base Line		cpu_seconds_total ①	⊚ 🖄 Base Lin
				170/s		
				136/s		
				102/s		
				68/s		
				34/s		
				0/s		
6:30	— graphd0			16:30	— graphd0	

- You can set the monitoring time range, instance, update frequency, period, aggregation and base line.
- You can search for or select the target metric. For details of monitoring metrics, see Monitor parameter.
- You can temporarily hide nodes that you do not need to view.

You can click the



button to view the detailed monitoring information.

• The Graph service supports a set of space-level metrics. For more information, see the following section **Graph space**.

#### Graph space



Space graph metrics record the information of different graph spaces separately. Currently, only the Graph service supports a set of space-level metrics.

For information about the space graph metrics, see Graph space.

Query Conditions			
Period :	60	•	
Metric :	num_queries	•	0
Spaces:	basketballplayer	•	
Methods :	rate	•	
Base Line :			
	Cancel Confirm		

#### 16.4.4 Management

#### Overview info

On the **Overview Info** page, you can see the information of the NebulaGraph cluster, including Storage leader distribution, Storage service details, versions and hosts information of each NebulaGraph service, and partition distribution and details.

Info Overview									
Storage Leader Distrib	ution					Balance Lead	ler Detail	Version	Detail
		Service		Leader Number	Lead	der distribution		Graph Service	Varia
	192.168.8.111:5779 192.168.8.111:6779	192.168.8.111:	5779	2	basl	ketballplayer:2		192.168.8.111:9669	version
	192.168.8.111:9779	192.168.8.111:	6779	2	basl	ketballplayer:2		192.168.8.111:7669	
		192.168.8.111:	7779	3	basl	ketballplayer:3		Storage Service	Version
		192.168.8.111:	9779	3	basl	ketballplayer:3		192.168.8.111:9779	
								192.168.8.111:7779 192.168.8.111:6779	
Service Info							Detail	192.168.8.111:5779	
Host ⑦ Port	t 🕐 Si	tatus 🕐	Git Info Sha 🕐	Leader Count ⑦	Partition Distribution	? Leade Distri	er ⑦ bution	Meta Service	
192.168.8.111 977	'9 0	NLINE		3	basketballpl 0	ayer:1 baske	tballplayer:3	Service 192.168.8.111:9559	Version

STORAGE LEADER DISTRIBUTION

In this section, the number of Leaders and the Leader distribution will be shown.

- Click the **Balance Leader** button in the upper right corner to distribute Leaders evenly and quickly in the NebulaGraph cluster. For details about the Leader, see Storage Service.
- Click **Detail** in the upper right corner to view the details of the Leader distribution.

#### VERSION

In this section, the version and host information of each NebulaGraph service will be shown. Click **Detail** in the upper right corner to view the details of the version and host information.

#### SERVICE INFORMATION

In this section, the information on Storage services will be shown. The parameter description is as follows:

Parameter	Description
Host	The IP address of the host.
Port	The port of the host.
Status	The host status.
Git Info Sha	The commit ID of the current version.
Leader Count	The number of Leaders.
Partition Distribution	The distribution of partitions.
Leader Distribution	The distribution of Leaders.

Click **Detail** in the upper right corner to view the details of the Storage service information.

PARTITION DISTRIBUTION

Select the specified graph space in the upper left corner, you can view the distribution of partitions in the specified graph space. You can see the IP addresses and ports of all Storage services in the cluster, and the number of partitions in each Storage service. Click **Detail** in the upper right corner to view more details.

#### PARTITION INFORMATION

In this section, the information on partitions will be shown. Before viewing the partition information, you need to select a graph space in the upper left corner. The parameter description is as follows:

Parameter	Description
Partition ID	The ID of the partition.
Leader	The IP address and port of the leader.
Peers	The IP addresses and ports of all the replicas.
Losts	The IP addresses and ports of faulty replicas.

Click **Detail** in the upper right corner to view details. You can also enter the partition ID into the input box in the upper right corner of the details page to filter the shown data.

#### Config

It shows the configuration of the NebulaGraph service. NebulaGraph Dashboard Community Edition does not support online modification of configurations for now.

#### 16.4.5 Others

In the lower left corner of the page, you can:

- Sign out
- Switch between Chinese and English
- View the current Dashboard release
- View the user manual and forum
- Fold the sidebar

# 16.5 Metrics

This topic will describe the monitoring metrics in NebulaGraph Dashboard.

# 16.5.1 Machine

Q.
Note
All the machine metrics listed below are for the Linux operating system.
The default unit in <b>Disk</b> and <b>Network</b> is byte. The unit will change with the data magnitude as the page displays. For example, when the flow is less than 1 KB/s, the unit will be Bytes/s.

• For versions of Dashboard Community Edition greater than v1.0.2, the memory occupied by Buff and Cache will not be counted in the memory usage.

#### CPU

Parameter	Description
cpu_utilization	The percentage of used CPU.
cpu_idle	The percentage of idled CPU.
cpu_wait	The percentage of CPU waiting for IO operations.
cpu_user	The percentage of CPU used by users.
cpu_system	The percentage of CPU used by the system.

#### Memory

Parameter	Description
memory_utilization	The percentage of used memory.
memory_used	The memory space used (not including caches).
memory_free	The memory space available.

#### Load

Parameter	Description
load_1m	The average load of the system in the last 1 minute.
Load_5m	The average load of the system in the last 5 minutes.
load_15m	The average load of the system in the last 15 minutes.

#### Disk

Parameter	Description
disk_used_percentage	The disk utilization percentage.
disk_used	The disk space used.
disk_free	The disk space available.
disk_readbytes	The number of bytes that the system reads in the disk per second.
disk_writebytes	The number of bytes that the system writes in the disk per second.
disk_readiops	The number of read queries that the disk receives per second.
disk_writeiops	The number of write queries that the disk receives per second.
inode_utilization	The percentage of used inode.

#### Network

Parameter	Description
network_in_rate	The number of bytes that the network card receives per second.
network_out_rate	The number of bytes that the network card sends out per second.
network_in_errs	The number of wrong bytes that the network card receives per second.
network_out_errs	The number of wrong bytes that the network card sends out per second.
network_in_packets	The number of data packages that the network card receives per second.
network_out_packets	The number of data packages that the network card sends out per second.

#### 16.5.2 Service

#### Period

The period is the time range of counting metrics. It currently supports 5 seconds, 60 seconds, 600 seconds, and 3600 seconds, which respectively represent the last 5 seconds, the last 1 minute, the last 10 minutes, and the last 1 hour.

#### Metric methods

Parameter	Description
rate	The average rate of operations per second in a period.
sum	The sum of operations in the period.
avg	The average latency in the cycle.
P75	The 75th percentile latency.
P95	The 95th percentile latency.
P99	The 99th percentile latency.
P999	The 99.9th percentile latency.

#### Q Note

Dashboard collects the following metrics from the NebulaGraph core, but only shows the metrics that are important to it.

Graph

Parameter	Description		
num_active_queries	The number of changes in the number of active queries. Formula: The number of started queries minus the number of finished queries within a specified time.		
num_active_sessions	The number of changes in the number of active sessions. Formula: The number of logged in sessions minus the number of logged out sessions within a specified time. For example, when querying <code>num_active_sessions.sum.5</code> , if there were 10 sessions logged in and 30 sessions logged out in the last 5 seconds, the value of this metric is -20 (10-30).		
<pre>num_aggregate_executors</pre>	The number of executions for the Aggregation operator.		
<pre>num_auth_failed_sessions_bad_username_password</pre>	The number of sessions where authentication failed due to incorrect username and password.		
<pre>num_auth_failed_sessions_out_of_max_allowed</pre>	The number of sessions that failed to authenticate logins because the value of the parameter $\mbox{FLAG_OUT_OF_MAX_ALLOWED_CONNECTIONS}$ was exceeded.		
num_auth_failed_sessions	The number of sessions in which login authentication failed.		
<pre>num_indexscan_executors</pre>	The number of executions for index scan operators.		
num_killed_queries	The number of killed queries.		
num_opened_sessions	The number of sessions connected to the server.		
num_queries	The number of queries.		
<pre>num_query_errors_leader_changes</pre>	The number of the raft leader changes due to query errors.		
num_query_errors	The number of query errors.		
num_reclaimed_expired_sessions	The number of expired sessions actively reclaimed by the server.		
num_rpc_sent_to_metad_failed	The number of failed RPC requests that the Graphd service sent to the Metad service.		
<pre>num_rpc_sent_to_metad</pre>	The number of RPC requests that the Graphd service sent to the Metad service.		
num_rpc_sent_to_storaged_failed	The number of failed RPC requests that the Graphd service sent to the Storaged service.		
<pre>num_rpc_sent_to_storaged</pre>	The number of RPC requests that the Graphd service sent to the Storaged service.		
num_sentences	The number of statements received by the Graphd service.		
num_slow_queries	The number of slow queries.		
<pre>num_sort_executors</pre>	The number of executions for the Sort operator.		
optimizer_latency_us	The latency of executing optimizer statements.		
query_latency_us	The latency of queries.		
<pre>slow_query_latency_us</pre>	The latency of slow queries.		
<pre>num_queries_hit_memory_watermark</pre>	The number of queries reached the memory watermark.		
<pre>resp_part_completeness</pre>	The completeness of the partial success. You need to set <code>accept_partial_success</code> to true in the graph configuration first.		

### Meta

Parameter	Description
<pre>commit_log_latency_us</pre>	The latency of committing logs in Raft.
<pre>commit_snapshot_latency_us</pre>	The latency of committing snapshots in Raft.
heartbeat_latency_us	The latency of heartbeats.
num_heartbeats	The number of heartbeats.
num_raft_votes	The number of votes in Raft.
transfer_leader_latency_us	The latency of transferring the raft leader.
<pre>num_agent_heartbeats</pre>	The number of heartbeats for the AgentHBProcessor.
agent_heartbeat_latency_us	The latency of the AgentHBProcessor.
replicate_log_latency_us	The latency of replicating the log record to most nodes by Raft.
num_send_snapshot	The number of times that Raft sends snapshots to other nodes.
append_log_latency_us	The latency of replicating the log record to a single node by Raft.
append_wal_latency_us	The Raft write latency for a single WAL.
num_grant_votes	The number of times that Raft votes for other nodes.
num_start_elect	The number of times that Raft starts an election.

Storage

Parameter	Description
add_edges_latency_us	The latency of adding edges.
add_vertices_latency_us	The latency of adding vertices.
<pre>commit_log_latency_us</pre>	The latency of committing logs in Raft.
<pre>commit_snapshot_latency_us</pre>	The latency of committing snapshots in Raft.
delete_edges_latency_us	The latency of deleting edges.
delete_vertices_latency_us	The latency of deleting vertices.
<pre>get_neighbors_latency_us</pre>	The latency of querying neighbor vertices.
<pre>get_dst_by_src_latency_us</pre>	The latency of querying the destination vertex by the source vertex.
num_get_prop	The number of executions for the GetPropProcessor.
num_get_neighbors_errors	The number of execution errors for the GetNeighborsProcessor.
<pre>num_get_dst_by_src_errors</pre>	The number of execution errors for the GetDstBySrcProcessor.
<pre>get_prop_latency_us</pre>	The latency of executions for the GetPropProcessor.
num_edges_deleted	The number of deleted edges.
num_edges_inserted	The number of inserted edges.
num_raft_votes	The number of votes in Raft.
<pre>num_rpc_sent_to_metad_failed</pre>	The number of failed RPC requests that the Storage service sent to the Meta service.
<pre>num_rpc_sent_to_metad</pre>	The number of RPC requests that the Storaged service sent to the Metad service.
num_tags_deleted	The number of deleted tags.
<pre>num_vertices_deleted</pre>	The number of deleted vertices.
<pre>num_vertices_inserted</pre>	The number of inserted vertices.
<pre>transfer_leader_latency_us</pre>	The latency of transferring the raft leader.
<pre>lookup_latency_us</pre>	The latency of executions for the LookupProcessor.
num_lookup_errors	The number of execution errors for the LookupProcessor.
num_scan_vertex	The number of executions for the ScanVertexProcessor.
<pre>num_scan_vertex_errors</pre>	The number of execution errors for the ScanVertexProcessor.
update_edge_latency_us	The latency of executions for the UpdateEdgeProcessor.
<pre>num_update_vertex</pre>	The number of executions for the UpdateVertexProcessor.
<pre>num_update_vertex_errors</pre>	The number of execution errors for the UpdateVertexProcessor.
kv_get_latency_us	The latency of executions for the Getprocessor.
<pre>kv_put_latency_us</pre>	The latency of executions for the PutProcessor.
kv_remove_latency_us	The latency of executions for the RemoveProcessor.
num_kv_get_errors	The number of execution errors for the GetProcessor.
num_kv_get	The number of executions for the GetProcessor.
<pre>num_kv_put_errors</pre>	The number of execution errors for the PutProcessor.
num_kv_put	The number of executions for the PutProcessor.

Parameter	Description
<pre>num_kv_remove_errors</pre>	The number of execution errors for the RemoveProcessor.
num_kv_remove	The number of executions for the RemoveProcessor.
<pre>forward_tranx_latency_us</pre>	The latency of transmission.
<pre>scan_edge_latency_us</pre>	The latency of executions for the ScanEdgeProcessor.
<pre>num_scan_edge_errors</pre>	The number of execution errors for the ScanEdgeProcessor.
num_scan_edge	The number of executions for the ScanEdgeProcessor.
<pre>scan_vertex_latency_us</pre>	The latency of executions for the ScanVertexProcessor.
num_add_edges	The number of times that edges are added.
<pre>num_add_edges_errors</pre>	The number of errors when adding edges.
num_add_vertices	The number of times that vertices are added.
<pre>num_start_elect</pre>	The number of times that Raft starts an election.
<pre>num_add_vertices_errors</pre>	The number of errors when adding vertices.
<pre>num_delete_vertices_errors</pre>	The number of errors when deleting vertices.
append_log_latency_us	The latency of replicating the log record to a single node by Raft.
num_grant_votes	The number of times that Raft votes for other nodes.
<pre>replicate_log_latency_us</pre>	The latency of replicating the log record to most nodes by Raft.
num_delete_tags	The number of times that tags are deleted.
<pre>num_delete_tags_errors</pre>	The number of errors when deleting tags.
<pre>num_delete_edges</pre>	The number of edge deletions.
<pre>num_delete_edges_errors</pre>	The number of errors when deleting edges
num_send_snapshot	The number of times that snapshots are sent.
update_vertex_latency_us	The latency of executions for the UpdateVertexProcessor.
append_wal_latency_us	The Raft write latency for a single WAL.
num_update_edge	The number of executions for the UpdateEdgeProcessor.
delete_tags_latency_us	The latency of deleting tags.
<pre>num_update_edge_errors</pre>	The number of execution errors for the UpdateEdgeProcessor.
num_get_neighbors	The number of executions for the GetNeighborsProcessor.
<pre>num_get_dst_by_src</pre>	The number of executions for the GetDstBySrcProcessor.
<pre>num_get_prop_errors</pre>	The number of execution errors for the GetPropProcessor.
num_delete_vertices	The number of times that vertices are deleted.
num_lookup	The number of executions for the LookupProcessor.
num_sync_data	The number of times the Storage service synchronizes data from the Drainer.
num_sync_data_errors	The number of errors that occur when the Storage service synchronizes data from the Drainer.
<pre>sync_data_latency_us</pre>	The latency of the Storage service synchronizing data from the Drainer.

### Graph space

#### Q Note

Space-level metrics are created dynamically, so that only when the behavior is triggered in the graph space, the corresponding metric is created and can be queried by the user.

Parameter	Description
num_active_queries	The number of queries currently being executed.
num_queries	The number of queries.
num_sentences	The number of statements received by the Graphd service.
optimizer_latency_us	The latency of executing optimizer statements.
query_latency_us	The latency of queries.
num_slow_queries	The number of slow queries.
num_query_errors	The number of query errors.
<pre>num_query_errors_leader_changes</pre>	The number of raft leader changes due to query errors.
num_killed_queries	The number of killed queries.
<pre>num_aggregate_executors</pre>	The number of executions for the Aggregation operator.
num_sort_executors	The number of executions for the Sort operator.
num_indexscan_executors	The number of executions for index scan operators.
<pre>num_auth_failed_sessions_bad_username_password</pre>	The number of sessions where authentication failed due to incorrect username and password.
num_auth_failed_sessions	The number of sessions in which login authentication failed.
num_opened_sessions	The number of sessions connected to the server.
<pre>num_queries_hit_memory_watermark</pre>	The number of queries reached the memory watermark.
<pre>num_reclaimed_expired_sessions</pre>	The number of expired sessions actively reclaimed by the server.
num_rpc_sent_to_metad_failed	The number of failed RPC requests that the Graphd service sent to the Metad service.
<pre>num_rpc_sent_to_metad</pre>	The number of RPC requests that the Graphd service sent to the Metad service.
<pre>num_rpc_sent_to_storaged_failed</pre>	The number of failed RPC requests that the Graphd service sent to the Storaged service.
num_rpc_sent_to_storaged	The number of RPC requests that the Graphd service sent to the Storaged service.
<pre>slow_query_latency_us</pre>	The latency of slow queries.

# 17. NebulaGraph Operator

# 17.1 What is NebulaGraph Operator

### 17.1.1 Concept

NebulaGraph Operator is a tool to automate the deployment, operation, and maintenance of NebulaGraph clusters on Kubernetes. Building upon the excellent scalability mechanism of Kubernetes, NebulaGraph introduced its operation and maintenance knowledge into the Kubernetes system, which makes NebulaGraph a real cloud-native graph database.



#### 17.1.2 How it works

For resource types that do not exist within Kubernetes, you can register them by adding custom API objects. The common way is to use the CustomResourceDefinition.

NebulaGraph Operator abstracts the deployment management of NebulaGraph clusters as a CRD. By combining multiple built-in API objects including StatefulSet, Service, and ConfigMap, the routine management and maintenance of a NebulaGraph cluster are coded as a control loop in the Kubernetes system. When a CR instance is submitted, NebulaGraph Operator drives database clusters to the final state according to the control process.

#### 17.1.3 Features

The following features are already available in NebulaGraph Operator:

- **Cluster deployment and deletion**: NebulaGraph Operator simplifies the process of deploying and uninstalling clusters for users. NebulaGraph Operator allows you to quickly create, update, or delete a NebulaGraph cluster by simply providing the corresponding CR file. For more information, see Install NebulaGraph Clusters.
- Cluster Upgrade: NebulaGraph Operator supports cluster upgrading from version 3.5.0 to version 3.6.0.
- **Self-Healing**: NebulaGraph Operator calls interfaces provided by NebulaGraph clusters to dynamically sense cluster service status. Once an exception is detected, NebulaGraph Operator performs fault tolerance. For more information, see Self-Healing.
- **Balance Scheduling**: Based on the scheduler extension interface, the scheduler provided by NebulaGraph Operator evenly distributes Pods in a NebulaGraph cluster across all nodes.

#### 17.1.4 Limitations

#### Version limitations

NebulaGraph Operator does not support the v1.x version of NebulaGraph. NebulaGraph Operator version and the corresponding NebulaGraph version are as follows:

NebulaGraph	NebulaGraph Operator
3.5.x ~ 3.6.0	1.5.0 ~ 1.7.x, 1.8.0
3.0.0 ~ 3.4.1	1.3.0, 1.4.0 ~ 1.4.2
3.0.0 ~ 3.3.x	1.0.0, 1.1.0, 1.2.0
2.5.x ~ 2.6.x	0.9.0
2.5.x	0.8.0

# L Jacy version compatibility

• The 1.x version NebulaGraph Operator is not compatible with NebulaGraph of version below v3.x.

• Starting from NebulaGraph Operator 0.9.0, logs and data are stored separately. Using NebulaGraph Operator 0.9.0 or later versions to manage a NebulaGraph 2.5.x cluster created with Operator 0.8.0 can cause compatibility issues. You can backup the data of the NebulaGraph 2.5.x cluster and then create a 2.6.x cluster with Operator 0.9.0.

#### 17.1.5 Release note

Release

Last update: April 28, 2024

# 17.2 Getting started

#### 17.2.1 Install NebulaGraph Operator

You can deploy NebulaGraph Operator with Helm.

#### Background

NebulaGraph Operator automates the management of NebulaGraph clusters, and eliminates the need for you to install, scale, upgrade, and uninstall NebulaGraph clusters, which lightens the burden on managing different application versions.

#### Prerequisites

Before installing NebulaGraph Operator, you need to install the following software and ensure the correct version of the software :

Software	Requirement
Kubernetes	>= 1.18
Helm	>= 3.2.0
CoreDNS	>= 1.6.0

Q Note

• If using a role-based access control policy, you need to enable RBAC (optional).

• CoreDNS is a flexible and scalable DNS server that is installed for Pods in NebulaGraph clusters.

#### Steps

#### 1. Add the NebulaGraph Operator Helm repository.

helm repo add nebula-operator https://vesoft-inc.github.io/nebula-operator/charts

2. Update information of available charts locally from repositories.

helm repo update

For more information about helm repo, see Helm Repo.

3. Create a namespace for NebulaGraph Operator.

kubectl create namespace <namespace\_name>

For example, run the following command to create a namespace named nebula-operator-system.

kubectl create namespace nebula-operator-system

All the resources of NebulaGraph Operator are deployed in this namespace.

#### 4. Install NebulaGraph Operator.

helm install nebula-operator nebula-operator/nebula-operator --namespace=<namespace\_name> --version=\${chart\_version}

For example, the command to install NebulaGraph Operator of version 1.8.0 is as follows.

helm install nebula-operator nebula-operator/nebula-operator --namespace=nebula-operator-system --version=1.8.0
1.8.0 is the version of the nebula-operator chart. When not specifying --version, the latest version of the nebula-operator chart is used by default.

Run helm search repo -l nebula-operator to see chart versions.

You can customize the configuration items of the NebulaGraph Operator chart before running the installation command. For more information, see Customize installation defaults.

# 5. View the information about the default-created CRD.

kubectl get crd

Output:

NAME	CREATED AT
nebulaautoscalers.autoscaling.nebula-graph.io	2023-11-01T04:16:51Z
nebulaclusters.apps.nebula-graph.io	2023-10-12T07:55:32Z
nebularestores.apps.nebula-graph.io	2023-02-04T23:01:00Z

# What's next

Create a NebulaGraph cluster

Last update: November 15, 2023

#### 17.2.2 Create a NebulaGraph cluster

This topic introduces how to create a NebulaGraph cluster with the following two methods:

- Create a NebulaGraph cluster with Helm
- Create a NebulaGraph cluster with Kubectl

#### Prerequisites

- NebulaGraph Operator is installed.
- A StorageClass is created.

#### Create a NebulaGraph cluster with Helm

L jacy version compatibility

The 1.x version NebulaGraph Operator is not compatible with NebulaGraph of version below v3.x.

#### 1. Add the NebulaGraph Operator Helm repository.

helm repo add nebula-operator https://vesoft-inc.github.io/nebula-operator/charts

#### 2. Update information of available charts locally from chart repositories.

helm repo update

#### 3. Set environment variables to your desired values.

 export NEBULA\_CLUSTER\_NAME=nebula
 # The desired NebulaGraph cluster name.

 export NEBULA\_CLUSTER\_NAMESPACE=nebula
 # The desired namespace where your NebulaGraph cluster locates.

 export STORAGE\_CLASS\_NAME=fast-disks
 # The name of the StorageClass that has been created.

#### 4. Create a namespace for your NebulaGraph cluster (If you have created one, skip this step).

kubectl create namespace "\${NEBULA\_CLUSTER\_NAMESPACE}"

#### 5. Apply the variables to the Helm chart to create a NebulaGraph cluster.

helm install "\${NEBULA\_CLUSTER\_NAME}" nebula-operator/nebula-cluster \
 --set nameOverride="\${NEBULA\_CLUSTER\_NAME}" \
 --set nebula.storageClassName="\${STORAGE\_CLASS\_NAME}" \
 # Specify the version of the NebulaGraph cluster.
 --set nebula.version=v3.6.0 \
 # Specify the version of the nebula-cluster chart. If not specified, the latest version of the chart is installed by default.
 # Run 'helm search repo nebula-operator/nebula-cluster' to view the available versions of the chart.
 --version=1.8.0 \
 --namespace="\${HEBULA\_CLUSTER\_NAMESPACE}" \

#### Create a NebulaGraph cluster with Kubectl

₽ Jacy version compatibility

The 1.x version NebulaGraph Operator is not compatible with NebulaGraph of version below v3.x.

 $The following \ example \ shows \ how \ to \ create \ a \ Nebula Graph \ cluster \ by \ creating \ a \ cluster \ named \ nebula \ .$ 

1. Create a namespace, for example, nebula. If not specified, the default namespace is used.

kubectl create namespace nebula

2. Define the cluster configuration file nebulacluster.yaml.

```
\mathbf{k}pand to see an example configuration for the cluster 	imes
apiVersion: apps.nebula-graph.io/vlalpha1
kind: NebulaCluster
metadata:
  name: nebula
  namespace: default
spec:
  topologySpreadConstraints:

    topologyKey: "kubernetes.io/hostname"
whenUnsatisfiable: "ScheduleAnyway"

  graphd:
    # Container image for the Graph service.
    image: vesoft/nebula-graphd
logVolumeClaim:
       resources:
        requests:
       storage: 2Gi
# Storage class name for storing Graph service logs.
       storageClassName: local-sc
     replicas: 1
     resources:
       Limits:
cpu: "1"
         memory: 1Gi
       requests:
        cpu: 500m
memory: 500Mi
  version: v3.6.0
imagePullPolicy: Always
  metad:
    # Container image for the Meta service.
    image: vesoft/nebula-metad
logVolumeClaim:
       resources:
        requests:
       storage: 2Gi
storageClassName: local-sc
    dataVolumeClaim:
       resources:
         requests:
            storage: 2Gi
       storageClassName: local-sc
     replicas: 1
     resources:
       limits:
cpu: "1"
         memory: 1Gi
       requests:
         cpu: 500m
memory: 500Mi
  version: v3.6.0
reference:
  name: statefulsets.apps
version: v1
schedulerName: default-scheduler
  storaged:
    # Container image for the Storage service.
image: vesoft/nebula-storaged
logVolumeClaim:
       resources:
         requests:
       storageClassName: local-sc
    dataVolumeClaims:
     - resources:
        requests:
storage: 2Gi
     storageClassName: local-sc
replicas: 1
     resources:
limits:
         cpu: "1"
memory: 1Gi
       requests:
         cpu: 500m
         memory: 500Mi
     version: v3.6.0
```

For more information about the other parameters, see Install NebulaGraph clusters.

# 3. Create a NebulaGraph cluster.

kubectl create -f nebulacluster.yaml

# Output:

nebulacluster.apps.nebula-graph.io/nebula created

# 4. Check the status of the NebulaGraph cluster.

kubectl get nc nebula

# Output:

NAME	READY	GRAPHD-DESIRED	GRAPHD-READY	METAD-DESIRED	METAD-READY	STORAGED-DESIRED	STORAGED-READY	AGE
nebula	True	1	1	1	1	1	1	86s

# What's next

Connect to a cluster

Last update: November 15, 2023

# 17.2.3 Connect to a NebulaGraph cluster

After creating a NebulaGraph cluster with NebulaGraph Operator on Kubernetes, you can connect to NebulaGraph databases from within the cluster and outside the cluster.

#### Prerequisites

A NebulaGraph cluster is created on Kubernetes. For more information, see Create a NebulaGraph cluster.

# Connect to NebulaGraph databases from within a NebulaGraph cluster

You can create a ClusterIP type Service to provide an access point to the NebulaGraph database for other Pods within the cluster. By using the Service's IP and the Graph service's port number (9669), you can connect to the NebulaGraph database. For more information, see ClusterIP.

- 1. Create a file named graphd-clusterip-service.yaml. The file contents are as follows:
  - apiVersion: v1 kind: Service metadata: labels app.kubernetes.io/cluster: nebula app.kubernetes.io/component: graphd app.kubernetes.io/managed-by: nebula-operator app.kubernetes.io/name: nebula-graph name: nebula-graphd-svc namespace: default spec: ports: - name: thrift port: 9669 protocol: TCP targetPort: 9669 name: http port: 19669 protocol: TCP targetPort: 19669 selector app.kubernetes.io/cluster: nebula app.kubernetes.io/component: graphd app.kubernetes.io/managed-by: nebula-operator app.kubernetes.io/name: nebula-graph type: ClusterIP # Set the type to ClusterIP.
- NebulaGraph uses port 9669 by default. 19669 is the HTTP port of the Graph service in a NebulaGraph cluster.
- targetPort is the port mapped to the database Pods, which can be customized.
- 2. Create a ClusterIP Service.

kubectl create -f graphd-clusterip-service.yaml

3. Check the IP of the Service:

\$ kubectl get service -l	app.kubernete	s.io/cluster= <ne< th=""><th>ebula&gt; # <neb< th=""><th>ula&gt; is the name of your NebulaGraph cluster.</th><th></th></neb<></th></ne<>	ebula> # <neb< th=""><th>ula&gt; is the name of your NebulaGraph cluster.</th><th></th></neb<>	ula> is the name of your NebulaGraph cluster.	
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nebula-graphd-svc	ClusterIP	10.98.213.34	<none></none>	9669/TCP,19669/TCP,19670/TCP	23h

4. Run the following command to connect to the NebulaGraph database using the IP of the <cluster-name>-graphd-svc Service above:

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- <nebula\_console\_name> -addr <cluster\_ip> -port <service\_port> -u <username> -p cpassword>

#### For example:

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- nebula-console -addr 10.98.213.34 -port 9669 -u root -p vesoft

- -- image : The image for the tool NebulaGraph Console used to connect to NebulaGraph databases.
- <nebula-console> : The custom Pod name.
- -addr : The IP of the ClusterIP Service, used to connect to Graphd services.
- -port : The port to connect to Graphd services, the default port of which is 9669.
- -u: The username of your NebulaGraph account. Before enabling authentication, you can use any existing username. The default username is root.
- 📭 : The password of your NebulaGraph account. Before enabling authentication, you can use any characters as the password.
- A successful connection to the database is indicated if the following is returned:

If you don't see a command prompt, try pressing enter.
(root@nebula) [(none)]>

You can also connect to NebulaGraph databases with **Fully Qualified Domain Name (FQDN)**. The domain format is <cluster-name>-graphd.<cluster-namespace>.svc.<CLUSTER\_DOMAIN>. The default value of CLUSTER\_DOMAIN is cluster.local.

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- <nebula\_console\_name> -addr <cluster\_name>-graphd-svc.default.svc.cluster.local -port <service\_port> -u <username> -p <password>

service\_port is the port to connect to Graphd services, the default port of which is 9669.

# Finte If the spec.console field is set in the cluster configuration file, you can also connect to NebulaGraph databases with the following command: # Enter the nebula-console Pod. kubectl exec -it nebula-console -- /bin/sh # Connect to NebulaGraph databases. nebula-console -addr nebula-graphd-svc.default.svc.cluster.local -port 9669 -u <username> -p <password>

For information about the nebula-console container, see nebula-console.

#### Connect to NebulaGraph databases from outside a NebulaGraph cluster via NodePort

You can create a NodePort type Service to access internal cluster services from outside the cluster using any node IP and the exposed node port. You can also utilize load balancing services provided by cloud vendors (such as Azure, AWS, etc.) by setting the Service type to LoadBalancer. This allows external access to internal cluster services through the public IP and port of the load balancer provided by the cloud vendor.

The Service of type NodePort forwards the front-end requests via the label selector spec.selector to Graphd pods with labels app.kubernetes.io/cluster: <cluster-name> and app.kubernetes.io/component: graphd.

After creating a NebulaGraph cluster based on the example template, where <code>spec.graphd.service.type=NodePort</code>, the NebulaGraph Operator will automatically create a NodePort type Service named <code><cluster-name>-graphd-svc</code> in the same namespace. You can directly connect to the NebulaGraph database through any node IP and the exposed node port (see step 4 below). You can also create a custom Service according to your needs.

Steps:

- 1. Create a YAML file named graphd-nodeport-service.yaml. The file contents are as follows:
  - apiVersion: v1 kind: Service metadata: labels app.kubernetes.io/cluster: nebula app.kubernetes.io/component: graphd app.kubernetes.io/managed-by: nebula-operator app.kubernetes.io/name: nebula-graph name: nebula-graphd-svc-nodeport namespace: default spec: externalTrafficPolicy: Local ports: name: thrift port: 9669 protocol: TCP targetPort: 9669 name: http port: 19669 protocol: TCP targetPort: 19669 selecto app.kubernetes.io/cluster: nebula app.kubernetes.io/component: graphd app.kubernetes.io/managed-by: nebula-operator app.kubernetes.io/name: nebula-graph
    type: NodePort # Set the type to NodePort
- NebulaGraph uses port 9669 by default. 19669 is the HTTP port of the Graph service in a NebulaGraph cluster.
- The value of targetPort is the port mapped to the database Pods, which can be customized.
- 2. Run the following command to create a NodePort Service.

kubectl create -f graphd-nodeport-service.yaml

3. Check the port mapped on all of your cluster nodes.

kubectl get services -l app.kubernetes.io/cluster=<nebula> # <nebula> is the name of your NebulaGraph cluster.

#### Output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nebula-graphd-svc-nodeport	NodePort	10.107.153.129	<none></none>	9669:32236/TCP,19669:31674/TCP,19670:31057/TCP	24h

As you see, the mapped port of NebulaGraph databases on all cluster nodes is 32236.

4. Connect to NebulaGraph databases with your node IP and the node port above.

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- <nebula\_console\_name> -addr <node\_ip> -port <node\_port> -u <username> -p <password>

#### For example:

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- nebula-console -addr 192.168.8.24 -port 32236 -u root -p vesoft
If you don't see a command prompt, try pressing enter.

(root@nebula) [(none)]>

- --image : The image for the tool NebulaGraph Console used to connect to NebulaGraph databases.
- <nebula-console> : The custom Pod name. The above example uses nebula-console .
- -addr : The IP of any node in a NebulaGraph cluster. The above example uses 192.168.8.24.
- -port : The mapped port of NebulaGraph databases on all cluster nodes. The above example uses 32236 .
- -u: The username of your NebulaGraph account. Before enabling authentication, you can use any existing username. The default username is root.
- -p: The password of your NebulaGraph account. Before enabling authentication, you can use any characters as the password.

#### Q Note

If the spec.console field is set in the cluster configuration file, you can also connect to NebulaGraph databases with the following command:

# Enter the nebula-console Pod. kubectl exec -it nebula-console /bin/sh	
<pre># Connect to NebulaGraph databases. nebula-console -addr <node ip=""> -port <node port=""> -u <username> -p <pre>password</pre></username></node></node></pre>	>

For information about the nebula-console container, see nebula-console.

#### Connect to NebulaGraph databases from outside a NebulaGraph cluster via Ingress

When dealing with multiple pods in a cluster, managing services for each pod separately is not a good practice. Ingress is a Kubernetes resource that provides a unified entry point for accessing multiple services. Ingress can be used to expose multiple services under a single IP address.

Nginx Ingress is an implementation of Kubernetes Ingress. Nginx Ingress watches the Ingress resource of a Kubernetes cluster and generates the Ingress rules into Nginx configurations that enable Nginx to forward 7 layers of traffic.

You can use Nginx Ingress to connect to a NebulaGraph cluster from outside the cluster using a combination of the host network and DaemonSet pattern.

Due to the use of HostNetwork, Nginx Ingress pods may be scheduled on the same node (port conflicts will occur when multiple pods try to listen on the same port on the same node). To avoid this situation, Nginx Ingress is deployed on these nodes in DaemonSet mode (ensuring that a pod replica runs on each node in the cluster). You first need to select some nodes and label them for the specific deployment of Nginx Ingress.

Ingress does not support TCP or UDP services. For this reason, the nginx-ingress-controller pod uses the flags --tcp-servicesconfigmap and --udp-services-configmap to point to an existing ConfigMap where the key refers to the external port to be used and the value refers to the format of the service to be exposed. The format of the value is <namespace/service\_name>:<service\_port>.

For example, the configurations of the ConfigMap named as tcp-services is as follows:

apiVersion: v1 kind: ConfigMap metadata: name: tcp-services namespace: nginx-ingress data: # update 9769: "default/nebula-graphd-svc:9669" Steps are as follows.

1. Create a file named nginx-ingress-daemonset-hostnetwork.yaml.

Click on nginx-ingress-daemonset-hostnetwork.yaml to view the complete content of the example YAML file.

# Note

The resource objects in the YAML file above use the namespace <code>nginx-ingress</code>. You can run <code>kubectl create namespace nginx-ingress</code> to create this namespace, or you can customize the namespace.

2. Label a node where the DaemonSet named nginx-ingress-controller in the above YAML file (The node used in this example is named worker2 with an IP of 192.168.8.160) runs.

kubectl label node worker2 nginx-ingress=true

3. Run the following command to enable Nginx Ingress in the cluster you created.

kubectl create -f nginx-ingress-daemonset-hostnetwork.yaml

#### Output:

configmap/nginx-ingress-controller created configmap/tcp-services created serviceaccount/nginx-ingress created serviceaccount/nginx-ingress-backend created clusterrole.rbac.authorization.k8s.io/nginx-ingress created role.rbac.authorization.k8s.io/nginx-ingress created role.rbac.authorization.k8s.io/nginx-ingress created service/nginx-ingress-controller-metrics created service/nginx-ingress-controller-metrics created service/nginx-ingress-proxy-tcp created daemonset.apps/nginx-ingress-controller created

Since the network type that is configured in Nginx Ingress is hostNetwork, after successfully deploying Nginx Ingress, with the IP (192.168.8.160) of the node where Nginx Ingress is deployed and with the external port (9769) you define, you can access NebulaGraph.

4. Use the IP address and the port configured in the preceding steps. You can connect to NebulaGraph with NebulaGraph Console.

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- <nebula\_console\_name> -addr <host\_ip> -port <external\_port> -u <username> -p <password>

#### Output:

kubectl run -ti --image vesoft/nebula-console:v3.6.0 --restart=Never -- nebula-console -addr 192.168.8.160 -port 9769 -u root -p vesoft

- --- image : The image for the tool NebulaGraph Console used to connect to NebulaGraph databases.
- <nebula-console> The custom Pod name. The above example uses nebula-console.
- -addr : The IP of the node where Nginx Ingress is deployed. The above example uses 192.168.8.160.
- -port : The port used for external network access. The above example uses 9769.
- -u: The username of your NebulaGraph account. Before enabling authentication, you can use any existing username. The default username is root.
- -p: The password of your NebulaGraph account. Before enabling authentication, you can use any characters as the password.

A successful connection to the database is indicated if the following is returned:

If you don't see a command prompt, try pressing enter.
(root@nebula) [(none)]>

#### Q Note

If the spec.console field is set in the cluster configuration file, you can also connect to NebulaGraph databases with the following command:

# Enter the nebula-console Pod. kubectl exec -it nebula-console -- /bin/sh # Connect to NebulaGraph databases. nebula-console -addr <ingress\_host\_ip> -port <external\_port> -u <username> -p <password>

For information about the nebula-console container, see nebula-console.

Last update: November 21, 2023

# 17.3 NebulaGraph Operator management

#### 17.3.1 Customize installation defaults

This topic introduces how to customize the default configurations when installing NebulaGraph Operator.

#### **Customizable parameters**

When executing the helm install [NAME] [CHART] [flags] command to install a chart, you can specify the chart configuration. For more information, see Customizing the Chart Before Installing.

You can view the configurable options in the nebula-operator chart configuration file. Alternatively, you can view the configurable options through the command helm show values nebula-operator/nebula-operator, as shown below.



```
# The TCP port the Webhook server binds to. (default 9443)
webhookBindPort: 9443
scheduler:
    create: true
    schedulerName: nebula-scheduler
    replicas: 2
    env: [ ]
    resources:
    limits:
        cpu: 200m
        memory: 200Mi
    requests:
        cpu: 100m
        memory: 100Wi
verbosity: 0
plugins:
    enabled: ["NodeZone"]
    disabled: [] # Only in-tree plugins need to be defined here
```

# Part of the above parameters are described as follows:

Parameter	Default value	Description
image.nebulaOperator.image	<pre>vesoft/nebula- operator:v1.8.0</pre>	The image of NebulaGraph Operator, version of which is 1.8.0.
<pre>image.nebulaOperator.imagePullPolicy</pre>	IfNotPresent	The image pull policy in Kubernetes.
imagePullSecrets	-	The image pull secret in Kubernetes. For example imagePullSecrets[0].name="vesoft".
kubernetesClusterDomain	cluster.local	The cluster domain.
controllerManager.create	true	Whether to enable the controller-manager component.
controllerManager.replicas	2	The number of controller-manager replicas.
controllerManager.env	[]	The environment variables for the controller-manager component.
controllerManager.extraInitContainers	[]	Runs an init container.
controllerManager.sidecarContainers	{}	Runs a sidecar container.
controllerManager.extraVolumes	[]	Sets a storage volume.
controllerManager.extraVolumeMounts	[]	Sets the storage volume mount path.
controllerManager.securityContext	0	Configures the access and control settings for NebulaGraph Operator.
admissionWebhook.create	false	Whether to enable Admission Webhook. This option is disabled. To enable it, set the value to true and you will need to install cert- manager. For details, see Enable admission control.
admissionWebhook.webhookBindPort	9443	The TCP port the Webhook server binds to. It is $9443\ by$ default.
shceduler.create	true	Whether to enable Scheduler.
shceduler.schedulerName	nebula-scheduler	The name of the scheduler customized by NebulaGraph Operator.
shceduler.replicas	2	The number of nebula-scheduler replicas.

#### Example

The following example shows how to enable AdmissionWebhook when you install NebulaGraph Operator (AdmissionWebhook is disabled by default):

helm install nebula-operator nebula-operator/nebula-operator --namespace=<nebula-operator-system> --set admissionWebhook.create=true

Check whether the specified configuration of NebulaGraph Operator is installed successfully:

helm get values nebula-operator -n <nebula-operator-system>

# Example output:

USER-SUPPLIED VALUES: admissionWebhook: create: true

For more information about helm install, see Helm Install.

Last update: March 6, 2024

# 17.3.2 Update NebulaGraph Operator

This topic introduces how to update the configuration of NebulaGraph Operator.

#### Steps

1. Update the information of available charts locally from chart repositories.

#### helm repo update

 $2. \ensuremath{\operatorname{View}}$  the default values of NebulaGraph Operator.

helm show values nebula-operator/nebula-operator

- 3. Update NebulaGraph Operator by passing configuration parameters via --set.
- --set : Overrides values using the command line. For more configurable items, see Customize installation defaults.

For example, to enable the AdmissionWebhook, run the following command:

helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=nebula-operator-system --version=1.8.0 --set admissionWebhook.create=true

For more information, see Helm upgrade.

4. Check whether the configuration of NebulaGraph Operator is updated successfully.

helm get values nebula-operator -n nebula-operator-system

Example output:

USER-SUPPLIED VALUES: admissionWebhook: create: true

Last update: November 28, 2023

# 17.3.3 Use NebulaGraph Operator to manage specific clusters

NebulaGraph Operator supports the management of multiple NebulaGraph clusters. By default, NebulaGraph Operator manages all NebulaGraph clusters. However, you can specify the clusters managed by NebulaGraph Operator. This topic describes how to specify the clusters managed by NebulaGraph Operator.

#### **Application scenarios**

- Gray release of NebulaGraph Operator: You want to run the new Nebula Operator version on a part of the clusters first to test and verify its performance before fully releasing it.
- Manage specific clusters: You want NebulaGraph Operator to manage only specific NebulaGraph clusters.

#### Configurations

NebulaGraph Operator supports specifying the clusters managed by controller-manager through startup parameters. The supported parameters are as follows:

- watchNamespaces : Specifies the namespace where the NebulaGraph cluster is located. To specify multiple namespaces, separate them with commas (,). For example, watchNamespaces=default,nebula. If this parameter is not specified, NebulaGraph Operator manages all NebulaGraph clusters in all namespaces.
- nebulaObjectSelector : Allows you to set specific labels and values to select the NebulaGraph clusters to be managed. It supports three label operation symbols: =, ==, and !=. Both = and == mean that the label's value is equal to the specified value, while != means the tag's value is not equal to the specified value. Multiple labels are separated by commas ( , ), and the comma needs to be escaped with \\. For example, nebulaObjectSelector=key1=value1\\,key2=value2, which selects only the NebulaGraph clusters with labels key1=value1 and key2=value2. If this parameter is not specified, NebulaGraph Operator manages all NebulaGraph clusters.

#### Examples

SPECIFY THE MANAGED CLUSTERS BY NAMESPACE

Run the following command to make NebulaGraph Operator manage only the NebulaGraph clusters in the default and nebula namespaces. Ensure that the current Helm Chart version supports this parameter. For more information, see Update the configuration.

helm upgrade nebula-operator nebula-operator/nebula-operator --set watchNamespaces=default,nebula

SPECIFY THE MANAGED CLUSTERS BY LABEL

Run the following command to make NebulaGraph Operator manage only the NebulaGraph clusters with the labels key1=value1 and key2=value2. Ensure that the current Helm Chart version supports this parameter. For more information, see Update the configuration.

helm upgrade nebula-operator nebula-operator/nebula-operator --set nebulaObjectSelector=key1=value1\\,key2=value2

#### FAQ

HOW TO SET LABELS FOR NEBULAGRAPH CLUSTERS?

Run the following command to set a label for the NebulaGraph cluster:

kubectl label nc <cluster\_name> -n <namespace> <key>=<value>

For example, set the label env=test for the NebulaGraph cluster named nebula in the nebulaspace namespace:

kubectl label nc nebula -n nebulaspace env=test

HOW TO VIEW THE LABELS OF NEBULAGRAPH CLUSTERS?

#### Run the following command to view the labels of NebulaGraph clusters:

kubectl get nc <cluster\_name> -n <namespace> --show-labels

For example, view the labels of the NebulaGraph cluster named nebula in the nebulaspace namespace:

kubectl get nc nebula -n nebulaspace --show-labels

HOW TO DELETE THE LABELS OF NEBULAGRAPH CLUSTERS?

Run the following command to delete the label of NebulaGraph clusters:

kubectl label nc <cluster\_name> -n <namespace> <key>-

For example, delete the label env=test of the NebulaGraph cluster named nebula in the nebulaspace namespace:

kubectl label nc nebula -n nebulaspace env-

HOW TO VIEW THE NAMESPACE WHERE THE NEBULAGRAPH CLUSTER IS LOCATED?

Run the following command to list all namespaces where the NebulaGraph clusters are located:

kubectl get nc --all-namespaces

Last update: March 6, 2024

# 17.3.4 Upgrade NebulaGraph Operator

Lyacy version compatibility
• Does not support upgrading 0.9.0 and below version NebulaGraph Operator to 1.x.
• The 1.x version NebulaGraph Operator is not compatible with NebulaGraph of version below v3.x.

## Steps

1. View the current version of NebulaGraph Operator.

helm list --all-namespaces

Example output:

NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
nebula-operator	nebula-operator-system	3	2023-11-06 12:06:24.742397418 +0800 CST	deployed	nebula-operator-1.7.0	1.7.0

2. Update the information of available charts locally from chart repositories.

helm repo update

3. View the latest version of NebulaGraph Operator.

helm search repo nebula-operator/nebula-operator

#### Example output:

 NAME
 CHART VERSION
 APP VERSION
 DESCRIPTION

 nebula-operator/nebula-operator
 1.8.0
 1.8.0
 Nebula Operator Helm chart for Kubernetes

4. Upgrade NebulaGraph Operator to version 1.8.0.

helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=<namespace\_name> --version=1.8.0

#### For example:

helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=nebula-operator-system --version=1.8.0

Output:

Release "nebula-operator" has been upgraded. Happy Helming! NAME: nebula-operator LAST DEPLOVED: Tue Apr 16 02:21:08 2022 NAMESPACE: nebula-operator-system STATUS: deployed REVISION: 3 TEST SUITE: None NOTES: NebulaGraph Operator installed!

5. Pull the latest CRD configuration file.

#### Q Note

You need to upgrade the corresponding CRD configurations after NebulaGraph Operator is upgraded. Otherwise, the creation of NebulaGraph clusters will fail. For information about the CRD configurations, see apps.nebula-graph.io\_nebulaclusters.yaml.

a. Pull the NebulaGraph Operator chart package.

helm pull nebula-operator/nebula-operator --version=1.8.0

- --version : The NebulaGraph Operator version you want to upgrade to. If not specified, the latest version will be pulled.
- b. Run tar -zxvf to unpack the charts.

For example: To unpack v1.8.0 chart to the  $\,/ {\rm tmp}\,$  path, run the following command:

tar -zxvf nebula-operator-1.8.0.tgz -C /tmp

- -C /tmp : If not specified, the chart files will be unpacked to the current directory.
- 6. Apply the latest CRD configuration file in the nebula-operator directory.

kubectl apply -f crds/nebulaclusters.yaml

#### Output:

customresourcedefinition.apiextensions.k8s.io/nebulaclusters.apps.nebula-graph.io configured

Last update: November 28, 2023

# 17.3.5 Uninstall NebulaGraph Operator

This topic introduces how to uninstall NebulaGraph Operator.

# Steps

1. Uninstall the NebulaGraph Operator chart.

helm uninstall nebula-operator --namespace=<nebula-operator-system>

2. View the information about the default-created CRD.

kubectl get crd

# Output:

NAME	CREATED AT
nebulaautoscalers.autoscaling.nebula-graph.io	2023-11-01T04:16:51Z
nebulaclusters.apps.nebula-graph.io	2023-10-12T07:55:32Z
nebularestores.apps.nebula-graph.io	2023-02-04T23:01:00Z

# 3. Delete CRD.

kubectl delete crd nebulaclusters.apps.nebula-graph.io nebularestores.apps.nebula-graph.io nebulaautoscalers.autoscaling.nebula-graph.io

Last update: November 15, 2023

# 17.4 Cluster administration

# 17.4.1 Deployment

# Install a NebulaGraph cluster using NebulaGraph Operator

Using NebulaGraph Operator to install NebulaGraph clusters enables automated cluster management with automatic error recovery. This topic covers two methods, kubectl apply and helm, for installing clusters using NebulaGraph Operator.

# Left storical version compatibility

NebulaGraph Operator versions 1.x are not compatible with NebulaGraph versions below 3.x.

PREREQUISITES

- Install NebulaGraph Operator
- Create a StorageClass

USE KUBECTL APPLY

1. Create a namespace for storing NebulaGraph cluster-related resources. For example, create the nebula namespace.

kubectl create namespace nebula

2. Create a YAML configuration file nebulacluster.yaml for the cluster. For example, create a cluster named nebula.

 $\mathbf{E}_{\mathbf{x}}$  pand to view an example configuration for the nebula cluster  $\mathbf{Y}$ apiVersion: apps.nebula-graph.io/v1alpha1 kind: NebulaCluster metadata: name: nebula namespace: default spec: # Control the Pod scheduling strategy. enablePVReclaim: false
# Enable monitoring. exporter: image: vesoft/nebula-stats-exporter version: v3.3.0 replicas: 1 maxRequests: 20
# Custom Agent image for cluster backup and restore, and log cleanup. agent: image: vesoft/nebula-agent version: latest resources: requests cpu: "100m" memory: "128Mi" limits: cpu: "200m" cpu: "200m" memory: "256Mi" # Configure the image pull policy. imagePullPolicy: Always # Select the nodes for Pod scheduling. nodeSelector nebula: cloud # Dependent controller name.
reference: name: statefulsets.apps
version: v1 # Scheduler name.
schedulerName: default-scheduler # Start NebulaGraph Console service for connecting to the Graph service. console: image: vesoft/nebula-console image: vesoft/nebula-console version: nightly username: "demo" password: "test" # Graph service configuration. graphd: # Used to check if the Graph service is running normally. # readinessProbe: # failureThreshold: 3 # httpGet:
# path: /status port: 19669 scheme: HTTP initialDelaySeconds: 40 periodSeconds: 10 successThreshold: 1 timeoutSeconds: 10 # Container image for the Graph service. image: vesoft/nebula-graphd logVolumeClaim: resources: requests: storage: 2Gi storage: Zoi # Storage class name for storing Graph service logs. storageClassName: local-sc # Number of replicas for the Graph service Pod. replicas: 1 # Resource configuration for the Graph service. resources: limits: cpu: "1" memory: 1Gi requests: cpu: 500m memory: 500Mi
# Version of the Graph service. wersion of the draph service.
version: v3.6.0
# Custom flags configuration for the Graph service. config: {}
# Meta service configuration. metad: # readinessProbe failureThreshold: 3 httpGet: path: /status port: 19559 scheme: HTTP initialDelaySeconds: 5 periodSeconds: 5 . successThreshold: 1 timeoutSeconds: 5 # # Container image for the Meta service. image: vesoft/nebula-metad

logVolumeClaim: requests: storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi requests: cpu: "1" memory: IGi requests: cpu: 500m memory: 500Mi version: V3.6.0 # Custom flags configuration for the Meta service. config: {} # Storage service configuration. storaged: # readinesSProbe: # failureThreshold: 3 # httpGet: # part: /status # port: 19779 # scheme: HTTP # initialDelaySeconds: 40 # periodSeconds: 10 # successThreshold: 1 # timeoutSeconds: 5 # Container image for the Storage service. image: vest/rebula-graphd logVolumeClaim: requests: storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZGi storage: ZG  $\bigcirc$ 

Parameter	Default Value	Description
metadata.name	-	The name of the created NebulaGraph cluster.
spec.console	-	Launches a Console container for connecting to the Graph service. For configuration details, see nebula-console.
spec.topologySpreadConstraints	-	Controls the scheduling strategy for Pods. For more details, see Topology Spread Constraints. When the value of topologyKey is kubernetes.io/zone, the value of whenUnsatisfiable must be set to DoNotSchedule, and the value of spec.schedulerName should be nebula-scheduler.
spec.graphd.replicas	1	The number of replicas for the Graphd service.
spec.graphd.image	vesoft/ nebula- graphd	The container image for the Graphd service.
spec.graphd.version	v3.6.0	The version of the Graphd service.
spec.graphd.service		Configuration for accessing the Graphd service via a Service.
<pre>spec.graphd.logVolumeClaim.storageClassName</pre>	-	The storage class name for the log volume claim of the Graphd service. When using sample configuration, replace it with the name of the pre-created storage class. See Storage Classes for creating a storage class.
spec.metad.replicas	1	The number of replicas for the Metad service.
spec.metad.image	vesoft/ nebula- metad	The container image for the Metad service.
spec.metad.version	v3.6.0	The version of the Metad service.
<pre>spec.metad.dataVolumeClaim.storageClassName</pre>	-	Storage configuration for the data disk of the Metad service. When using sample configuration, replace it with the name of the pre-created storage class. See Storage Classes for creating a storage class.
<pre>spec.metad.logVolumeClaim.storageClassName</pre>	-	Storage configuration for the log disk of the Metad service. When using sample configuration, replace it with the name of the pre-created storage class. See Storage Classes for creating a storage class.
spec.storaged.replicas	3	The number of replicas for the Storaged service.
spec.storaged.image	vesoft/ nebula- storaged	The container image for the Storaged service.
spec.storaged.version	v3.6.0	The version of the Storaged service.
<pre>spec.storaged.dataVolumeClaims.resources.requests.storage</pre>	-	The storage size for the data disk of the Storaged service. You can specify multiple data disks. When specifying multiple data disks, the paths are like /usr/local/nebula/data1, /usr/local/nebula/data2, and so on.
<pre>spec.storaged.dataVolumeClaims.storageClassName</pre>	-	Storage configuration for the data disks of the Storaged service. When using sample configuration, replace it with the name of the pre-created storage

Parameter	Default Value	Description
		class. See Storage Classes for creating a storage class.
spec.storaged.logVolumeClaim.storageClassName	-	Storage configuration for the log disk of the Storaged service. When using sample configuration, replace it with the name of the pre-created storage class. See Storage Classes for creating a storage class.
<pre>spec.<metad storaged graphd>.securityContext</metad storaged graphd></pre>	{}	Defines the permission and access control for the cluster containers to control access and execution of container operations. For details, see SecurityContext.
spec.agent	{}	Configuration for the Agent service used for backup and recovery, and log cleaning functions. If you don't customize this configuration, the default configuration is used.
spec.reference.name	{}	The name of the controller it depends on.
spec.schedulerName	default- scheduler	The name of the scheduler.
spec.imagePullPolicy	Always	The image pull policy for NebulaGraph images. For more details on pull policies, please see Image pull policy.
spec.logRotate	{}	Log rotation configuration. For details, see Managing Cluster Logs.
spec.enablePVReclaim	false	Defines whether to automatically delete PVCs after deleting the cluster to release data. For details, see Reclaim PV.
spec.metad.licenseManagerURL	-	Configures the URL pointing to the License Manager (LM), consisting of the access address and port (default port 9119). For example, 192.168.8.xxx:9119. For the NebulaGraph Enterprise Edition only.
spec.storaged.enableAutoBalance	false	Whether to enable automatic balancing. For details, see Balancing Storage Data After Scaling Out.
spec.enableBR	false	Defines whether to enable the BR tool. For details, see Backup and Restore.
spec.imagePullSecrets	0	Defines the Secret required to pull images from a private repository.

# 3. Create the NebulaGraph cluster.

kubectl create -f nebulacluster.yaml -n nebula

Output:

nebulacluster.apps.nebula-graph.io/nebula created

If you don't specify the namespace using -n, it will default to the default namespace.

# 4. Check the status of the NebulaGraph cluster.

kubectl get nebulaclusters nebula -n nebula

# Output:

NAME	READY	GRAPHD-DESIRED	GRAPHD-READY	METAD-DESIRED	METAD-READY	STORAGED-DESIRED	STORAGED-READY	AGE
nebula	True	1	1	1	1	1	1	86s

USE HELM

1. Add the NebulaGraph Operator Helm repository (if it's already added, run the next step directly).

helm repo add nebula-operator https://vesoft-inc.github.io/nebula-operator/charts

 $2. \ Update the Helm repository to fetch the latest resources.$ 

helm repo update nebula-operator

3. Set environment variables for the configuration parameters required for installing the cluster.

 export
 NEBULA\_CLUSTER\_NAME=nebula
 # Name of the NebulaGraph cluster.

 export
 NEBULA\_CLUSTER\_NAMESPACE=nebula
 # Namespace for the NebulaGraph cluster.

 export
 STORAGE\_CLASS\_NAME=local-sc
 # StorageClass for the NebulaGraph cluster.

#### 4. Create a namespace for the NebulaGraph cluster if it is not created.

kubectl create namespace "\${NEBULA\_CLUSTER\_NAMESPACE}"

- 5. Check the customizable configuration parameters for the nebula-cluster Helm chart of the nebula-operator when creating the cluster.
- Visit nebula-cluster/values.yaml to see all the configuration parameters for the NebulaGraph cluster.

 $\ensuremath{{\scriptstyle\bullet}}$  Run the following command to view all the configurable parameters.

helm show values nebula-operator/nebula-cluster

```
\mathbf{k}ample to view all configurable parameters \checkmark
nebula:
   ebula:
version: v3.6.0
imagePullPolicy: Always
storageClassName: ""
enablePVReclaim: false
    enableBR: false
   enableForceUpdate: false
schedulerName: default-scheduler
topologySpreadConstraints:

    topologyKey: "kubernetes.io/hostname"
whenUnsatisfiable: "ScheduleAnyway"

     logRotate: {}
    reference:
       name: statefulsets.apps
version: v1
    graphd:
    image: vesoft/nebula-graphd
       replicas: 2
serviceType: NodePort
        env: []
config: {}
       resources:
requests:
cpu: "500m"
memory: "500Mi"
       memory: "500MI"
limits:
cpu: "1"
memory: "500Mi"
logVolume:
enable: true
storage: "500Mi"
podLabels: {}
coddamptations: {}
        podAnnotations: {}
securityContext: {}
        nodeSelector: {}
tolerations: []
        affinity: {}
readinessProbe: {}
        LivenessProbe: {}
initContainers: []
        sidecarContainers: []
volumes: []
volumeMounts: []
    metad:
       image: vesoft/nebula-metad
replicas: 3
       env: []
config: {}
        resources:
           requests:
              cpu: "500m"
memory: "500Mi"
       memory: "500Mi'
limits:
cpu: "1"
memory: "1Gi"
logVolume:
enable: true
storage: "500Mi"
dataVolume:
storage: "2Gi"
licenssManager[R]:
       storage: "26i"
licenseManagerURL: ""
license: {}
podLabeLs: {}
podAnnotations: {}
securityContext: {}
       securityContext: {}
nodeSelector: {}
tolerations: []
affinity: {}
readinessProbe: {}
        LivenessProbe: {}
initContainers: []
        sidecarContainers: []
volumes: []
volumeMounts: []
    storaged:
       image: vesoft/nebula-storaged
replicas: 3
opus 52
        env: []
config: {}
resources:
            requests:
cpu: "500m"
memory: "500Mi"
        limits:
cpu: "1"
memory: "1Gi"
logVolume:
        enable: true
storage: "500Mi"
dataVolumes:
- storage: "10Gi"
        enableAutoBalance: false
podLabels: {}
```

podAnnotations: {}
securityContext: {}
nodeSelector: {}
tolerations: []
affinity: {}
readinessProbe: {}
LivenessProbe: {}
initContainers: []
volumes: []
volumes: []
volumeHounts: [] exporter: image: vesoft/nebula-stats-exporter version: v3.3.0 replicas: 1 env: [] resources: requests: cpu: "100m" memory: "128Mi" limits: memory: "128Wi" timits: cpu: "200m" memory: "256Wi" podLabels: {} podLabels: {} podLabels: {} modeSelector: {} tolerations: [] affinity: {} readinessProbe: {} LivenessProbe: {} tivenessProbe: {} sidecarContainers: [] volume\*(] volume\*(] maxRequests: 20 agent: image: vesoft/nebula-agent version: latest resources: cpu: "100m" memory: "128Mi" limits: cpu: "200m" memory: "256Mi" console: console: username: root password: nebula image: vesoft/nebula-console version: latest nodeSelector: {} alpineImage: "" imagePullSecrets: []
nameOverride: ""
fullnameOverride: ""

 $\dot{\mathbf{L}}$  pand to view parameter descriptions  $\checkmark$
Parameter	Default Value	Description
nebula.version	v3.6.0	Version of the cluster.
nebula.imagePullPolicy	Always	Container image pull policy. Always means always attempting to pull the latest image from the remote.
nebula.storageClassName		Name of the Kubernetes storage class for dynamic provisioning of persistent volumes.
nebula.enablePVReclaim	false	Enable persistent volume reclaim. See Reclaim PV for details.
nebula.enableBR	false	Enable the backup and restore feature. See Backup and Restore with NebulaGraph Operator for details.
nebula.enableForceUpdate	false	Force update the Storage service without transferring the leader partition replicas. See Optimize leader transfer in rolling updates for details.
nebula.schedulerName	default-scheduler	Name of the Kubernetes scheduler. Must be configured as nebula-scheduler when using the Zone feature.
nebula.topologySpreadConstraints	[]	Control the distribution of pods in the cluster.
nebula.logRotate	0	Log rotation configuration. See Manage cluster logs for details.
nebula.reference	<pre>{"name": "statefulsets.apps", "version": "v1"}</pre>	The workload referenced for a NebulaGraph cluster.
nebula.graphd.image	vesoft/nebula-graphd	Container image for the Graph service.
nebula.graphd.replicas	2	Number of replicas for the Graph service.
nebula.graphd.serviceType	NodePort	Service type for the Graph service, defining how the Graph service is accessed. See Connect to the Cluster for details.
nebula.graphd.env	0	Container environment variables for the Graph service.
nebula.graphd.config	0	Configuration for the Graph service. See Customize the configuration of the NebulaGraph cluster for details.
nebula.graphd.resources	<pre>{"resources":{"requests":     {"cpu":"500m","memory":"500Mi"},"Limits":     {"cpu":"1","memory":"500Mi"}}}</pre>	Resource limits and requests for the Graph service.
nebula.graphd.logVolume	{"logVolume": {"enable": true,"storage": "500Mi"}}	Log storage configuration for the Graph service. When enable is false, log volume is not used.
nebula.metad.image	vesoft/nebula-metad	

Parameter	Default Value	<b>Description</b> Container image for the Meta service.
nebula.metad.replicas	3	Number of replicas for the Meta service.
nebula.metad.env	[]	Container environment variables for the Meta service.
nebula.metad.config	0	Configuration for the Meta service. See Customize the configuration of the NebulaGraph cluster for details.
nebula.metad.resources	{"resources":{"requests": {"cpu":"500m","memory":"500Mi"},"Limits": {"cpu":"1","memory":"1Gi"}}}	Resource limits and requests for the Meta service.
nebula.metad.logVolume	<pre>{"logVolume": {"enable": true,"storage": "500Mi"}}</pre>	Log storage configuration for the Meta service. When enable is false, log volume is not used.
nebula.metad.dataVolume	{"dataVolume": {"storage": "2Gi"}}	Data storage configuration for the Meta service.
nebula.metad.licenseManagerURL	н	URL for the license manager (LM) to obtain license information. For the NebulaGraph Enterprise Edition only.
nebula.storaged.image	vesoft/nebula-storaged	Container image for the Storage service.
nebula.storaged.replicas	3	Number of replicas for the Storage service.
nebula.storaged.env	[]	Container environment variables for the Storage service.
nebula.storaged.config	0	Configuration for the Storage service. See Customize the configuration of the NebulaGraph cluster for details.
nebula.storaged.resources	<pre>{"resources":{"requests":     {"cpu":"500m","memory":"500Mi"},"Limits":     {"cpu":"1","memory":"1Gi"}}</pre>	Resource limits and requests for the Storage service.
nebula.storaged.logVolume	{"logVolume": {"enable": true,"storage": "500Mi"}}	Log storage configuration for the Storage service. When enable is false, log volume is not used.
nebula.storaged.dataVolumes	{"dataVolumes": [{"storage": "10Gi"}]}	Data storage configuration for the Storage service. Supports specifying multiple data volumes.
nebula.storaged.enableAutoBalance	false	Enable automatic balancing. See Balance storage data after scaling out for details.
nebula.exporter.image	vesoft/nebula-stats-exporter	Container image for the Exporter service.
nebula.exporter.version	v3.3.0	Version of the Exporter service.

Parameter	Default Value	Description
nebula.exporter.replicas	1	Number of replicas for the Exporter service.
nebula.exporter.env	0	Environment variables for the Exporter service.
nebula.exporter.resources	<pre>{"resources":{"requests":     {"cpu":"100m","memory":"128Mi"},"Limits":     {"cpu":"200m","memory":"256Mi"}}}</pre>	Resource limits and requests for the Exporter service.
nebula.agent.image	vesoft/nebula-agent	Container image for the agent service.
nebula.agent.version	latest	Version of the agent service.
nebula.agent.resources	{"resources":{"requests": {"cpu":"100m","memory":"128Mi"},"Limits": {"cpu":"200m","memory":"256Mi"}}}	Resource limits and requests for the agent service.
nebula.console.username	root	Username for accessing the NebulaGraph Console client. See Connect to the cluster for details.
nebula.console.password	nebula	Password for accessing the NebulaGraph Console client.
nebula.console.image	vesoft/nebula-console	Container image for the NebulaGraph Console client.
nebula.console.version	latest	Version of the NebulaGraph Console client.
nebula.alpineImage	111	Alpine Linux container image used to obtain zone information for nodes.
imagePullSecrets	0	Names of secrets to pull private images.
nameOverride		Cluster name.
fullnameOverride		Name of the released chart instance.
nebula. <graphd metad storaged  exporter&gt;.podLabels</graphd metad storaged  	0	Additional labels to be added to the pod.
nebula. <graphd metad storaged  exporter&gt;.podAnnotations</graphd metad storaged  	{}	Additional annotations to be added to the pod.
nebula. <graphd metad storaged  exporter&gt;.securityContext</graphd metad storaged  	{}	Security context for setting pod-level security attributes, including user ID, group ID, Linux Capabilities, etc.
nebula. <graphd metad storaged  exporter&gt;.nodeSelector</graphd metad storaged  	0	Label selectors for determining which nodes to run the pod on.
nebula. <graphd metad storaged  exporter&gt;.tolerations</graphd metad storaged  	[]	Tolerations allow a pod to be scheduled to nodes with specific taints.
nebula. <graphd metad storaged  exporter&gt;.affinity</graphd metad storaged  	{}	Affinity rules for the pod, including node affinity, pod affinity, and pod anti-affinity.
nebula. <graphd metad storaged  exporter&gt;.readinessProbe</graphd metad storaged  	{}	Probe to check if a container is ready to accept service requests. When the

Parameter	Default Value	<b>Description</b> probe returns success, traffic can be routed to the container.
nebula. <graphd metad storaged  exporter&gt;.livenessProbe</graphd metad storaged  	0	Probe to check if a container is still running. If the probe fails, Kubernetes will kill and restart the container.
nebula. <graphd metad storaged  exporter&gt;.initContainers</graphd metad storaged  		Special containers that run before the main application container starts, typically used for setting up the environment or initializing data.
nebula. <graphd metad storaged  exporter&gt;.sidecarContainers</graphd metad storaged  		Containers that run alongside the main application container, typically used for auxiliary tasks such as log processing, monitoring, etc.
nebula. <graphd metad storaged  exporter&gt;.volumes</graphd metad storaged  	[]	Storage volumes to be attached to the service pod.
<pre>nebula.<graphd metad storaged  exporter="">.volumeMounts</graphd metad storaged ></pre>		Specifies where to mount the storage volume inside the container.

## 6. Create the NebulaGraph cluster.

You can use the --set flag to customize the default values of the NebulaGraph cluster configuration. For example, --set nebula.storaged.replicas=3 sets the number of replicas for the Storage service to 3.

- helm install "\${NEBULA\_CLUSTER\_NAME}" nebula-operator/nebula-cluster \
   # Specify the version of the cluster chart. If not specified, it will install the latest version by default.
   # You can check all chart versions by running the command: helm search repo -l nebula-operator/nebula-cluster

  - --version=1.8.0 \ # Specify the namespace for the NebulaGraph cluster. --namespace="\${NEBULA\_CLUSTER\_NAMESPACE}" \

  - # Customize the cluster name.
    --set nameOverride="\${NEBULA\_CLUSTER\_NAME}" \
  - --set nebula.storageClassName="\${STORAGE\_CLASS\_NAME}" \ # Specify the version for the NebulaGraph cluster. --set nebula.version=v3.6.0

## 7. Check the status of NebulaGraph cluster pods.

kubectl -n "\${NEBULA\_CLUSTER\_NAMESPACE}" get pod -l "app.kubernetes.io/cluster=\${NEBULA\_CLUSTER\_NAME}"

## Output:

NAME	READY	STATUS	RESTARTS	AGE
nebula-exporter-854c76989c-mp725	<b>1</b> /1	Running	0	14h
nebula-graphd-0	<b>1</b> /1	Running	0	14h
nebula-graphd-1	<b>1</b> /1	Running	0	14h
nebula-metad-0	<b>1</b> /1	Running	0	14h
nebula-metad-1	<b>1</b> /1	Running	0	14h
nebula-metad-2	<b>1</b> /1	Running	0	14h
nebula-storaged-0	<b>1</b> /1	Running	0	14h
nebula-storaged-1	<b>1</b> /1	Running	0	14h
nebula-storaged-2	<mark>1</mark> /1	Running	0	14h

Last update: March 7, 2024

## Upgrade NebulaGraph clusters created with NebulaGraph Operator

This topic introduces how to upgrade a NebulaGraph cluster created with NebulaGraph Operator.

# L Jacy version compatibility

The 1.x version NebulaGraph Operator is not compatible with NebulaGraph of version below v3.x.

LIMITS

- Only for upgrading the NebulaGraph clusters created with NebulaGraph Operator.
- Only support upgrading the NebulaGraph version from 3.5.0 to 3.6.0.
- For upgrading NebulaGraph Enterprise Edition clusters, contact us.

#### PREREQUISITES

You have created a NebulaGraph cluster. For details, see Create a NebulaGraph cluster.

UPGRADE A NEBULAGRAPH CLUSTER WITH KUBECTL

The following steps upgrade a NebulaGraph cluster from version 3.5.0 to v3.6.0.

1. Check the image version of the services in the cluster.

kubectl get pods -l app.kubernetes.io/cluster=nebula -o jsonpath="{.items[\*].spec.containers[\*].image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c

Output:

```
1 vesoft/nebula-graphd:3.5.0
1 vesoft/nebula-metad:3.5.0
3 vesoft/nebula-storaged:3.5.0
```

- 2. Edit the nebula cluster configuration to change the version value of the cluster services from 3.5.0 to v3.6.0.
- a. Open the YAML file for the nebula cluster.

kubectl edit nebulacluster nebula -n <namespace>

b. Change the value of version.

After making these changes, the YAML file should look like this:

```
apiVersion: apps.nebula-graph.io/vlalphal
kind: NebulaCluster
metadata:
name: nebula
spec:
graphd:
version: v3.6.0 // Change the value from 3.5.0 to v3.6.0.
...
storaged:
version: v3.6.0 // Change the value from 3.5.0 to v3.6.0.
...
```

3. Apply the configuration.

After saving the YAML file and exiting, Kubernetes automatically updates the cluster's configuration and starts the cluster upgrade.

4. After waiting for about 2 minutes, run the following command to see if the image versions of the services in the cluster have been changed to v3.6.0.

kubectl get pods -l app.kubernetes.io/cluster=nebula -o jsonpath="{.items[\*].spec.containers[\*].image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c

Output:

```
1 vesoft/nebula-graphd:v3.6.0
1 vesoft/nebula-metad:v3.6.0
3 vesoft/nebula-storaged:v3.6.0
```

UPGRADE A NEBULAGRAPH CLUSTER WITH HELM

1. Update the information of available charts locally from chart repositories.

helm repo update

2. Set environment variables to your desired values.

export NEBULA\_CLUSTER\_NAME=nebula # The desired NebulaGraph cluster name. export NEBULA\_CLUSTER\_NAMESPACE=nebula # The desired namespace where your NebulaGraph cluster locates.

#### 3. Upgrade a NebulaGraph cluster.

For example, upgrade a cluster to v3.6.0.

```
helm upgrade "${NEBULA_CLUSTER_NAME}" nebula-operator/nebula-cluster \
    --namespace="${NEBULA_CLUSTER_NAMESPACE}" \
    --set nameOverride=${NEBULA_CLUSTER_NAME} \
    --set nebula.version=V3.6.0
```

The value of --set nebula.version specifies the version of the cluster you want to upgrade to.

4. Run the following command to check the status and version of the upgraded cluster.

Check cluster status:

<pre>\$ kubectl -n "\${NEB</pre>	ULA_CLUS	TER_NAMESP	ACE}" get	pod -l	"app.kubernetes.io/cluster=\${NEBULA_CLUSTER_NAME}"
NAME	READY	STATUS	RESTARTS	AGE	
nebula-graphd-0	<b>1</b> /1	Running	0	2m	
nebula-graphd-1	<b>1</b> /1	Running	0	2m	
nebula-metad-0	<b>1</b> /1	Running	0	2m	
nebula-metad-1	<b>1</b> /1	Running	0	2m	
nebula-metad-2	<b>1</b> /1	Running	0	2m	
nebula-storaged-0	<b>1</b> /1	Running	0	2m	
nebula-storaged-1	<b>1</b> /1	Running	0	2m	
nebula-storaged-2	<mark>1</mark> /1	Running	0	2m	

#### Check cluster version:

```
$ kubectl get pods -l app.kubernetes.io/cluster=nebula -o jsonpath="{.items[*].spec.containers[*].image}" |tr -s '[[:space:]]' '\n' |sort |uniq -c
1 vesoft/nebula-graphd:v3.6.0
1 vesoft/nebula-metad:v3.6.0
3 vesoft/nebula-storaged:v3.6.0
```

· •

ACCELERATE THE UPGRADE PROCESS

The upgrade process of a cluster is a rolling update process and can be time-consuming due to the state transition of the leader partition replicas in the Storage service. You can configure the enableForceUpdate field in the cluster instance's YAML file to skip the leader partition replica transfer operation, thereby accelerating the upgrade process. For more information, see Specify a rolling update strategy.

#### TROUBLESHOOTING

If you encounter issues during the upgrade process, you can check the logs of the cluster service pods.

kubectl logs <pod-name> -n <namespace>

Additionally, you can inspect the cluster's status and events.

kubectl describe nebulaclusters <cluster-name> -n <namespace>

Last update: December 14, 2023

## Delete a NebulaGraph cluster

This topic explains how to delete a NebulaGraph cluster created using NebulaGraph Operator.

USAGE LIMITATIONS

- Deletion is only supported for NebulaGraph clusters created with the NebulaGraph Operator.
- You cannot delete a NebulaGraph cluster that has deletion protection enabled. For more information, see Configure deletion protection.

DELETE A NEBULAGRAPH CLUSTER USING KUBECTL

## 1. View all created clusters.

kubectl get nc --all-namespaces

Example output:

NAMESPACE	NAME	READY	GRAPHD-DESIRED	GRAPHD-READY	METAD-DESIRED	METAD-READY	STORAGED-DESIRED	STORAGED-READY	AGE
default	nebula	True	2	2	3	3	3	3	38h
nebula	nebula2	True	1	1	1	1	1	1	2m7s

2. Delete a cluster. For example, run the following command to delete a cluster named nebula2 :

2.	Delete a clus	ster. i or example	<i>, 1 uii tii</i>	e following command to de		uster numeu net		
	kubectl delete no	c nebula2 -n nebula						
	Example out	put:						
	nebulacluster.neb	bula-graph.io "nebula2" (	deleted					
3.	Confirm the	deletion.						
	kubectl get nc ne	ebula2 -n nebula						
	Example out	put:						
	No resources four	nd in nebula namespace.						
	DELETE A NEBU	JLAGRAPH CLUSTER	USING HEL	м				
1.	View all Heli	m releases.						
	helm listall-r	namespaces						
	Example out	put:						
	NAME nebula nebula-operator	NAMESPACE default nebula-operator-system	REVISION 1 3	UPDATED 2023-11-06 20:16:07.913136377 +0800 CS 2023-11-06 12:06:24.742397418 +0800 CS	STATUS T deployed T deployed	CHART nebula-cluster-1.7.1 nebula-operator-1.7.1	APP VERSION 1.7.1 1.7.1	

2. View detailed information about a Helm release. For example, to view the cluster information for a Helm release named nebula:

helm get values nebula -n default

## Example output:

USER-SUPPLIED VALUES:
imagePullSecrets:
<pre>- name: secret_for_pull_image</pre>
<pre>nameOverride: nebula # The cluster name</pre>
nebula:
graphd:
image: reg.vesoft-inc.com/xx
metad:
<pre>image: reg.vesoft-inc.com/xx</pre>
LicenseManagerURL: xxx:9119
storageClassName: local-sc
storaged:
image: reg.vesoft-inc.com/xx
version: v1.8.0 # The cluster version

 $_{\mbox{3.}}$  Uninstall a Helm release. For example, to uninstall a Helm release named  $\mbox{ nebula}$  :

helm uninstall nebula -n default

## Example output:

#### release "nebula" uninstalled

Once the Helm release is uninstalled, NebulaGraph Operator will automatically remove all K8s resources associated with that release.

4. Verify that the cluster resources are removed.

kubectl get nc nebula -n default

## Example output:

No resources found in default namespace.

Last update: March 6, 2024

## 17.4.2 Customize the configuration of the NebulaGraph cluster

The Meta, Storage, and Graph services each have their default configurations within the NebulaGraph cluster. NebulaGraph Operator allows for the customization of these cluster service configurations. This topic describes how to update the settings of the NebulaGraph cluster.

#### Q Note

 $Configuring \ the \ parameters \ of \ the \ NebulaGraph \ cluster \ via \ Helm \ isn't \ currently \ supported.$ 

#### Prerequisites

A cluster is created using NebulaGraph Operator. For details, see Create a NebulaGraph Cluster.

#### **Configuration method**

You can update the configurations of cluster services by customizing parameters through spec.<metad|graphd|storaged>.config. NebulaGraph Operator loads the configurations from config into the corresponding service's ConfigMap, which is then mounted into the service's configuration file directory (/usr/local/nebula/etc/) at the time of the service launch.

The structure of config is as follows:

#### Config map[string]string `json:"config,omitempty"

For instance, when updating the Graph service's enable\_authorize parameter settings, the spec.graphd.config parameter can be specified at the time of cluster creation, or during cluster runtime.

```
apiVersion: apps.nebula-graph.io/vlalphal
kind: NebulaCluster
metadata:
name: nebula
namespace: default
spec:
graphd:
...
config: // Custom-defined parameters for the Graph service.
"enable_authorize": "true" // Enable authorization. Default value is false.
```

If you need to configure config for the Meta and Storage services, add corresponding configuration items to spec.metad.config and spec.storaged.config.

#### **Configurable parameters**

For more detailed information on the parameters that can be set under the config field, see the following:

- Meta Service Configuration Parameters
- Storage Service Configuration Parameters
- Graph Service Configuration Parameters

#### Parameter updates & Pod restart rules

Configuration parameters for cluster services fall into two categories: those which require a service restart for any updates; and those which can be dynamically updated during service runtime. For the latter type, the updates will not be saved; subsequent to a service restart, configurations will revert to the state as shown in the configuration file.

Regarding if the configuration parameters support dynamic updates during service runtime, please verify the information within the **Whether supports runtime dynamic modifications** column on each of the service configuration parameter detail pages linked above or see Dynamic runtime flags.

During the update of cluster service configurations, keep the following points in mind:

- If the updated parameters under config **all allow for dynamic runtime updates**, a service Pod restart will not be triggered and the configuration parameter updates will not be saved.
- If the updated parameters under config **include one or more that don't allow for dynamic runtime updates**, a service Pod restart will be triggered, but only updates to those parameters that don't allow for dynamic updates will be saved.

#### Q Note

If you wish to modify the parameter settings during cluster runtime without triggering a Pod restart, make sure that all the parameters support dynamic updates during runtime.

## Customize port configuration

The following example demonstrates how to customize the port configurations for the Meta, Storage, and Graph services.

You can add port and ws\_http\_port parameters to the config field in order to set custom ports. For detailed information regarding these two parameters, see the networking configuration sections at Meta Service Configuration Parameters, Storage Service Configuration Parameters, Graph Service Configuration Parameters.

#### Q Note

• After customizing the port and ws\_http\_port parameter settings, a Pod restart is triggered and then the updated settings take effect after the restart.

• Once the cluster is started, it is not recommended to modify the port parameter.

# $_{1.}$ Modify the cluster configuration file.

a. Open the cluster configuration file.

kubectl edit nc nebula

b. Modify the configuration file as follows.

Add the config field to the graphd, metad, and storaged sections to customize the port configurations for the Graph, Meta, and Storage services, respectively.

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
  name: nebula
  namespace: default
spec:
  graphd:
    config: // Custom port configuration for the Graph service.
    port: "3669"
       .
ws_http_port: "8080"
    resources:
      requests
         cpu: "200m
          nemory: "500Mi"
      limits:
cpu: "1"
    memory: "1Gi"
replicas: 1
     image: vesoft/nebula-graphd
    version: v3.6.0
  metad:
    config: // Custom port configuration for the Meta service.
    ws_http_port: 8081
resources:
      requests:
cpu: "300m'
          nemory: "500Mi"
      limits:
cpu: "1"
    memory: "1Gi"
replicas: 1
    image: vesoft/nebula-metad
    version: v3.6.0
dataVolumeClaim:
      resources:
         requests:
          storage: 2Gi
      storageClassName: local-path
  storaged:
    config: // Custom port configuration for the Storage service.
      ws_http_port: 8082
    resources:
      requests
         cpu: "300m"
         memory: "500Mi"
    limits:
cpu: "1"
memory: "1Gi"
replicas: 1
    image: vesoft/nebula-storaged
    version: v3.6.0
    dataVolumeClaims:
     - resources:
         requests:
      storage: 2Gi
storageClassName: local-path
    enableAutoBalance: true
  reference:
    name: statefulsets.apps
  version: v1
schedulerName: default-scheduler
imagePullPolicy: IfNotPresent
  imagePullSecrets:
    name: nebula-image
  enablePVReclaim: true
  topologySpreadConstraints:
    topologyKey: kubernetes.io/hostname
    whenUnsatisfiable: "ScheduleAnyway"
```

2. Save the changes.

Changes will be saved automatically after saving the file.

- a. Press Esc to enter command mode.
- b. Enter :wq to save and exit.

# $_{\ensuremath{3.}}$ Validate that the configurations have taken effect.

KUDECLL get SVC	kubectl	get	SVC	
-----------------	---------	-----	-----	--

# Example output:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nebula-graphd-headless	ClusterIP	None	<none></none>	3669/TCP,8080/TCP	10m
nebula-graphd-svc	ClusterIP	10.102.13.115	<none></none>	3669/TCP,8080/TCP	10m
nebula-metad-headless	ClusterIP	None	<none></none>	9559/TCP,8081/TCP	11m
nebula-storaged-headless	ClusterIP	None	<none></none>	9779/TCP,8082/TCP,9778/TCP	11m

As can be noticed, the Graph service's RPC daemon port is changed to 3669 (default 9669), the HTTP port to 8080 (default 19669); the Meta service's HTTP port is changed to 8081 (default 19559); the Storage service's HTTP port is changed to 8082 (default 19779).

Last update: March 6, 2024

## 17.4.3 Storage management

#### Dynamically expand persistent volumes

In a Kubernetes environment, NebulaGraph's data is stored on Persistent Volumes (PVs). Dynamic volume expansion refers to increasing the capacity of a volume without stopping the service, enabling NebulaGraph to accommodate growing data. This topic explains how to dynamically expand the PV for NebulaGraph services in a Kubernetes environment.

#### O Note

• After the cluster is created, you cannot dynamically increase the number of PVs while the cluster is running.

• The method described in this topic is only for online volume expansion and does not support volume reduction.

#### BACKGROUND

In Kubernetes, a StorageClass is a resource that defines a particular storage type. It describes a class of storage, including its provisioner, parameters, and other details. When creating a PersistentVolumeClaim (PVC) and specifying a StorageClass, Kubernetes automatically creates a corresponding PV. The principle of dynamic volume expansion is to edit the PVC and increase the volume's capacity. Kubernetes will then automatically expand the capacity of the PV associated with this PVC based on the specified storageClassName in the PVC. During this process, new PVs are not created; the size of the existing PV is changed. Only dynamic storage volumes, typically those associated with a storageClassName, support dynamic volume expansion. Additionally, the altowVolumeExpansion field in the StorageClass must be set to true. For more details, see the Kubernetes documentation on expanding Persistent Volume Claims.

In NebulaGraph Operator, you cannot directly edit PVC because Operator automatically creates PVC based on the configuration in the spec.<metad|storaged>.dataVolumeClaim of the Nebula Graph cluster. Therefore, you need to modify the cluster's configuration to update the PVC and trigger dynamic online volume expansion for the PV.

#### PREREQUISITES

- Kubernetes version is equal to or greater than 1.18.
- A StorageClass has been created in the Kubernetes environment. For details, see Expanding Persistent Volumes Claims.
- Ensure the allowVolumeExpansion field in the StorageClass is set to true.
- Make sure that the provisioner configured in the StorageClass supports dynamic expansion.
- A NebulaGraph cluster is created in Kubernetes. For specific steps, see Create a NebulaGraph cluster.
- NebulaGraph cluster Pods are in running status.

#### ONLINE VOLUME EXPANSION EXAMPLE

In the following example, we assume that the StorageClass is named <code>ebs-sc</code> and the NebulaGraph cluster is named <code>nebula</code>. We will demonstrate how to dynamically expand the PV for the Storage service.

1. Check the status of the Storage service Pod:

	kubectl get pod
	Example output:
	nebula-storaged-0 1/1 Running 0 43h
2.	Check the PVC and PV information for the Storage service:
	# View PVC kubectl get pvc
	Example output:
	storaged-data-nebula-storaged-0 Bound pvc-36ca3871-9265-460f-b812-7e73a718xxxx 5Gi RWO ebs-sc 43h
	# View PV and confirm that the capacity of the PV is 5Gi kubectl get pv
	Example output:
	pvc-36ca3871-9265-460f-b812-xxx 5Gi RWO Delete Bound default/storaged-data-nebula-storaged-0 ebs-sc 43h
3.	Assuming all the above-mentioned prerequisites are met, use the following command to request an expansion of the PV for the Storage service to 10Gi:
	kubectl patch nc nebulatype='merge'patch '{"spec": {"storaged": {"dataVolumeClaims": {{"requests": {"storage": "106i"}}, "storageClassName": "ebs-sc"}]}}}'
	Example output:
	nebulacluster.apps.nebula-graph.io/nebula patched
4.	After waiting for about a minute, check the expanded PVC and PV information:
	kubectl get pvc
	Example output:
	storaged-data-nebula-storaged-0 Bound pvc-36ca3871-9265-460f-b812-7e73a718xxxx 10Gi RWO ebs-sc 43h
	kubectl get pv
	Example output:
	pvc-36ca3871-9265-460f-b812-xxx 106i RWO Delete Bound default/storaged-data-nebula-storaged-0 ebs-sc 43h
	As you can see, both the PVC and PV capacity have been expanded to 10Gi.

Last update: November 15, 2023

#### Use Local Persistent Volumes in a NebulaGraph cluster

Local Persistent Volumes, abbreviated as Local PVs in K8s store container data directly using the node's local disk directory. Compared with network storage, Local Persistent Volumes provide higher IOPS and lower read and write latency, which is suitable for data-intensive applications. This topic introduces how to use Local PVs in Google Kubernetes Engine (GKE) and Amazon Elastic Kubernetes Service (EKS) clusters, and how to enable automatic failover for Local PVs in the cloud.

While using Local Persistent Volumes can enhance performance, it's essential to note that, unlike network storage, local storage does not support automatic backup. In the event of a node failure, all data in local storage may be lost. Therefore, the utilization of Local Persistent Volumes involves a trade-off between service availability, data persistence, and flexibility.

#### PRINCIPLES

NebulaGraph Operator implements a Storage Volume Provisioner interface to automatically create and delete PV objects. Utilizing the provisioner, you can dynamically generate Local PVs as required. Based on the PVC and StorageClass specified in the cluster configuration file, NebulaGraph Operator automatically generates PVCs and associates them with their respective Local PVs.

When a Local PV is initiated by the provisioner interface, the provisioner controller generates a local type PV and configures the nodeAffinity field. This configuration ensures that Pods using the local type PV are scheduled onto specific nodes. Conversely, when a Local PV is deleted, the provisioner controller eliminates the local type PV object and purges the node's storage resources.

#### PREREQUISITES

NebulaGraph Operator is installed. For details, see Install NebulaGraph Operator.

### STEPS

The resources in the following examples are all created in the default namespace.

Use Local PV on GKE Use Local PV on EKS

1. Create a node pool with local SSDs if not existing

gcloud container node-pools create "pool-1" --cluster "gke-1" --region us-central1 --node-version "1.27.10-gke.1055000" --machine-type "n2-standard-2" --local-nvme-ssd-block count=2 --maxsurge-upgrade 1 --max-unavailable-upgrade 0 --num-nodes 1 --enable-autoscaling --min-nodes 1 --max-nodes 2

For information about the parameters to create a node pool with local SSDs, see Create a node pool with Local SSD.

- 2. Format and mount the local SSDs using a DaemonSet.
- a. Download the gke-daemonset-raid-disks.yaml file.
- b. Deploy the RAID disks DaemonSet. The DaemonSet sets a RAID 0 array on all Local SSD disks and formats the device to an ext4 filesystem.

kubectl apply -f gke-daemonset-raid-disks.yaml

- 3. Deploy the Local PV provisioner.
- a. Download the local-pv-provisioner.yaml file.
- b. Run the provisioner.

kubectl apply -f local-pv-provisioner.yaml

4. In the NebulaGraph cluster configuration file, specify spec.storaged.dataVolumeClaims or spec.metad.dataVolumeClaim, and the StorageClass needs to be configured as local-nvme. For more information about cluster configurations, see Create a NebulaGraph cluster.

#### Partial configuration of the NebulaGraph cluster

```
metad:
dataVolumeClaim:
requests:
storage: 2Gi
storageClassName: local-nvme
storaged:
dataVolumeClaims:
- resources:
requests:
storage: 2Gi
storageClassName: local-nvme
...
```

After the NebulaGraph is deployed, the Local PVs are automatically created.

### 5. View the PV list.

kubectl get pv

Return:

NAME	CAPACITY	ACCESS MODES	RECLAIM POLICY	STATUS	CLAIM	STORAGECLASS	REASON	AGE
pvc-01be9b75-9c50-4532-8695-08e11b489718	5Gi	RWO	Delete	Bound	default/storaged-data-nebula-storaged-0	local-nvme		3m35s
pvc-09de8eb1-1225-4025-b91b-fbc0bcce670f	5G i	RWO	Delete	Bound	default/storaged-data-nebula-storaged-1	local-nvme		3m35s
pvc-4b2a9ffb-9000-4998-a7bb-edb825c872cb	5G i	RWO	Delete	Bound	default/storaged-data-nebula-storaged-2	local-nvme		3m35s

#### 6. View the detailed information of the PV.

kubectl get pv pvc-01be9b75-9c50-4532-8695-08e11b489718 -o yaml

#### Return:

ReadWriteOnce

```
apiVersion: v1
kind: PersistentVolume
metadata:
 annotations:
   local.pv.provisioner/selected-node: gke-snap-test-snap-test-591403a8-xdfc
   nebula-graph.io/pod-name: nebula-storaged-0
   pv.kubernetes.io/provisioned-by: nebula-cloud.io/local-pv
 creationTimestamp: "2024-03-05T06:12:32Z"
 finalizers:
   kubernetes.io/pv-protection
 labels:
   app.kubernetes.io/cluster: nebula
   app.kubernetes.io/component: storaged
   app.kubernetes.io/managed-by: nebula-operator
   app.kubernetes.io/name: nebula-graph
 name: pvc-01be9b75-9c50-4532-8695-08e11b489718
resourceVersion: "9999469"
 uid: ee28a4da-6026-49ac-819b-2075154b4724
spec:
 accessModes:
```

#### FAILOVER FOR LOCAL PERSISTENT VOLUMES IN THE CLOUD

When using network storage (e.g., AWS EBS, Google Cloud Persistent Disk, Azure Disk Storage, Ceph, NFS, etc.) as a PV, the storage resource is independent of any particular node. Therefore, the storage resource can be mounted and used by Pods regardless of the node to which the Pods are scheduled. However, when using a local storage disk as a PV, the storage resource can only be used by Pods on a specific node due to nodeAffinity.

The Storage service of NebulaGraph supports data redundancy, which allows you to set multiple odd-numbered partition replicas. When a node fails, the associated partition is automatically transferred to a healthy node. However, Storage Pods using Local Persistent Volumes cannot run on other nodes due to the node affinity setting and must wait for the node to recover. To run on another node, the Pods must be unbound from the associated Local Persistent Volume.

NebulaGraph Operator supports automatic failover in the event of a node failure while using Local Persistent Volumes in the cloud for elastic scaling. This is achieved by setting spec.enableAutoFailover to true in the cluster configuration file, which automatically unbinds the Pods from the Local Persistent Volume, allowing the Pods to run on another node.

#### Example configuration:



Last update: March 6, 2024

## **Reclaim PVs**

NebulaGraph Operator uses PVs (Persistent Volumes) and PVCs (Persistent Volume Claims) to store persistent data. If you accidentally deletes a NebulaGraph cluster, by default, PV and PVC objects and the relevant data will be retained to ensure data security.

You can also define the automatic deletion of PVCs to release data by setting the parameter spec.enablePVReclaim to true in the configuration file of the cluster instance. As for whether PV will be deleted automatically after PVC is deleted, you need to customize the PV reclaim policy. See reclaimPolicy in StorageClass and PV Reclaiming for details.

PREREQUISITES

A NebulaGraph cluster is created in Kubernetes. For specific steps, see Create a NebulaGraph cluster.

## STEPS

The following example uses a cluster named nebula and the cluster's configuration file named nebula\_cluster.yaml to show how to set enablePVReclaim:

 $_{\mbox{\scriptsize 1.}}$  Run the following command to edit the <code>nebula</code> cluster's configuration file.

kubectl edit nebulaclusters.apps.nebula-graph.io nebula

2. Add enablePVReclaim and set its value to true under spec.

```
apiVersion: apps.nebula-graph.io/v1alpha1
kind: NebulaCluster
metadata:
 name: nebula
spec:
 enablePVReclaim: true //Set its value to true.
 graphd:
    image: vesoft/nebula-graphd
    logVolumeClaim:
      resources:
       requests:
          storage: 2Gi
      storageClassName: fast-disks
    replicas: 1
    resources:
      limits:
cpu: "1"
        memory: 1Gi
      requests
        cpu: 500m
  memory: 500Mi
version: v3.6.0
imagePullPolicy: IfNotPresent
 metad:
    dataVolumeClaim:
      resources:
        requests:
      storage: 2Gi
storageClassName: fast-disks
    image: vesoft/nebula-metad
    logVolumeClaim:
      resources:
        requests:
           storage: 2Gi
    storageClassName: fast-disks
replicas: 1
    resources:
      limits:
cpu: "1"
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 500Mi
    version: v3.6.0
 nodeSelector
    nebula: cloud
 reference:
   name: statefulsets.apps
version: v1
  schedulerName: default-scheduler
 storaged:
   dataVolumeClaims:
     - resources:
        requests:
storage: 2Gi
      storageClassName: fast-disks
    - resources:
       requests:
      storage: 2Gi
storageClassName: fast-disks
    image: vesoft/nebula-storaged
    logVolumeClaim:
      resources:
       requests:
         storage: 2Gi
      storageClassName: fast-disks
    replicas: 3
    resources:
      limits:
        cpu: "1"
memory: 1Gi
      requests
        cpu: 500m
memory: 500Mi
    version: v3.6.0
```

 $3. \ Run \ {\tt kubectl apply -f nebula\_cluster.yaml} \ to \ push \ your \ configuration \ changes \ to \ the \ cluster.$ 

After setting enablePVReclaim to true, the PVCs of the cluster will be deleted automatically after the cluster is deleted. If you want to delete the PVs, you need to set the reclaim policy of the PVs to Delete.

Last update: November 15, 2023

## 17.4.4 Manage cluster logs

Running logs of NebulaGraph cluster services (graphd, metad, storaged) are generated and stored in the /usr/local/nebula/logs directory of each service container by default.

#### View logs

To view the running logs of a NebulaGraph cluster, you can use the kubectl logs command.

For example, to view the running logs of the Storage service:

#### **Clean logs**

Running logs generated by cluster services during runtime will occupy disk space. To avoid occupying too much disk space, the NebulaGraph Operator uses a sidecar container to periodically clean and archive logs.

To facilitate log collection and management, each NebulaGraph service deploys a sidecar container responsible for collecting logs generated by the service container and sending them to the specified log disk. The sidecar container automatically cleans and archives logs using the logrotate tool.

In the YAML configuration file of the cluster instance, set spec.logRotate to enable log rotation and set timestamp\_in\_logfile\_name to false to disable the timestamp in the log file name to implement log rotation for the target service. The timestamp\_in\_logfile\_name parameter is configured under the spec.<graphd|metad|storaged>.config field. By default, the log rotation feature is turned off. Here is an example of enabling log rotation for all services:

```
spec:
 graphd:
    config:
      # Whether to include a timestamp in the log file name
      # You must set this parameter to false to enable log rotation.
# It is set to true by default.
      "timestamp_in_logfile_name": "false"
  metad:
    config:
      "timestamp_in_logfile_name": "false"
  storaged:
    config
      "timestamp_in_logfile_name": "false"
  logRotate: # Log rotation configuration
     The number of times a log file is rotated before being deleted.
    # The default value is 5, and 0 means the log file will not be rotated before being deleted.
    # The log file is rotated only if it grows larger than the specified size. The default value is 200M. size: "200M"
```

## Collect logs

If you don't want to mount additional log disks to back up log files, or if you want to collect logs and send them to a log center using services like fluent-bit, you can configure logs to be output to standard error. The Operator uses the glog tool to log to standard error output.

#### O Note

Currently, NebulaGraph Operator only collects standard error logs.

In the YAML configuration file of the cluster instance, you can configure logging to standard error output in the config and env fields of each service.

```
spec:
  graphd:
      config:
# Whether to redirect standard error to a separate output file. The default value is false, which means it is not redirected.
        # medier to realized standard error to a separate output fire. The default value is facte, which means it is not
realized_standard. "false"
# The severity level of log content: INFO, WARNING, ERROR, and FATAL. The corresponding values are 0, 1, 2, and 3.
stderrthreshold: "0"
     stderrinreshold: 0
env:
- name: GLOG_Logtostderr # Write log to standard error output instead of a separate file.
value: "1" # 1 represents writing to standard error output, and 0 represents writing to a file.
image: vesoft/nebula-graphd
replicas: 1

      resources:
        requests:
           cpu: 500m
            memory: 500Mi
      service:
        externalTrafficPolicy: Local
      type: NodePort
version: v3.6.0
   metad:
      config:
   redirect_stdout: "false"
      stderrthreshold: "0"
dataVolumeClaim:
        resources:
           requests:
             storage: 1Gi
        storageClassName: ebs-sc
      env:
- name: GLOG_logtostderr
      value: "1"
image: vesoft/nebula-metad
```

Last update: November 15, 2023

## 17.4.5 Security

#### Enable admission control

Kubernetes Admission Control is a security mechanism running as a webhook at runtime. It intercepts and modifies requests to ensure the cluster's security. Admission webhooks involve two main operations: validation and mutation. NebulaGraph Operator supports only validation operations and provides some default admission control rules. This topic describes NebulaGraph Operator's default admission control rules and how to enable admission control.

PREREQUISITES

A NebulaGraph cluster is created with NebulaGraph Operator. For detailed steps, see Create a NebulaGraph cluster.

#### ADMISSION CONTROL RULES

Kubernetes admission control allows you to insert custom logic or policies before Kubernetes API Server processes requests. This mechanism can be used to implement various security policies, such as restricting a Pod's resource consumption or limiting its access permissions. NebulaGraph Operator supports validation operations, which means it validates and intercepts requests without making changes.

After admission control is enabled, NebulaGraph Operator implements the following admission validation control rules by default. You cannot disable these rules:

- $\bullet$  Forbid adding additional PVs to Storage service via <code>dataVolumeClaims</code> .
- Forbid shrinking the capacity of all service's PVCs, but allow expansion.
- · Forbid any secondary operation during Storage service scale-in/scale-out.

After admission control is enabled, NebulaGraph Operator allows you to add annotations to implement the following admission validation control rules:

- Clusters with the ha-mode annotation must have the minimum number of replicas as required by high availability mode:
- For Graph service: At least 2 replicas are required.
- For Meta service: At least 3 replicas are required.
- For Storage service: At least 3 replicas are required.

# Note

High availability mode refers to the high availability of NebulaGraph cluster services. Storage and Meta services are stateful, and the number of replicas should be an odd number due to Raft protocol requirements for data consistency. In high availability mode, at least 3 Storage services and 3 Meta services are required. Graph services are stateless, so their number of replicas can be even but should be at least 2.

• Clusters with the delete-protection annotation cannot be deleted. For more information, see Configure deletion protection.

TLS CERTIFICATES FOR ADMISSION WEBHOOKS

To ensure secure communication and data integrity between the K8s API server and the admission webhook, this communication is done over HTTPS by default. This means that TLS certificates are required for the admission webhook. cert-manager is a Kubernetes certificate management controller that automates the issuance and renewal of certificates. NebulaGraph Operator uses cert-manager to manage certificates.

Once cert-manager is installed and admission control is enabled, NebulaGraph Operator will automatically create an Issuer for issuing the necessary certificate for the admission webhook, and a Certificate for storing the issued certificate. The issued certificate is stored in the nebula-operator-webhook-secret Secret.

STEPS OF ENABLING ADMISSION CONTROL

1. Install cert-manager.

kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.13.1/cert-manager.yaml

It is suggested to deploy the latest version of cert-manager. For details, see the official cert-manager documentation.

2. Modify the NebulaGraph Operator configuration file to enable admission control. Admission control is disabled by default and needs to be enabled manually.



# Note

nebula-operator is the name of the chart repository, and nebula-operator/nebula-operator is the chart name. If the chart's namespace is not specified, it defaults to default.

3. View the certificate Secret for the admission webhook.

kubectl get secret nebula-operator-webhook-secret -o yaml

If the output includes certificate contents, it means that the admission webhook's certificate has been successfully created.

- 4. Verify the control rules.
- Verify preventing additional PVs from being added to Storage service.

\$ kubectl patch nc nebula --type='merge' --patch '{"spec": {"storaged": {"dataVolumeClaims":[{"requests": {"storage": "2Gi"}}, "storageClassName": "local-path"},{"resources": {"requests": {"storage": "3Gi"}}, "storageClassName": "fask-disks"]]}}'
Error from server: admission webhook "nebulaclustervalidating.nebula-graph.io" deniedthe request: spec.storaged.dataVolumeClaims: Forbidden: storaged dataVolumeClaims is immutable

· Verify disallowing shrinking Storage service's PVC capacity.

\$ kubectl patch nc nebula --type='merge' --patch '{"spec": {"storaged": {"dataVolumeClaims": {"requests": {"storage": "1Gi"}}, "storageClassName": "fast-disks"}]}}'
Error from server: admission webhook "nebulaclustervalidating.nebula-graph.io" denied the request: spec.storaged.dataVolumeClaims: Invalid value:
resource.Quantity{i:resource.int64Amount{value:1073741824, scale:0}, d:resource.infDecAmount{Dec:(\*inf.Dec)(nil)}, s:"1Gi", Format:"BinarySI"}: data volume size can only be increased

· Verify disallowing any secondary operation during Storage service scale-in.

\$ kubectl patch nc nebula --type='merge' --patch '{"spec": {"replicas": 5}}}'
nebulacluster.apps.nebula-graph.io/nebula patched
\$ kubectl patch nc nebula --type='merge' --patch '{"spec": {"storaged": {"replicas": 3}}'
Error from server: admission webhook "nebulaclustervalidating.nebula-graph.io" denied the request: [spec.storaged: Forbidden: field is immutable while in ScaleOut phase,
spec.storaged.replicas: Invalid value: 3: field is immutable while not in Running phase]

• Verify the minimum number of replicas in high availability mode.

# Annotate the cluster to enable high availability mode. \$ kubectl annotate nc nebula nebula-graph.io/ha-mode=true # Verify the minimum number of the Graph service's replicas. \$ kubectl patch nc nebula --type='merge' --patch '{"spec": {"graphd": {"replicas":1}}}' Error from server: admission webhook "nebulaclustervalidating.nebula-graph.io" denied the request: spec.graphd.replicas: Invalid value: 1: should be at least 2 in HA mode

• Verify deletion protection. For more information, see Configure deletion protection.

Last update: March 6, 2024

## Configure deletion protection

NebulaGraph Operator supports deletion protection to prevent NebulaGraph clusters from being deleted by accident. This topic describes how to configure deletion protection for a NebulaGraph cluster.

PREREQUISITES

- A NebulaGraph cluster is created with NebulaGraph Operator. For more information, see Create a NebulaGraph cluster.
- Admission control is enabled on the NebulaGraph cluster. For more information, see Enable admission control.

ADD AN ANNOTATION TO ENABLE DELETION PROTECTION

Add the delete-protection annotation to the cluster.

kubectl annotate nc nebula -n nebula-test nebula-graph.io/delete-protection=true

The preceding command enables deletion protection for the nebula cluster in the nebula-test namespace.

VERIFY DELETION PROTECTION

To verify that deletion protection is enabled, run the following command:

kubectl delete nc nebula -n nebula-test

The preceding command attempts to delete the nebula cluster in the nebula-test namespace.

#### Return:

Error from server: admission webhook "nebulaclustervalidating.nebula-graph.io" denied the request: metadata.annotations[nebula-graph.io/delete-protection]: Forbidden: protected cluster cannot be deleted

REMOVE THE ANNOTATION TO DISABLE DELETION PROTECTION

Remove the delete-protection annotation from the cluster as follows:

 ${\tt kubectl\ annotate\ nc\ nebula\ -n\ nebula-test\ nebula-graph.io/delete-protection-}$ 

The preceding command disables deletion protection for the nebula cluster in the nebula-test namespace.

Last update: March 6, 2024

## 17.4.6 HA and balancing

## Self-healing

NebulaGraph Operator calls the interface provided by NebulaGraph clusters to dynamically sense cluster service status. Once an exception is detected (for example, a component in a NebulaGraph cluster stops running), NebulaGraph Operator automatically performs fault tolerance. This topic shows how Nebular Operator performs self-healing by simulating cluster failure of deleting one Storage service Pod in a NebulaGraph cluster.

PREREQUISITES

Install NebulaGraph Operator

STEPS

- 1. Create a NebulaGraph cluster. For more information, see Create a NebulaGraph clusters.
- 2. Delete the Pod named <cluster\_name>-storaged-2 after all pods are in the Running status.

kubectl delete pod <cluster-name>-storaged-2 --now

<cluster\_name> is the name of your NebulaGraph cluster.

3. NebulaGraph Operator automates the creation of the Pod named <cluster-name>-storaged-2 to perform self-healing.

Run the kubectl get pods command to check the status of the Pod <cluster-name>-storaged-2.

 nebula-cluster-storaged-1 nebula-cluster-storaged-2 	1/1 0/1	Running ContainerCr	eating	0 0		5d23h 1s
 nebula-cluster-storaged-1 nebula-cluster-storaged-2 	1/1 1/1	Running Running	0 0	1	5d23h 4m2s	

When the status of <cluster-name>-storaged-2 is changed from ContainerCreating to Running, the self-healing is performed successfully.

Last update: November 15, 2023

## 17.4.7 Advanced

#### Optimize leader transfer in rolling updates

NebulaGraph clusters use a distributed architecture to divide data into multiple logical partitions, which are typically evenly distributed across different nodes. In distributed systems, there are usually multiple replicas of the same data. To ensure the consistency of data across multiple replicas, NebulaGraph clusters use the Raft protocol to synchronize multiple partition replicas. In the Raft protocol, each partition elects a leader replica, which is responsible for handling write requests, while follower replicas handle read requests.

When a NebulaGraph cluster created by NebulaGraph Operator performs a rolling update, a storage node temporarily stops providing services for the update. For an overview of rolling updates, see Performing a Rolling Update. If the node hosting the leader replica stops providing services, it will result in the unavailability of read and write operations for that partition. To avoid this situation, by default, NebulaGraph Operator transfers the leader replicas to other unaffected nodes during the rolling update process of a NebulaGraph cluster. This way, when a storage node is being updated, the leader replicas on other nodes can continue processing client requests, ensuring the read and write availability of the cluster.

The process of migrating all leader replicas from one storage node to the other nodes may take a long time. To better control the rolling update duration, Operator provides a field called <code>enableForceUpdate</code>. When it is confirmed that there is no external access traffic, you can set this field to <code>true</code>. This way, the leader replicas will not be transferred to other nodes, thereby speeding up the rolling update process.

ROLLING UPDATE TRIGGER CONDITIONS

Operator triggers a rolling update of the NebulaGraph cluster under the following circumstances:

- The version of the NebulaGraph cluster changes.
- The configuration of the NebulaGraph cluster changes.
- NebulaGraph cluster services are restarted.

#### SPECIFY A ROLLING UPDATE STRATEGY

In the YAML file for creating a cluster instance, add the spec.storaged.enableForceUpdate field and set it to true or false to control the rolling update speed.

When enableForceUpdate is set to true, it means that the leader partition replicas are not transferred, thus speeding up the rolling update process. Conversely, when set to false, it means that the leader replicas are transferred to other nodes to ensure the read and write availability of the cluster. The default value is false.

## Marning

When setting enableForceUpdate to true, make sure there is no traffic entering the cluster for read and write operations. This is because this setting will force the cluster pods to be rebuilt, and during this process, data loss or client request failures may occur.

### Configuration example:

1111	
spec	
st	<pre>praged: # When set to true, # it means that the leader partition replicas are not transferred # but the cluster pods are rebuilt directly. enableForceUpdate: true </pre>

Last update: March 6, 2024

## Restart service Pods in a NebulaGraph cluster on K8s

#### O Note

Restarting NebulaGraph cluster service Pods is a feature in the Alpha version.

During routine maintenance, it might be necessary to restart a specific service Pod in the NebulaGraph cluster, for instance, when the Pod's status is abnormal or to enforce a restart. Restarting a Pod essentially means restarting the service process. To ensure high availability, NebulaGraph Operator supports gracefully restarting all Pods of the Graph, Meta, or Storage service respectively and gracefully restarting an individual Pod of the Storage service.

PREREQUISITES

A NebulaGraph cluster is created in a K8s environment. For details, see Create a NebulaGraph cluster.

RESTART ALL PODS OF A CERTAIN SERVICE TYPE

To gracefully roll restart all Pods of a certain service type in the cluster, you can add an annotation (nebula-graph.io/restart-timestamp) with the current time to the configuration of the StatefulSet controller of the corresponding service.

When NebulaGraph Operator detects that the StatefulSet controller of the corresponding service has the annotation nebulagraph.io/restart-timestamp and its value is changed, it triggers the graceful rolling restart operation for all Pods of that service type in the cluster.

In the following example, the annotation is added for all Graph services so that all Pods of these Graph services are restarted one by one.

Assume that the cluster name is nebula and the cluster resources are in the default namespace. Run the following command:

1. Check the name of the StatefulSet controller.

kubectl get statefulset

Sample output:

NAME	READY	AGE	
nebula-graphd	<mark>2</mark> /2	33s	
nebula-metad	<mark>3</mark> /3	69s	
nebula-storaged	3/3	69s	

#### 2. Get the current timestamp.

date -u +%s

Example output:

1700547115

3. Overwrite the timestamp annotation of the StatefulSet controller to trigger the graceful rolling restart operation.

kubectl annotate statefulset nebula-graphd nebula-graph.io/restart-timestamp="1700547115" -- overwrite

#### Example output:

statefulset.apps/nebula-graphd annotate

#### 4. Observe the restart process.

kubectl get pods -l app.kubernetes.io/cluster=nebula,app.kubernetes.io/component=graphd -w

#### Example output:

NAME	READY	STATUS RESTARTS	AGE
nebula-graphd-0	<b>1</b> /1	Runn i ng 🛛 0	9m37s
nebula-graphd-1	<mark>0</mark> /1	Runn i ng 🛛 0	17s
nebula-graphd-1	<b>1</b> /1	Runn i ng 🛛 0	20s
nebula-graphd-0	<b>1</b> /1	Terminating 0	9m40s
nebula-graphd-0	<mark>0</mark> /1	Terminating 0	9m41s
nebula-graphd-0	<mark>0</mark> /1	Terminating 0	9m42s
nebula-graphd-0	<mark>0</mark> /1	Terminating 0	9m42s
nebula-graphd-0	<mark>0</mark> /1	Terminating 0	9m42s
nebula-graphd-0	<mark>0</mark> /1	Pending 0	0s
nebula-graphd-0	<mark>0</mark> /1	Pending 0	0s
nebula-graphd-0	<mark>0</mark> /1	ContainerCreating	0 0s
nebula-graphd-0	0/1	Running	0 29

This above output shows the status of Graph service Pods during the restart process.

## 5. Verify that the StatefulSet controller annotation is updated.

kubectl get statefulset nebula-graphd -o yaml | grep "nebula-graph.io/restart-timestamp"

#### Example output:

nebula-graph.io/Last-applied-configuration: '{"persistentVolumeClaimRetentionPolicy": {"whenDeleted": "Retain", "whenScaled": "Retain"}, "podManagementPolicy": "Parallel", "replicas": 2, "revisionHistoryLimit": 10, "selector": {"matchLabels": {"app.kubernetes.io/cluster": "nebula", "app.kubernetes.io/component": "graphd", "app.kubernetes.io/nanaged-by": "nebulaoperator", "app.kubernetes.io/name": "nebula-graph"}, "serviceName": "nebula-graphd-headless", "template": {"metadata": {"annotations": {"nebula-graph.io/cm-hash": "7c55c0e5ac74e85f", "nebula-graph.io/ restart-timestamp": "1700547815"}, "creationTimestamp": mll, "Labels": {"app.kubernetes.io/cluster": "nebula", "app.kubernetes.io/component": "graphd", "app.kubernetes.io/managed-by": "nebulaoperator", "app.kubernetes.io/name": "nebula-graph"}, "spec": {"containers": [{"command": ["/bin/sh", "-ecx", "exec nebula-graph.io/restart-timestamp: "1700547815" nebula-graph.io/restart-timestamp: "1700547815"

The above output indicates that the annotation of the StatefulSet controller has been updated, and all graph service Pods has been restarted.

#### RESTART A SINGLE STORAGE SERVICE POD

To gracefully roll restart a single Storage service Pod, you can add an annotation (nebula-graph.io/restart-ordinal) with the value set to the ordinal number of the Storage service Pod you want to restart. This triggers a graceful restart or state transition for that specific Storage service Pod. The added annotation will be automatically removed after the Storage service Pod is restarted.

In the following example, the annotation is added for the Pod with ordinal number 1, indicating a graceful restart for the nebulastoraged-1 Storage service Pod.

Assume that the cluster name is nebula, and the cluster resources are in the default namespace. Run the following commands:

#### 1. Check the name of the StatefulSet controller.

kubectl	get	statefu	lset

## Example output:

NAME	READY	AGE	
nebula-graphd	<mark>2</mark> /2	33s	
nebula-metad	<mark>3</mark> /3	69s	
nebula-storaged	<mark>3</mark> /3	69s	

2. Get the ordinal number of the Storage service Pod.

kubectl get pods -l app.kubernetes.io/cluster=nebula,app.kubernetes.io/component=storaged

#### Example output:

NAME	READY	STATUS	RESTARTS	AGE
nebula-storaged-0	<mark>1</mark> /1	Running	0	13h
nebula-storaged-1	<mark>1</mark> /1	Running	0	13h
nebula-storaged-2	<b>1</b> /1	Running	0	13h
nebula-storaged-3	<b>1</b> /1	Running	0	13h
nebula-storaged-4	<mark>1</mark> /1	Running	0	13h
nebula-storaged-5	<b>1</b> /1	Running	0	13h
nebula-storaged-6	<b>1</b> /1	Running	0	13h
nebula-storaged-7	<b>1</b> /1	Running	0	13h
nebula-storaged-8	1/1	Running	0	13h

3. Add the annotation for the nebula-storaged-1 Pod to trigger a graceful restart for that specific Pod.

kubectl annotate statefulset nebula-storaged nebula-graph.io/restart-ordinal="1"

#### Example output:

statefulset.apps/nebula-storaged annotate

## 4. Observe the restart process.

kubectl get pods -l app.kubernetes.io/cluster=nebula,app.kubernetes.io/component=storaged -w

#### Example output:

NAME	READY	STATUS	RESTARTS	AGE		
nebula-storaged-0	<b>1</b> /1	Running	0	13h		
nebula-storaged-1	1/1	Running	0	13h		
nebula-storaged-2	<b>1</b> /1	Running	0	13h		
nebula-storaged-3	1/1	Running	0	13h		
nebula-storaged-4	<b>1</b> /1	Running	0	13h		
nebula-storaged-5	<b>1</b> /1	Running	0	12h		
nebula-storaged-6	<b>1</b> /1	Running	0	12h		
nebula-storaged-7	1/1	Running	0	12h		
nebula-storaged-8	<b>1</b> /1	Running	0	12h		
nebula-storaged-1	1/1	Runn i ng	0	13h		
nebula-storaged-1	<b>1</b> /1	Terminatir	ng <mark>0</mark>		13h	
nebula-storaged-1	0/1	Terminatir	ng <mark>0</mark>		13h	
nebula-storaged-1	0/1	Terminatir	ng <mark>0</mark>		13h	
nebula-storaged-1	0/1	Terminatir	ng <mark>0</mark>		13h	
nebula-storaged-1	0/1	Terminatir	ng <mark>0</mark>		13h	
nebula-storaged-1	0/1	Pending	0		0s	
nebula-storaged-1	0/1	Pending	0		0s	
nebula-storaged-1	0/1	Container	Creating	0		0s
nebula-storaged-1	0/1	Running		0		1s
nebula-storaged-1	<b>1</b> /1	Running		0		10s

The above output indicates that the nebula-storaged-1 Storage service Pod is successfully restarted.

After restarting a single Storage service Pod, the distribution of storage leader replicas may become unbalanced. You can execute the BALANCE LEADER command to rebalance the distribution of leader replicas. For information about how to view the leader distribution, see SHOW HOSTS.

Last update: March 7, 2024

# 17.5 FAQ

## 17.5.1 Does NebulaGraph Operator support the v1.x version of NebulaGraph?

No, because the v1.x version of NebulaGraph does not support DNS, and NebulaGraph Operator requires the use of DNS.

## 17.5.2 Is cluster stability guaranteed if using local storage?

There is no guarantee. Using local storage means that the Pod is bound to a specific node, and NebulaGraph Operator does not currently support failover in the event of a failure of the bound node.

## 17.5.3 How to ensure the stability of a cluster when scaling the cluster?

It is suggested to back up data in advance so that you can roll back data in case of failure.

## 17.5.4 Is the replica in the Operator docs the same as the replica in the NebulaGraph core docs?

They are different concepts. A replica in the Operator docs indicates a pod replica in K8s, while a replica in the core docs is a replica of a NebulaGraph storage partition.

## 17.5.5 How to view the logs of each service in the NebulaGraph cluster?

To obtain the logs of each cluster service, you need to access the container and view the log files that are stored inside.

Steps to view the logs of each service in the NebulaGraph cluster:

# To view the name of the pod where the container you want to access is located.
# Replace <cluster-name> with the name of the cluster.
kubectl get pods -l app.kubernetes.io/cluster=<cluster-name>

# To access the container within the pod, such as the nebula-graphd-0 container. <code>kubectl exec -it nebula-graphd-0 -- /bin/bash</code>

# To go to /usr/local/nebula/logs directory to view the logs. cd /usr/local/nebula/logs
# 17.5.6 How to resolve the host not found:nebula-<metad|storaged|graphd>-0.nebula.<metad|storaged|graphd>- headless.default.svc.cluster.local error?

This error is generally caused by a DNS resolution failure, and you need to check whether the cluster domain has been modified. If the cluster domain has been modified, you need to modify the kubernetesClusterDomain field in the NebulaGraph Operator configuration file accordingly. The steps for modifying the Operator configuration file are as follows:

#### 1. View the Operator configuration file.

```
[abby@master ~]$ helm show values nebula-operator/nebula-operator
image:
    nebulaOperator:
    image: vesoft/nebula-operator:v1.8.0
    imagePullPolicy: Always
kubeRBACProxy:
    imagePullPolicy: Always
kubeScheduler:
    imagePullPolicy: Always
imagePullPolicy: Always
imagePullSecrets: []
kubernetesClusterDomain: "" # The cluster domain name, and the default is cluster.local.
```

## 2. Modify the value of the kubernetesClusterDomain field to the updated cluster domain name.

helm upgrade nebula-operator nebula-operator/nebula-operator --namespace=<nebula-operator-system> --version=1.8.0 --set kubernetesClusterDomain=<cluster-domain>

is the namespace where Operator is located and is the updated domain name.

```
Last update: November 15, 2023
```

# 18. Graph computing

# 18.1 NebulaGraph Algorithm

NebulaGraph Algorithm (Algorithm) is a Spark application based on GraphX. It uses a complete algorithm tool to perform graph computing on the data in the NebulaGraph database by submitting a Spark task. You can also programmatically use the algorithm under the lib repository to perform graph computing on DataFrame.

# 18.1.1 Version compatibility

The correspondence between the NebulaGraph Algorithm release and the NebulaGraph core release is as follows.

NebulaGraph	NebulaGraph Algorithm
nightly	3.0-SNAPSHOT
$3.0.0 \sim 3.4.x$	3.x.0
2.6.x	2.6.x
2.5.0 \ 2.5.1	2.5.0
2.0.0 \ 2.0.1	2.1.0

# 18.1.2 Prerequisites

Before using the NebulaGraph Algorithm, users need to confirm the following information:

- The NebulaGraph services have been deployed and started. For details, see NebulaGraph Installation.
- The Spark version is 2.4.x.
- The Scala version is 2.11.
- (Optional) If users need to clone, compile, and package the latest Algorithm in Github, install Maven.

# 18.1.3 Limitations

Graph computing outputs vertex datasets, and the algorithm results are stored in DataFrames as the properties of vertices. You can do further operations such as statistics and filtering according to your business requirements.

!!!

Before Algorithm v3.1.0, when submitting the algorithm package directly, the data of the vertex ID must be an integer. That is, the vertex ID can be INT or String, but the data itself is an integer.

# 18.1.4 Supported algorithms

The graph computing algorithms supported by NebulaGraph Algorithm are as follows.

Algorithm	Description	Scenario	Properties name	Properties type
PageRank	The rank of pages	Web page ranking, key node mining	pagerank	double/ string
Louvain	Louvain	Community mining, hierarchical clustering	louvain	int/string
KCore	K core	Community discovery, financial risk control	kcore	int/string
LabelPropagation	Label propagation	Information spreading, advertising, and community discovery	lpa	int/string
Hanp	Label propagation advanced	Community discovery, recommendation system	hanp	int/string
ConnectedComponent	Weakly connected component	Community discovery, island discovery	CC	int/string
StronglyConnectedComponent	Strongly connected component	Community discovery	SCC	int/string
ShortestPath	The shortest path	Path planning, network planning	shortestpath	string
TriangleCount	Triangle counting	Network structure analysis	trianglecount	int/string
GraphTriangleCount	Graph triangle counting	Network structure and tightness analysis	count	int
BetweennessCentrality	Intermediate centrality	Key node mining, node influence computing	betweenness	double/ string
ClosenessCentrality	Closeness centrality	Key node mining, node influence computing	closeness	double/ string
DegreeStatic	Degree of statistical	Graph structure analysis	degree,inDegree,outDegree	int/string
ClusteringCoefficient	Aggregation coefficient	Recommendation system, telecom fraud analysis	clustercoefficient	double/ string
Jaccard			jaccard	string

Algorithm	Description	Scenario	Properties name	Properties type
	Jaccard similarity	Similarity computing, recommendation system		
BFS	Breadth- First Search	Sequence traversal, shortest path planning	bfs	string
DFS	Depth-First Search	Sequence traversal, shortest path planning	dfs	string
Node2Vec	-	Graph classification	node2vec	string

# Note

When writing the algorithm results into the NebulaGraph, make sure that the tag in the corresponding graph space has properties names and data types corresponding to the table above.

# 18.1.5 Implementation methods

NebulaGraph Algorithm implements the graph calculating as follows:

- 1. Read the graph data of DataFrame from the NebulaGraph database using the NebulaGraph Spark Connector.
- 2. Transform the graph data of DataFrame to the GraphX graph.
- 3. Use graph algorithms provided by GraphX (such as PageRank) or self-implemented algorithms (such as Louvain).

For detailed implementation methods, see Scala file.

# 18.1.6 Get NebulaGraph Algorithm

## Compile and package

1. Clone the repository nebula-algorithm.

\$ git clone -b v3.0.0 https://github.com/vesoft-inc/nebula-algorithm.git

2. Enter the directory nebula-algorithm.

\$ cd nebula-algorithm

3. Compile and package.

\$ mvn clean package -Dgpg.skip -Dmaven.javadoc.skip=true -Dmaven.test.skip=true

After the compilation, a similar file nebula-algorithm-3.x.x.jar is generated in the directory nebula-algorithm/target.

#### Download maven from the remote repository

### Download address

#### 18.1.7 How to use



### Use algorithm interface (recommended)

The lib repository provides 10 common graph algorithms.

1. Add dependencies to the file pom.xmt.

```
<dependency>
<groupId>com.vesoft</groupId>
<artifactId>nebula-algorithm</artifactId>
<version>3.0.0</version>
</dependency>
```

2. Use the algorithm (take PageRank as an example) by filling in parameters. For more examples, see example.

# Note

By default, the DataFrame that executes the algorithm sets the first column as the starting vertex, the second column as the destination vertex, and the third column as the edge weights (not the rank in the NebulaGraph).

```
val prConfig = new PRConfig(5, 1.0)
val prResult = PageRankAlgo.apply(spark, data, prConfig, false)
```

If your vertex IDs are Strings, see Pagerank Example for how to encoding and decoding them.

#### Submit the algorithm package directly

1. Set the Configuration file.

```
# Configurations related to Spark
spark: {
  app: {
      name: LPA
      # The number of partitions of Spark
      partitionNum:100
  master:local
data: {
  # Data source. Optional values are nebula, csv, and json.
  source: csv
  # Data sink. The algorithm result will be written into this sink. Optional values are nebula, csv, and text.
  sink: nebula
  # Whether the algorithm has a weight.
hasWeight: false
# Configurations related to NebulaGraph
nebula: {
  # Data source. When NebulaGraph is the data source of the graph computing, the configuration of `nebula.read` is valid.
  read:
      # The IP addresses and ports of all Meta services. Multiple addresses are separated by commas (,). Example: "ip1:port1,ip2:port2".
      # To deploy NebulaGraph by using Docker Compose, fill in the port with which Docker Compose maps to the outside.
       # Check the status with `docker-compose ps`
      metaAddress: "192.168.*.10:9559"
# The name of the graph space in NebulaGraph.
      space: basketballplayer
      # Edge types in NebulaGraph. When there are multiple labels, the data of multiple edges will be merged. 
labels: ["serve"]
      # The property name of each edge type in NebulaGraph. This property will be used as the weight column of the algorithm. Make sure that it corresponds to the edge type
```

weightCols: ["start\_year"] # Data sink. When the graph computing result sinks into NebulaGraph, the configuration of `nebula.write` is valid. write: # The IP addresses and ports of all Graph services. Multiple addresses are separated by commas (,). Example: "ip1:port1,ip2:port2". # To deploy by using Docker Compose, fill in the port with which Docker Compose maps to the outside. # Check the status with `docker-compose ps`. graphAddress: "192.168.\*.11:9669" # The IP addresses and ports of all Meta services. Multiple addresses are separated by commas (,). Example: "ip1:port1,ip2:port2". # To deploy NebulaGraph by using Docker Compose, fill in the port with which Docker Compose maps to the outside. # Check the staus with `docker-compose ps`. metaAddress: "192.168.\*.12:9559" user:root pswd:nebula . # Before submitting the graph computing task, create the graph space and tag. # The name of the graph space in NebulaGraph. space:nb The name of the tag in NebulaGraph. The graph computing result will be written into this tag. The property name of this tag is as follows. # PageRank: pagerank
# Louvain: louvain # ConnectedComponent: cc # StronglyConnectedComponent: scc
# LabelPropagation: lpa # ShortestPath: shortestpath # DegreeStatic: degree, inDegree, outDegree # KCore: kcore # TriangleCount: tranglecpunt
# BetweennessCentrality: betweennedss tag:pagerank local: { # Data source. When the data source is csv or json, the configuration of `local.read` is valid. read:{ filePath: "hdfs://127.0.0.1:9000/edge/work for.csv" # If the CSV file has a header or it is a json file, use the header. If not, use [\_c0, \_c1, \_c2, ..., \_cn] instead. # The header of the source VID column.
srcId:"\_c0" # The header of the destination VID column dstId:"\_c1"
# The header of the weight column. weight: "\_c2" # Whether the csv file has a header. header: false # The delimiter in the csv file delimiter:", # Data sink. When the graph computing result sinks to the csv or text file, the configuration of `local.write` is valid. write:{ resultPath:/tmp/ algorithm: { # The algorithm to execute. Optional values are as follow: # pagerank, louvain, connectedcomponent, labelpropagation, shortestpaths, # degreestatic, kcore, stronglyconnectedcomponent, trianglecount , # betweenness, graphtriangleCount. executeAlgo: pagerank # PageRank pagerank: maxIter: 10 resetProb: 0.15 encodeId:false # Configure true if the VID is of string type. # Louvain louvain: { maxIter: 20 internalIter: 10 tol: 0.5 encodeId:false # Configure true if the VID is of string type. # ...

#### Q Note

When sink: nebula is configured, it means that the algorithm results will be written back to the NebulaGraph cluster. The property names of the tag have implicit conventions. For details, see **Supported algorithms** section of this topic.

# $_{2.}$ Submit the graph computing task.

\${SPARK\_HONE}/bin/spark-submit --master <mode> --class com.vesoft.nebula.algorithm.Main <nebula-algorithm-3.0.0.jar\_path> -p <application.conf\_path>

# Example:

\${SPARK.HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.algorithm.Main /root/nebula-algorithm/target/nebula-algorithm-3.0-SNAPSHOT.jar -p /root/nebula-algorithm/src/main/ resources/application.conf

Last update: December 18, 2023

# 19. NebulaGraph Bench

NebulaGraph Bench is a performance test tool for NebulaGraph using the LDBC data set.

# 19.1 Scenario

- Generate test data and import NebulaGraph.
- Performance testing in the NebulaGraph cluster.

# 19.2 Release note

Release

# 19.3 Test process

For detailed usage instructions, see NebulaGraph Bench.

Last update: October 25, 2023

# 20. FAQ

This topic lists the frequently asked questions for using NebulaGraph 3.6.0. You can use the search box in the help center or the search function of the browser to match the questions you are looking for.

If the solutions described in this topic cannot solve your problems, ask for help on the NebulaGraph forum or submit an issue on GitHub issue.

# 20.1 About manual updates

20.1.1 "Why is the behavior in the manual not consistent with the system?"

NebulaGraph is still under development. Its behavior changes from time to time. Users can submit an issue to inform the team if the manual and the system are not consistent.

# Note

If you find some errors in this topic:

. Click the pencil button at the top right side of this page.

2. Use markdown to fix this error. Then click "Commit changes" at the bottom, which will start a Github pull request.

3. Sign the CLA. This pull request will be merged after the acceptance of at least two reviewers.

# 20.2 About legacy version compatibility

# **P**\_mpatibility

Neubla Graph 3.6.0 is **not compatible** with NebulaGraph 1.x nor 2.0-RC in both data formats and RPC-protocols, and **vice versa**. The service process may **quit** if using an **lower version** client to connect to a **higher version** server.

To upgrade data formats, see Upgrade NebulaGraph to the current version. Users must upgrade all clients.

# 20.3 About execution errors

20.3.1 "How to resolve the error -1005:GraphMemoryExceeded: (-2600) ?"

This error is issued by the Memory Tracker when it observes that memory usage has exceeded a set threshold. This mechanism can help avoid service processes from being terminated by the system's OOM (Out of Memory) killer. Steps to resolve:

- 1. Check memory usage: First, you need to check the memory usage during the execution of the command. If the memory usage is indeed high, then this error might be expected.
- 2. Check the configuration of the Memory Tracker: If the memory usage is not high, check the relevant configurations of the Memory Tracker. These include memory\_tracker\_untracked\_reserved\_memory\_mb (untracked reserved memory in MB), memory\_tracker\_limit\_ratio (memory limit ratio), and memory\_purge\_enabled (whether memory purge is enabled). For the configuration of the Memory Tracker, see memory tracker configuration.
- 3. Optimize configurations: Adjust these configurations according to the actual situation. For example, if the available memory limit is too low, you can increase the value of memory\_tracker\_limit\_ratio.

# 20.3.2 "How to resolve the error SemanticError: Missing yield clause. ?"

Starting with NebulaGraph 3.0.0, the statements LOOKUP , GO , and FETCH must output results with the YIELD clause. For more information, see YIELD.

#### 20.3.3 "How to resolve the error Host not enough! ?"

From NebulaGraph version 3.0.0, the Storage services added in the configuration files **CANNOT** be read or written directly. The configuration files only register the Storage services into the Meta services. You must run the ADD HOSTS command to read and write data on Storage servers. For more information, see Manage Storage hosts.

20.3.4 "How to resolve the error To get the property of the vertex in 'v.age', should use the format 'var.tag.prop'?"

From NebulaGraph version 3.0.0, patterns support matching multiple tags at the same time, so you need to specify a tag name when querying properties. The original statement RETURN variable\_name.property\_name is changed to RETURN variable\_name.stag\_name>.property\_name .

#### 20.3.5 "How to resolve Used memory hits the high watermark(0.800000) of total system memory. ?"

The error may be caused if the system memory usage is higher than the threshold specified by system\_memory\_high\_watermark\_ratio, which defaults to 0.8. When the threshold is exceeded, an alarm is triggered and NebulaGraph stops processing queries.

Possible solutions are as follows:

- Clean the system memory to make it below the threshold.
- Modify the Graph configuration. Add the system\_memory\_high\_watermark\_ratio parameter to the configuration files of all Graph servers, and set it greater than 0.8, such as 0.9.

However, the system\_memory\_high\_watermark\_ratio parameter is deprecated. It is recommended that you use the Memory Tracker feature instead to limit the memory usage of Graph and Storage services. For more information, see Memory Tracker for Graph service and Memory Tracker for Storage service.

# 20.3.6 "How to resolve the error Storage Error E\_RPC\_FAILURE ?"

The reason for this error is usually that the storaged process returns too many data back to the graphd process. Possible solutions are as follows:

- Modify configuration files: Modify the value of --storage\_client\_timeout\_ms in the nebula-graphd.conf file to extend the connection timeout of the Storage client. This configuration is measured in milliseconds (ms). For example, set -- storage\_client\_timeout\_ms=60000. If this parameter is not specified in the nebula-graphd.conf file, specify it manually. Tip: Add -- local\_config=true at the beginning of the configuration file and restart the service.
- Optimize the query statement: Reduce queries that scan the entire database. No matter whether LIMIT is used to limit the number of returned results, use the 60 statement to rewrite the MATCH statement (the former is optimized, while the latter is not).
- Check whether the Storaged process has OOM. ( dmesg |grep nebula ).
- Use better SSD or memory for the Storage Server.
- Retry.

#### 20.3.7 "How to resolve the error The leader has changed. Try again later ?"

It is a known issue. Just retry 1 to N times, where N is the partition number. The reason is that the meta client needs some heartbeats to update or errors to trigger the new leader information.

If this error occurs when logging in to NebulaGraph, you can consider using df -h to view the disk space and check whether the local disk is full.

#### 20.3.8 "How to resolve Schema not exist: xxx ?"

If the system returns Schema not exist when querying, make sure that:

- Whether there is a tag or an edge type in the Schema.
- Whether the name of the tag or the edge type is a keyword. If it is a keyword, enclose them with backquotes (`). For more information, see Keywords.

#### 20.3.9 Unable to download SNAPSHOT packages when compiling Exchange, Connectors, or Algorithm

Problem description: The system reports Could not find artifact com.vesoft:client:jar:xxx-SNAPSHOT when compiling.

Cause: There is no local Maven repository for storing or downloading SNAPSHOT packages. The default central repository in Maven only stores official releases, not development versions (SNAPSHOTs).

Solution: Add the following configuration in the profiles scope of Maven's setting.xml file:



#### 20.3.10 "How to resolve [ERROR (-1004)]: SyntaxError: syntax error near ?"

In most cases, a query statement requires a YIELD or a RETURN. Check your query statement to see if YIELD or RETURN is provided.

#### 20.3.11 "How to resolve the error can't solve the start vids from the sentence ?"

The graphd process requires start vids to begin a graph traversal. The start vids can be specified by the user. For example:

```
> G0 FROM ${vids} ...
> MATCH (src) WHERE id(src) == ${vids}
# The "start vids" are explicitly given by ${vids}.
```

It can also be found from a property index. For example:

# CREATE TAG INDEX IF NOT EXISTS i\_player ON player(name(20)); # REBUILD TAG INDEX i\_player; > LOOKUP ON player WHERE player.name == "abc" | ... YIELD ... > MATCH (src) WHERE src.name == "abc" ... # The "start vids" are found from the property index "name".

Otherwise, an error like can't solve the start vids from the sentence will be returned.

#### 20.3.12 "How to resolve the error Wrong vertex id type: 1001 ?"

Check whether the VID is INT64 or FIXED\_STRING(N) set by create space. For more information, see create space.

## 20.3.13 "How to resolve the error The VID must be a 64-bit integer or a string fitting space vertex id length limit. ?"

Check whether the length of the VID exceeds the limitation. For more information, see create space.

## 20.3.14 "How to resolve the error edge conflict or vertex conflict ?"

NebulaGraph may return such errors when the Storage service receives multiple requests to insert or update the same vertex or edge within milliseconds. Try the failed requests again later.

#### 20.3.15 "How to resolve the error RPC failure in MetaClient: Connection refused ?"

The reason for this error is usually that the metad service status is unusual, or the network of the machine where the metad and graphd services are located is disconnected. Possible solutions are as follows:

- Check the metad service status on the server where the metad is located. If the service status is unusual, restart the metad service.
- Use telnet meta-ip:port to check the network status under the server that returns an error.
- Check the port information in the configuration file. If the port is different from the one used when connecting, use the port in the configuration file or modify the configuration.

20.3.16 "How to resolve the error StorageClientBase.inl:214] Request to "x.x.x.x":9779 failed: N6apache6thrift9transport19TTransportExceptionE: Timed Out in nebula-graph.INFO ?"

The reason for this error may be that the amount of data to be queried is too large, and the storaged process has timed out. Possible solutions are as follows:

- When importing data, set Compaction manually to make read faster.
- Extend the RPC connection timeout of the Graph service and the Storage service. Modify the value of --storage\_client\_timeout\_ms in the nebula-graphd.conf file. This configuration is measured in milliseconds (ms). The default value is 60000ms.

20.3.17 "How to resolve the error MetaClient.cpp:65] Heartbeat failed, status:Wrong cluster! in nebula-storaged.INF0, or HBProcessor.cpp:54] Reject wrong cluster host "x.x.x.x":9771! in nebula-metad.INF0?"

The reason for this error may be that the user has modified the IP or the port information of the metad process, or the storage service has joined other clusters before. Possible solutions are as follows:

Delete the cluster.id file in the installation directory where the storage machine is deployed (the default installation directory is /usr/local/nebula ), and restart the storaged service.

#### 20.3.18 "How to resolve the error

Storage Error: More than one request trying to add/update/delete one edge/vertex at he same time.?"

The reason for this error is that the current NebulaGraph version does not support concurrent requests to the same vertex or edge at the same time. To solve this error, re-execute your commands.

# 20.4 About design and functions

20.4.1 "How is the time spent value at the end of each return message calculated?"

Take the returned message of SHOW SPACES as an example:

nebula>	SHOW	SPACES;	
+			+
Name			
+			+
"baske	etball	.player"	L
			÷.,

Got 1 rows (time spent 1235/1934 us)

- The first number 1235 shows the time spent by the database itself, that is, the time it takes for the query engine to receive a query from the client, fetch the data from the storage server, and perform a series of calculations.
- The second number 1934 shows the time spent from the client's perspective, that is, the time it takes for the client from sending a request, receiving a response, and displaying the result on the screen.

# 20.4.2 "Why does the port number of the nebula-storaged process keep showing red after connecting to NebulaGraph?"

Because the nebula-storaged process waits for nebula-metad to add the current Storage service during the startup process. The Storage works after it receives the ready signal. Starting from NebulaGraph 3.0.0, the Meta service cannot directly read or write data in the Storage service that you add in the configuration file. The configuration file only registers the Storage service to the Meta service. You must run the ADD HOSTS command to enable the Meta to read and write data in the Storage service. For more information, see Manage Storage hosts.

#### 20.4.3 "Why is there no line separating each row in the returned result of NebulaGraph 2.6.0?"

This is caused by the release of NebulaGraph Console 2.6.0, not the change of NebulaGraph core. And it will not affect the content of the returned data itself.

#### 20.4.4 About dangling edges

A dangling edge is an edge that only connects to a single vertex and only one part of the edge connects to the vertex.

Dangling edges may appear in NebulaGraph 3.6.0 as the design. And there is no WERGE statements of openCypher. The guarantee for dangling edges depends entirely on the application level. For more information, see INSERT VERTEX, DELETE VERTEX, INSERT EDGE, DELETE EDGE.

#### 20.4.5 "Can I set replica\_factor as an even number in CREATE SPACE statements, e.g., replica\_factor = 2?"

NO.

The Storage service guarantees its availability based on the Raft consensus protocol. The number of failed replicas must not exceed half of the total replica number.

When the number of machines is 1, replica\_factor can only be set to 1.

When there are enough machines and replica\_factor=2, if one replica fails, the Storage service fails. No matter replica\_factor=3 or replica\_factor=4, if more than one replica fails, the Storage Service fails. To prevent unnecessary waste of resources, we recommend that you set an odd replica number.

We suggest that you set replica\_factor=3 for a production environment and replica\_factor=1 for a test environment. Do not use an even number.

# 20.4.6 "Is stopping or killing slow queries supported?"

Yes. For more information, see Kill query.

# 20.4.7 "Why are the query results different when using 60 and MATCH to execute the same semantic query?"

The possible reasons are listed as follows.

- GO statements find the dangling edges.
- RETURN commands do not specify the sequence.
- The dense vertex truncation limitation defined by max\_edge\_returned\_per\_vertex in the Storage service is triggered.
- Using different types of paths may cause different query results.
- 60 statements use walk. Both vertices and edges can be repeatedly visited in graph traversal.
- MATCH statements are compatible with openCypher and use trait. Only vertices can be repeatedly visited in graph traversal.

The example is as follows.



All queries that start from A with 5 hops will end at C (A - B - C - D - E - C). If it is 6 hops, the GO statement will end at D (A - B - C - D - E - C), because the edge C->D can be visited repeatedly. However, the MATCH statement returns empty, because edges cannot be visited repeatedly.

Therefore, using 60 and MATCH to execute the same semantic query may cause different query results.

For more information, see Wikipedia.

# 20.4.8 "How to count the vertices/edges number of each tag/edge type?"

See show-stats.

### 20.4.9 "How to get all the vertices/edge of each tag/edge type?"

## 1. Create and rebuild the index.

> CREATE TAG INDEX IF NOT EXISTS i\_player ON player(); > REBUILD TAG INDEX IF NOT EXISTS i\_player;

2. Use LOOKUP or MATCH. For example:

> LOOKUP ON player; > MATCH (n:player) RETURN n;

For more information, see INDEX, LOOKUP, and MATCH.

20.4.10 "Can non-English characters be used as identifiers, such as the names of graph spaces, tags, edge types, properties, and indexes?"

Yes, for more information, see Keywords and reserved words.

## 20.4.11 "How to get the out-degree/the in-degree of a given vertex?"

The out-degree of a vertex refers to the number of edges starting from that vertex, while the in-degree refers to the number of edges pointing to that vertex.

This is a very slow operation to get the out/in degree since no accelaration can be applied (no indices or caches). It also could be out-of-memory when hitting a supper-node.

#### 20.4.12 "How to quickly get the out-degree and in-degree of all vertices?"

There is no such command.

You can use NebulaGraph Algorithm.

# 20.5 About operation and maintenance

#### 20.5.1 "The runtime log files are too large. How to recycle the logs?"

NebulaGraph uses glog for log printing, which does not support log recycling. You can manage runtime logs by using cron jobs or the log management tool logrotate. For operational details, see Log recycling.

### 20.5.2 "How to check the NebulaGraph version?"

If the service is running: run command SHOW HOSTS META in nebula-console. See SHOW HOSTS.

If the service is not running:

Different installation methods make the method of checking the version different. The instructions are as follows:

If the service is not running, run the command ./<binary\_name> --version to get the version and the Git commit IDs of the NebulaGraph binary files. For example:

\$ ./nebula-graphd --version

• If you deploy NebulaGraph with Docker Compose

Check the version of NebulaGraph deployed by Docker Compose. The method is similar to the previous method, except that you have to enter the container first. The commands are as follows:

docker exec -it	nebula-docker-compose_graphd_1 bash
cd bin/	
./nebula-graphd	version

#### • If you install NebulaGraph with RPM/DEB package

Run rpm -qa |grep nebula to check the version of NebulaGraph.

#### 20.5.3 "How to scale my cluster up/down or out/in?"

# **M**arning

The cluster scaling function has not been officially released in the community edition. The operations involving SUBMIT JOB BALANCE DATA REMOVE and SUBMIT JOB BALANCE DATA are experimental features in the community edition and the functionality is not stable. Before using it in the community edition, make sure to back up your data first and set enable\_experimental\_feature and enable\_data\_balance to true in the Graph configuration file.

#### Increase or decrease the number of Meta, Graph, or Storage nodes

- NebulaGraph 3.6.0 does not provide any commands or tools to support automatic scale out/in. You can refer to the following steps:
- Scale out and scale in metad: The metad process can not be scaled out or scale in. The process cannot be moved to a new machine. You cannot add a new metad process to the service.

#### Q Note

You can use the Meta transfer script tool to migrate Meta services. Note that the Meta-related settings in the configuration files of Storage and Graph services need to be modified correspondingly.

- Scale in graphd: Remove the IP of the graphd process from the code in the client. Close this graphd process.
- Scale out graphd: Prepare the binary and config files of the graphd process in the new host. Modify the config files and add all existing addresses of the metad processes. Then start the new graphd process.
- Scale in storaged: See Balance remove command. After the command is finished, stop this storaged process.

# Caution

- Before executing this command to migrate the data in the specified Storage node, make sure that the number of other Storage nodes is sufficient to meet the set replication factor. For example, if the replication factor is set to 3, then before executing this command, make sure that the number of other Storage nodes is greater than or equal to 3.
- If there are multiple space partitions in the Storage node to be migrated, execute this command in each space to migrate all space partitions in the Storage node.
- Scale out storaged: Prepare the binary and config files of the storaged process in the new host, modify the config files and add all existing addresses of the metad processes. Then register the storaged process to the metad, and then start the new storaged process. For details, see Register storaged services.

You also need to run Balance Data and Balance leader after scaling in/out storaged.

#### Add or remove disks in the Storage nodes

Currently, Storage cannot dynamically recognize new added disks. You can add or remove disks in the Storage nodes of the distributed cluster by following these steps:

1. Execute SUBMIT JOB BALANCE DATA REMOVE <ip:port> to migrate data in the Storage node with the disk to be added or removed to other Storage nodes.

# Caution

- Before executing this command to migrate the data in the specified Storage node, make sure that the number of other Storage nodes is sufficient to meet the set replication factor. For example, if the replication factor is set to 3, then before executing this command, make sure that the number of other Storage nodes is greater than or equal to 3.
- If there are multiple space partitions in the Storage node to be migrated, execute this command in each space to migrate all space partitions in the Storage node.
- 2. Execute DROP HOSTS <ip:port> to remove the Storage node with the disk to be added or removed.
- 3. In the configuration file of all Storage nodes, configure the path of the new disk to be added or removed through --data\_path, see Storage configuration file for details.
- 4. Execute ADD HOSTS <ip:port> to re-add the Storage node with the disk to be added or removed.
- 5. As needed, execute SUBMIT JOB BALANCE DATA to evenly distribute the shards of the current space to all Storage nodes and execute SUBMIT JOB BALANCE LEADER command to balance the leaders in all spaces. Before running the command, select a space.

#### 20.5.4 "After changing the name of the host, the old one keeps displaying OFFLINE . What should I do?"

Hosts with the status of OFFLINE will be automatically deleted after one day.

#### 20.5.5 "How do I view the dmp file?"

The dmp file is an error report file detailing the exit of the process and can be viewed with the gdb utility. the Coredump file is saved in the directory of the startup binary (by default it is /usr/local/nebula) and is generated automatically when the NebulaGraph service crashes.

1. Check the Core file process name, pid is usually a numeric value.

\$ file core.<pid>

2. Use gdb to debug.

\$ gdb <process.name> core.<pid>

### 3. View the contents of the file.

\$(gdb) bt

#### For example:

\$ file core.1316027 core.1316027: ELF 64-bit LSB core file, x86-64, version 1 (SYSV), SVR4-style, from '/home/workspace/fork/nebula-debug/bin/nebula-metad --flagfile /home/k', real uid: 1008, effective uid: 1008, real gid: 1008, effective gid: 1008, execfn: '/home/workspace/fork/nebula-debug/bin/nebula-metad', platform: 'x86\_64' \$ gdb /home/workspace/fork/nebula-debug/bin/nebula-metad core.1316027 \$(gdb) bt #0 0x000079de58fecf5 in \_\_memcpy\_ssse3\_back () from /lib64/libc.so.6 #1 0x00000000b02299 in void std::\_cxx11::basic\_stringschar, std::char\_traits<char>, std::allocator<char>>::\_M\_construct<char\*, char\*, std::forward\_iterator\_tag) () #2 0x00000000b0271a7 in nebula::meta::cpp2::QueryDesc:(nebula::meta::cpp2::QueryDesc const&) () If you are not clear about the information that dmp prints out, you can post the printout with the OS version, hardware configuration, error logs before and after the Core file was created and actions that may have caused the error on the NebulaGraph forum.

# 20.5.6 How can I set the NebulaGraph service to start automatically on boot via systemctl?

1. Execute systemctl enable to start the metad, graphd and storaged services.

[root]# systemctl enable nebula-metad.service Created symlink from /etc/system/multi-user.target.wants/nebula-metad.service to /usr/lib/system/nebula-metad.service. [root]# systemctl enable nebula-graphd.service Created symlink from /etc/system/multi-user.target.wants/nebula-graphd.service to /usr/lib/system/nebula-graphd.service. [root]# systemctl enable nebula-storaged.service Created symlink from /etc/system/multi-user.target.wants/nebula-storaged.service to /usr/lib/systemd/system/nebula-graphd.service.

2. Configure the service files for metad, graphd and storaged to set the service to pull up automatically.

# Caution

The following points need to be noted when configuring the service file. - The paths of the PIDFile, ExecStart, ExecReload and ExecStop parameters need to be the same as those on the server. - RestartSec is the length of time (in seconds) to wait before restarting, which can be modified according to the actual situation. - (Optional) StartLimitInterval is the unlimited restart, the default is 10 seconds if the restart exceeds 5 times, and set to 0 means unlimited restart. - (Optional) LimitNOFILE is the maximum number of open files for the service, the default is 1024 and can be changed according to the actual situation.

Configure the service file for the metad service.



#### Configure the service file for the graphd service.

<pre>\$ vi /usr/lib/systemd/system/nebula-graphd.service [Unit]</pre>
Description=Nebula Graph Graphd Service
After=network.target
[Service]
Type=forking
Restart=always
RestartSec=15s
PIDFile=/usr/local/nebula/pids/nebula-graphd.pid
ExecStart=/usr/local/nebula/scripts/nebula.service start graph
ExecReload=/usr/local/nebula/scripts/nebula.service restart gr
<pre>ExecStop=/usr/local/nebula/scripts/nebula.service stop graphd</pre>
PrivateTmp=true
StartLimitInterval=0
LimitNOFILE=1024
[Instal]]
WantedBy-multi-user target
wanteuby-mutti-user.target

aphd

Configure the service file for the storaged service.

```
$ vi /usr/lib/systemd/system/nebula-storaged.service
[Unit]
Description=Nebula Graph Storaged Service
After=network.target
```

[Service] Type=Forking Restart=always RestartSec=15s PIDFile=/usr/local/nebula/scripts/nebula.service start storaged ExecReload=/usr/local/nebula/scripts/nebula.service restart storaged ExecStop=/usr/local/nebula/scripts/nebula.service stop storaged PrivateTmp=true StartLimitInterval=0 LimitNOFILE=1024

[Install] WantedBy=multi-user.target

#### 3. Reload the configuration file.

[root]# sudo systemctl daemon-reload

### 4. Restart the service.

\$ systemctl restart nebula-metad.service \$ systemctl restart nebula-graphd.service \$ systemctl restart nebula-storaged.service

# 20.6 About connections

## 20.6.1 "Which ports should be opened on the firewalls?"

If you have not modified the predefined ports in the Configurations, open the following ports for the NebulaGraph services:

Service	Port
Meta	9559, 9560, 19559
Graph	9669, 19669
Storage	9777 ~ 9780, 19779

If you have customized the configuration files and changed the predefined ports, find the port numbers in your configuration files and open them on the firewalls.

For more port information, see Port Guide for Company Products.

## 20.6.2 "How to test whether a port is open or closed?"

You can use telnet as follows to check for port status.

telnet <ip> <port>

#### O Note

If you cannot use the telnet command, check if telnet is installed or enabled on your host.

For example:

// If the port is open: \$ telnet 192.168.1.10 9669 Trying 192.168.1.10... Connected to 192.168.1.10. Escape character is '^]'.

// If the port is closed or blocked: \$ telnet 192.168.1.10 9777 Trying 192.168.1.10... telnet: connect to address 192.168.1.10: Connection refused Last update: January 31, 2024

# 21. Appendix

# 21.1 Release Note

# 21.1.1 NebulaGraph 3.6.0 release notes

# Features

- Enhance the full-text index. #5567 #5575 #5577 #5580 #5584 #5587
  - The changes involved are listed below:
- The original full-text indexing function has been changed from calling Elasticsearch's Term-level queries to Full text queries.
- In addition to supporting wildcards, regulars, fuzzy matches, etc. (but the syntax has been changed), support for word splitting (relying on Elasticsearch's own word splitter) has been added, and the query results include scoring results. For more syntax, see official Elasticsearch documentation.

# Enhancements

- Support variables when querying vertex id or property index in a match clause.  $#5486 \ #5553$
- Performance
- Support parallel startup of RocksDB instances to speed up the startup of the Storage service. #5521
- Optimize the prefix search performance of the RocksDB iterator after the DeleteRange operation. #5525
- Optimize the appendLog sending logic to avoid impacting write performance when a follower is down. #5571
- Optimize the performance of the MATCH statement when querying for non-existent properties. #5634

# Bug fixes

- DQL
- Fix the crash of the Graph service when executing a single big query. #5619
- Fix the crash of the Graph service when executing the Find All Path statement. #5621 #5640
- Fix the bug that some expired data is not recycled at the bottom level.  $#5447 \ #5622$
- Fix the bug that adding a path variable in the MATCH statement causes the all() function push-down optimization to fail. #5631
- Fix the bug in the MATCH statement that returns incorrect results when querying the self-loop by the shortest path. #5636
- Fix the bug that deleting edges by pipe causes the Graph service to crash. #5645
- Fix the bug in the MATCH statement that returns missing properties of edges when matching multiple hops. #5646
- Others
- Fix the bug of meta data inconsistency. #5517
- Fix the bug that RocksDB ingest causes the leader lease to be invalid. #5534
- Fix the error in the statistics logic of storage. #5547
- Fix the bug that causes the web service to crash if a flag is set for an invalid request parameter. #5566
- Fix the bug that too many logs are printed when listing sessions. #5618

Last update: November 16, 2023

# 21.1.2 NebulaGraph Studio release notes

#### v3.8.0

- Features
- Supported the use of MySQL databases as backend storage.
- Enhancements
- Supported customizing the read and write parameters of the WebSocket.
- Usability
- Supported filtering tasks in the import task list based on the map space name.
- Compatibility
- Since the database table structure has changed, you need to set DB.AutoWigrate to true in the configuration file, and the system will automatically upgrade and adapt the existing historical data.

If the tables were created manually after you consulted our after-sales staff, please modify these tables manually: task\_infos, task\_effects, sketches, schema\_snapshots, favorites, files, and datasources.

For example:

```
ALTER TABLE 'task_infos' ADD COLUMN 'b_id' CHAR(32) NOT NULL DEFAULT '';

UPDATE TABLE 'task_infos' SET 'b_id' = 'id';

CREATE UNIQUE INDEX 'idx_task_infos_id' ON 'task_infos'('b_id');

ALTER TABLE 'task_effects' ADD COLUMN 'b_id' CHAR(32) NOT NULL DEFAULT '';

UPDATE TABLE 'task_effects' SET 'b_id' = 'id';

CREATE UNIQUE INDEX 'idx_task_effects_id' ON 'task_effects'('b_id');

...
```

### v3.7.0

- Enhancements
- Supported importing SFTP, Amazon S3 data files.
- The import page is supported to configure more import parameters, such as concurrency, retries, etc.
- Supported re-running tasks.
- Supported saving tasks as drafts.
- Supported running Studio in a docker container on the ARM architecture.

Last update: December 1, 2023

# 21.1.3 NebulaGraph Dashboard Community Edition release notes

# **Community Edition 3.4.0**

- Feature
- Support the built-in dashboard.service script to manage the Dashboard services with one-click and view the Dashboard version.
- Support viewing the configuration of Meta services.
- Enhancement
- Adjust the directory structure and simplify the deployment steps.
- Display the names of the monitoring metrics on the overview page of machine.
- Optimize the calculation of monitoring metrics such as <code>num\_queries</code>, and adjust the display to time series aggregation.

Last update: April 25, 2024

# 21.2 Ecosystem tools overview



# 21.2.1 NebulaGraph Studio

NebulaGraph Studio (Studio for short) is a graph database visualization tool that can be accessed through the Web. It can be used with NebulaGraph DBMS to provide one-stop services such as composition, data import, writing nGQL queries, and graph exploration. For details, see What is NebulaGraph Studio.

O Note	
The release of the Studio is in	ndependent of NebulaGraph core, and its naming method is also not the same as the core naming rules.
NebulaGraph version	Studio version
v3.6.0	v3.8.0

# 21.2.2 NebulaGraph Dashboard Community Edition

NebulaGraph Dashboard Community Edition (Dashboard for short) is a visualization tool for monitoring the status of machines and services in the NebulaGraph cluster. For details, see What is NebulaGraph Dashboard.

NebulaGraph version	Dashboard Community version
v3.6.0	v3.4.0

# 21.2.3 NebulaGraph Exchange

NebulaGraph Exchange (Exchange for short) is an Apache Spark&trade application for batch migration of data in a cluster to NebulaGraph in a distributed environment. It can support the migration of batch data and streaming data in a variety of different formats. For details, see What is NebulaGraph Exchange.

NebulaGraph version	Exchange Community version
v3.6.0	v3.6.1

# 21.2.4 NebulaGraph Operator

NebulaGraph Operator (Operator for short) is a tool to automate the deployment, operation, and maintenance of NebulaGraph clusters on Kubernetes. Building upon the excellent scalability mechanism of Kubernetes, NebulaGraph introduced its operation

and maintenance knowledge into the Kubernetes system, which makes NebulaGraph a real cloud-native graph database. For more information, see What is NebulaGraph Operator.

NebulaGraph version	Operator version
v3.6.0	v1.8.0

## 21.2.5 NebulaGraph Importer

NebulaGraph Importer (Importer for short) is a CSV file import tool for NebulaGraph. The Importer can read the local CSV file, and then import the data into the NebulaGraph database. For details, see What is NebulaGraph Importer.

NebulaGraph version	Importer version
v3.6.0	v4.1.0

## 21.2.6 NebulaGraph Spark Connector

NebulaGraph Spark Connector is a Spark connector that provides the ability to read and write NebulaGraph data in the Spark standard format. NebulaGraph Spark Connector consists of two parts, Reader and Writer. For details, see What is NebulaGraph Spark Connector.

NebulaGraph version	Spark Connector version
v3.6.0	v3.6.0

# 21.2.7 NebulaGraph Flink Connector

NebulaGraph Flink Connector is a connector that helps Flink users quickly access NebulaGraph. It supports reading data from the NebulaGraph database or writing data read from other external data sources to the NebulaGraph database. For details, see What is NebulaGraph Flink Connector.

NebulaGraph version	Flink Connector version
v3.6.0	v3.5.0

# 21.2.8 NebulaGraph Algorithm

NebulaGraph Algorithm (Algorithm for short) is a Spark application based on GraphX, which uses a complete algorithm tool to analyze data in the NebulaGraph database by submitting a Spark task To perform graph computing, use the algorithm under the lib repository through programming to perform graph computing for DataFrame. For details, see What is NebulaGraph Algorithm.

NebulaGraph version	Algorithm version
v3.6.0	v3.0.0

# 21.2.9 NebulaGraph Console

NebulaGraph Console is the native CLI client of NebulaGraph. For how to use it, see NebulaGraph Console.

NebulaGraph version	<b>Console version</b>
v3.6.0	v3.6.0

# 21.2.10 NebulaGraph Docker Compose

Docker Compose can quickly deploy NebulaGraph clusters. For how to use it, please refer to Docker Compose Deployment NebulaGraph.

NebulaGraph version	Docker Compose version	
v3.6.0	v3.6.0	

# 21.2.11 Backup & Restore

Backup&Restore (BR for short) is a command line interface (CLI) tool that can help back up the graph space data of NebulaGraph, or restore it through a backup file data.

NebulaGraph version	<b>BR version</b>		
v3.6.0	v3.6.0		

# 21.2.12 NebulaGraph Bench

NebulaGraph Bench is used to test the baseline performance data of NebulaGraph. It uses the standard data set of LDBC.

NebulaGraph version	<b>Bench version</b>		
v3.6.0	v1.2.0		

# 21.2.13 API, SDK

 Empatibility

 Select the latest version of X.Y.\* which is the same as the core version.

NebulaGra	ph version	Language	
v3.6.0		C++	
v3.6.0		Go	
v3.6.0		Python	
v3.6.0		Java	
v3.6.0		HTTP	
v3.6.0		HIIP	

# 21.2.14 Community contributed tools

The following are useful utilities and tools contributed and maintained by community users.

- Object Relational Mapping (ORM) frameworks
- NGBATIS: An ORM framework that integrates with the Spring Boot ecosystem
- graph-ocean: An ORM framework developed based on NebulaGraph Java client
- nebula-jdbc: An ORM framework that supports JDBC
- nebula-carina: An ORM framework developed based on NebulaGraph Python client
- norm: An ORM framework written in Golang

## • Data processing tools

- nebula-real-time-exchange: Enables real-time data synchronization from MySQL to NebulaGraph
- nebula-datax-plugin: Provides NebulaGraph's Reader and Writer plugins based on DataX to enable offline data synchronization

# • Quick deployment

- nebulagraph-docker-ext: Starts NebulaGraph in Docker Desktop in 10 seconds
- nebulagraph-lite: A NebulaGraph sandbox running in the browser

### • Testing

• testcontainers-nebula: A lightweight database testing library for Java

## • Clients

- zio-nebula: Scala client
- nebula-node: Node.js client
- nebula-php: PHP client
- nebula-net: .NET client
- nebula-rust: Rust client
- Terminal tools
- nebula-console-intellij-plugin: A Nebula-console plugin for JetBrains IDEs that supports syntax highlighting, function field autocompletion, data table pagination, and relationship graphs.

Last update: April 15, 2024

# 21.3 Port guide for company products

The following are the default ports used by NebulaGraph core and peripheral tools.

No.	Product / Service	Туре	Default	Description
1	NebulaGraph	ТСР	9669	Graph service RPC daemon listening port. Commonly used for client connections to the Graph service.
2	NebulaGraph	TCP	19669	Graph service HTTP port.
3	NebulaGraph	TCP	19670	Graph service HTTP/2 port. (Deprecated after version 3.x)
4	NebulaGraph	ТСР	9559, 9560	9559 is the RPC daemon listening port for Meta service. Commonly used by Graph and Storage services for querying and updating metadata in the graph database. The neighboring +1 (9560) port is used for Raft communication between Meta services.
5	NebulaGraph	TCP	19559	Meta service HTTP port.
6	NebulaGraph	TCP	19560	Meta service HTTP/2 port. (Deprecated after version $3.x$ )
7	NebulaGraph	TCP	9779, 9778, 9780	<ul> <li>9779 is the RPC daemon listening port for Storage service.</li> <li>Commonly used by Graph services for data storage-related operations, such as reading, writing, or deleting data.</li> <li>The neighboring ports -1 (9778) and +1 (9780) are also used.</li> <li>9778 : The port used by the Admin service, which receives Meta commands for Storage.</li> <li>9780 : The port used for Raft communication between Storage services.</li> </ul>
8	NebulaGraph	TCP	19779	Storage service HTTP port.
9	NebulaGraph	TCP	19780	Storage service HTTP/2 port. (Deprecated after version $3.x$ )
10	NebulaGraph	ТСР	8888	Backup and restore Agent service port. The Agent is a daemon running on each machine in the cluster, responsible for starting and stopping NebulaGraph services and uploading and downloading backup files.
11	NebulaGraph	TCP	9789, 9788, 9790	<ul> <li>9789 is the Raft Listener port for Full-text index, which reads data from Storage services and writes it to the Elasticsearch cluster.</li> <li>Also the port for Storage Listener in inter-cluster data synchronization, used for synchronizing Storage data from the primary cluster.</li> <li>The neighboring ports -1 (9788) and +1 (9790) are also used.</li> <li>9788 : An internal port.</li> <li>9790 : The port used for Raft communication.</li> </ul>
12	NebulaGraph	ТСР	9200	NebulaGraph uses this port for HTTP communication with Elasticsearch to perform full-text search queries and manage full-text indexes.
13	NebulaGraph	TCP	9569, 9568, 9570	<ul> <li>9569 is the Meta Listener port in inter-cluster data synchronization, used for synchronizing Meta data from the primary cluster.</li> <li>The neighboring ports -1 (9568) and +1 (9570) are also used.</li> <li>9568 : An internal port.</li> <li>9570 : The port used for Raft communication.</li> </ul>
14	NebulaGraph	TCP		

No.	Product / Service	Туре	Default	Description
			9889, 9888, 9890	Drainer service port in inter-cluster data synchronization, used for synchronizing Storage and Meta data to the primary cluster. The neighboring ports -1 (9888) and +1 (9890) are also used. 9888 : An internal port. 9890 : The port used for Raft communication.
15	NebulaGraph Studio	TCP	7001	Studio web service port.
16	NebulaGraph Dashboard	ТСР	8090	Nebula HTTP Gateway dependency service port. Provides an HTTP interface for cluster services to interact with the NebulaGraph database using nGQL statements.0
17	NebulaGraph Dashboard	ТСР	9200	Nebula Stats Exporter dependency service port. Collects cluster performance metrics, including service IP addresses, versions, and monitoring metrics (such as query count, query latency, heartbeat latency, etc.).
18	NebulaGraph Dashboard	TCP	9100	Node Exporter dependency service port. Collects resource information for machines in the cluster, including CPU, memory, load, disk, and traffic.
19	NebulaGraph Dashboard	TCP	9090	Prometheus service port. Time-series database for storing monitoring data.
20	NebulaGraph Dashboard	TCP	7003	Dashboard Community Edition web service port.

Last update: November 10, 2023

# 21.4 How to Contribute

## 21.4.1 Before you get started

#### Commit an issue on the github or forum

You are welcome to contribute any code or files to the project. But firstly we suggest you raise an issue on the github or the forum to start a discussion with the community. Check through the topic for Github.

#### Sign the Contributor License Agreement CLA

1. Open the CLA sign-in page.

- 2. Click the **Sign in with GitHub** button to sign in.
- 3. Read and agree to the vesoft inc. Contributor License Agreement.

If you have any questions, submit an issue.

## 21.4.2 Modify a single document

This manual is written in the Markdown language. Click the pencil icon on the right of the document title to commit the modification.

This method applies to modifying a single document only.

# 21.4.3 Batch modify or add files

This method applies to contributing code, modifying multiple documents in batches, or adding new documents.

# 21.4.4 Step 1: Fork in the github.com

The NebulaGraph project has many repositories. Take the nebul repository for example:

#### 1. Visit https://github.com/vesoft-inc/nebula.

2. Click the Fork button to establish an online fork.

# 21.4.5 Step 2: Clone Fork to Local Storage

#### 1. Define a local working directory.

# Define the working directory.
working\_dir=\$HOME/Workspace

#### 2. Set user to match the Github profile name.

user={the Github profile name}

#### 3. Create your clone.

mkdir -p \$working\_dir cd \$working\_dir git clone https://github.com/\$user/nebula.git # or: git clone git@github.com:\$user/nebula.git

#### cd Śworking dir/nebula

git remote add upstream https://github.com/vesoft-inc/nebula.git
# or: git remote add upstream git@github.com:vesoft-inc/nebula.git

# Never push to upstream master since you do not have write access. git remote set-url --push upstream no\_push

# Confirm that the remote branch is valid.

```
# origin git@github.com:$(user)/nebula.git (fetch)
# origin git@github.com:$(user)/nebula.git (push)
# upstream https://github.com/vesoft-inc/nebula (fetch)
# upstream no_push (push)
git remote -v
```

4. (Optional) Define a pre-commit hook.

Please link the NebulaGraph pre-commit hook into the .git directory.

This hook checks the commits for formatting, building, doc generation, etc.

cd \$working\_dir/nebula/.git/hooks
ln -s \$working\_dir/nebula/.linters/cpp/hooks/pre-commit.sh .

Sometimes, the pre-commit hook cannot be executed. You have to execute it manually.

cd \$working\_dir/nebula/.git/hooks
chmod +x pre-commit

# 21.4.6 Step 3: Branch

1. Get your local master up to date.

cd \$working\_dir/nebula git fetch upstream git checkout master git rebase upstream/master

2. Checkout a new branch from master.

git checkout -b myfeature

# Note

Because the PR often consists of several commits, which might be squashed while being merged into upstream. We strongly suggest you to open a separate topic branch to make your changes on. After merged, this topic branch can be just abandoned, thus you could synchronize your master branch with upstream easily with a rebase like above. Otherwise, if you commit your changes directly into master, you need to use a hard reset on the master branch. For example:

git fetch upstream git checkout master git reset --hard upstream/master git push --force origin master

# 21.4.7 Step 4: Develop

• Code style

**NebulaGraph** adopts cpplint to make sure that the project conforms to Google's coding style guides. The checker will be implemented before the code is committed.

• Unit tests requirements

Please add unit tests for the new features or bug fixes.

• Build your code with unit tests enabled

For more information, see Install NebulaGraph by compiling the source code.

# Note

Make sure you have enabled the building of unit tests by setting  $\mbox{-DENABLE\_TESTING=ON}$  .

#### • Run tests

In the root directory of nebula, run the following command:

cd nebula/build
ctest -j\$(nproc)

#### 21.4.8 Step 5: Bring Your Branch Update to Date

# While on your myfeature branch.
git fetch upstream
git rebase upstream/master

Users need to bring the head branch up to date after other contributors merge PR to the base branch.

#### 21.4.9 Step 6: Commit

Commit your changes.

git commit -a

Users can use the command --amend to re-edit the previous code.

### 21.4.10 Step 7: Push

When ready to review or just to establish an offsite backup, push your branch to your fork on github.com :

git push origin myfeature

# 21.4.11 Step 8: Create a Pull Request

1. Visit your fork at https://github.com/\$user/nebula (replace \$user here).

2. Click the Compare & pull request button next to your myfeature branch.

## 21.4.12 Step 9: Get a Code Review

Once your pull request has been created, it will be assigned to at least two reviewers. Those reviewers will do a thorough code review to make sure that the changes meet the repository's contributing guidelines and other quality standards.

## 21.4.13 Add test cases

For detailed methods, see How to add test cases.

# 21.4.14 Donation

# Step 1: Confirm the project donation

Contact the official NebulaGraph staff via email, WeChat, Slack, etc. to confirm the donation project. The project will be donated to the NebulaGraph Contrib organization.

Email address: info@vesoft.com

WeChat: NebulaGraphbot

Slack: Join Slack

# Step 2: Get the information of the project recipient

The NebulaGraph official staff will give the recipient ID of the NebulaGraph Contrib project.

## Step 3: Donate a project

The user transfers the project to the recipient of this donation, and the recipient transfers the project to the NebulaGraph Contrib organization. After the donation, the user will continue to lead the development of community projects as a Maintainer.

For operations of transferring a repository on GitHub, see Transferring a repository owned by your user account.

Last update: January 2, 2024

# 21.5 History timeline for NebulaGraph

1. 2018.9: dutor wrote and submitted the first line of NebulaGraph database code.



 $2.\ 2019.5:\ NebulaGraph\ v0.1.0-alpha\ was\ released\ as\ open-source.$


 $NebulaGraph \ v1.0.0-beta, \ v1.0.0-rc1, \ v1.0.0-rc2, \ v1.0.0-rc3, \ and \ v1.0.0-rc4 \ were \ released \ one \ after \ another \ within \ a \ year \ thereafter.$ 

Pre-release           V0.1.0           -O- b0d817f	Nebula Graph v0.1.0
Compare 🕶	This is the first release of <i>Nebula Graph</i> , a brand new, fast and distributed graph database.
	Available Features
	<ul> <li>Physical data isolation with Graph Space</li> <li>Strongly typed schema support</li> <li>Vertices and edges insertion</li> <li>Graph traversal(the go statement)</li> </ul>
	<ul> <li>Variable definition and reference</li> <li>Piping query result between statements</li> <li>Client API in C++, Golang and Java</li> </ul>
	Features Coming Soon
	<ul> <li>Raft support</li> <li>Query based on secondary index(the LOOKUP statement)</li> <li>Sub-graph retrieval(the матсн statement)</li> <li>User defined function call</li> <li>User management</li> </ul>
	Try Out
	A Docker image is available for trial purpose. You can get it by following the guide here.
	✓ Assets 2
	Source code (zip)

3. 2019.7: NebulaGraph's debut at HBaseCon<sup>1</sup>. @dangleptr

Source code (tar.gz)



- 4. 2020.3: NebulaGraph v2.0 was starting developed in the final stage of v1.0 development.
- 5. 2020.6: The first major version of NebulaGraph v1.0.0 GA was released.

## V1.0.0 GA

🖸 v1.0.0

-O- 06a5db4

Verified Compare • jude-zhu released this on Jun 10, 2020 · 146 commits to master since this release

## **Basic Features**

- Online DDL & DML. Support updating schemas and data without stopping or affecting your ongoing operations.
- Graph traversal. 60 statement supports forward/reverse and bidirectional graph traversal. 60 minHops TO maxHops is supported to get variable hops relationships.
- Aggregate. Support aggregation functions such as GROUP BY, ORDER BY, and LIMIT.
- Composite query. Support composite clauses: UNION , UNION DISTINCT , INTERSECT , and MINUS .
- PIPE statements. The result yielded from the previous statement could be piped to the next statement as input.
- Use defined variables. Support user-defined variables to pass the result of a query to another.
- Index. Both the single-property index and composite index are supported to make searches of related data more efficient. LOOKUP ON statement is to query on the index.

## **Advanced Features**

- Privilege Management. Support user authentication and role-based access control. Nebula Graph can easily integrate with third-party
  authentication systems. There are five built-in roles in Nebula Graph: GOD, ADMIN, DBA, USER, and GUEST. Each role has its
  corresponding privileges.
- Support Reservoir Sampling, which will retrieve k elements randomly for the sampling of the supernode at the complexity of O(n).
- Cluster snapshot. Support creating snapshots for the cluster as an online backup strategy.
- TTL. Support TTL to expire items after a certain amount of time automatically.
- Operation & Maintenance
  - Scale in/out. Support online scale in/out and load balance for storage
  - HOSTS clause to manage storage hosts
  - CONFIGS clause to manage configuration options
- Job Manager & Scheduler. A tool for job managing and scheduling. Currently, COMPACT and FLUSH jobs are supported.
- · Graph Algorithms. Support finding the full path and the shortest path between vertices.
- Provide OLAP interfaces to integrate with third-party graph analytics platforms.
- Support multiple character sets and collations. The default CHARSET and COLLATE are utf8 and utf8\_bin .

## Clients

- Java Client. Support source code building and downloading from the MVN repository, see Java Client for more details.
- Python Client. Support source code building and installation with pip, see Python Client for more details.
- Golang Client. Install the client with the command go get -u -v github.com/vesoft-inc/nebula-go , see Go Client for more details.

## Nebula Graph Studio

A graphical user interface for working with Nebula Graph. Support querying, designing schema, data loading, and graph exploring. See Nebula Graph Studio for more details.

6. 2021.3: The second major version of NebulaGraph v2.0 GA was released.

# © v2.00 ⊙ v16394b (Verified) Verified ↓ ude-zhu released this on Mar 23

- Compare New Features
  - vertexID supports both Integer and String.
  - New data types: NULL: the property can be set to NULL. NOT NULL constraint is also supported
  - Composite types: LIST, SET, and MAP(Cannot be set as property types)
  - Temporal types: DATE and DATETIME.
  - · FIXED\_STRING: a fixed size String
  - Full-text indexes are supported to do prefix, wildcard, regex, and fuzzy search on a string property
  - · Explain & Profile outputs the execution plan of an nGQL statement and execution profile Subgraph to retrieve vertices and edges reachable from the start vertices.
  - · Support to collect statistics of the graph space.
  - OpenCypher compatibility
     Partially support the MATCH clause

    - Support RETURN, WITH, UNWIND, LIMIT & SKIP clauses
  - More built-in functions Predicate functions
    - Scalar functions
    - List functions
    - Aggregating functions
    - Mathematical functions String functions
    - Temporal function

#### Improvements

- · Optimize the performance of inserting, updating, and deleting data with indexes.
- · LOOKUP ON filtering data supports OR and AND operators.
- FIND PATH supports finding paths with or without regard to direction, and also supports excluding cycles in paths. SHOW HOSTS graph/meta/storage supports to retrieve the basic information of graphd/metad/storaged hosts

#### Changelog

- The data type of vertexID must be specified when creating a graph space.
- FETCH PROP ON returns a composite object if not specify the result set
- · Changed the default port numbers of metad, graphd, and storaged. Refactor metrics counters

## Nebula-graph Console

Supports local commands mode. :set csv outputs the query results to the console and the specified CSV file. For more information, please refer to https://github.c

#### Clients

Support connection pool and load balance.

- cpp client https://github.com/vesoft-inc/nebula-c
- java client https://github.com/vesoft-inc/nebula-java
- python client https://github.com/vesoft-inc/nebula-py go client https://github.com/vesoft-inc/nebula-go

## Nebula Graph Studio

With Studio, you can create a graph schema, load data, execute nGQL statements, and explore graphs in one stop. For more information please refer to https://github.co m/vesoft-inc/nebula-web-docke

Known Issues

• #860

- 7. 2021.8: NebulaGraph v2.5.0 was released.
- 8. 2021.10: NebulaGraph v2.6.0 was released.
- 9. 2022.2: NebulaGraph v3.0.0 was released.
- 10. 2022.4: NebulaGraph v3.1.0 was released.
- 11. 2022.7: NebulaGraph v3.2.0 was released.
- 12. 2022.10: NebulaGraph v3.3.0 was released.
- 13. 2023.2: NebulaGraph v3.4.0 was released.
- 14. 2023.5: NebulaGraph v3.5.0 was released.
- 15. 2023.8: NebulaGraph v3.6.0 was released.

1. NebulaGraph v1.x supports both RocksDB and HBase as its storage engines. NebulaGraph v2.x removes HBase supports. 🛩

Last update: October 25, 2023

## 21.6 Error code

 $Nebula Graph \ returns \ an \ error \ code \ when \ an \ error \ occurs. \ This \ topic \ describes \ the \ details \ of \ the \ error \ code \ returned.$ 

### Q Note

• If an error occurs but no error code is returned, or if the error code description is unclear, we welcome your feedback or suggestions on the forum or GitHub.

• When the code returned is [0], it means that the operation is successful.

Error name	Error Code	Description
E_DISCONNECTED	-1	Lost connection
E_FAIL_TO_CONNECT	-2	Unable to establish connection
E_RPC_FAILURE	-3	RPC failure
E_LEADER_CHANGED	-4	Raft leader has been changed
E_SPACE_NOT_FOUND	-5	Graph space does not exist
E_TAG_NOT_FOUND	-6	Tag does not exist
E_EDGE_NOT_FOUND	-7	Edge type does not exist
E_INDEX_NOT_FOUND	-8	Index does not exist
E_EDGE_PROP_NOT_FOUND	-9	Edge type property does not exist
E_TAG_PROP_NOT_FOUND	-10	Tag property does not exist
E_ROLE_NOT_FOUND	-11	The current role does not exist
E_CONFIG_NOT_FOUND	-12	The current configuration does not exist
E_MACHINE_NOT_FOUND	-13	The current host does not exist
E_LISTENER_NOT_FOUND	-15	Listener does not exist
E_PART_NOT_FOUND	-16	The current partition does not exist
E_KEY_NOT_FOUND	-17	Key does not exist
E_USER_NOT_FOUND	-18	User does not exist
E_STATS_NOT_FOUND	-19	Statistics do not exist
E_SERVICE_NOT_FOUND	-20	No current service found
E_DRAINER_NOT_FOUND	-21	Drainer does not exist
E_DRAINER_CLIENT_NOT_FOUND	-22	Drainer client does not exist
E_PART_STOPPED	-23	The current partition has already been stopped
E_BACKUP_FAILED	-24	Backup failed
E_BACKUP_EMPTY_TABLE	-25	The backed-up table is empty
E_BACKUP_TABLE_FAILED	-26	Table backup failure
E_PARTIAL_RESULT	-27	MultiGet could not get all data
E_REBUILD_INDEX_FAILED	-28	Index rebuild failed
E_INVALID_PASSWORD	-29	Password is invalid
E_FAILED_GET_ABS_PATH	-30	Unable to get absolute path
E_BAD_USERNAME_PASSWORD	-1001	Authentication failed
E_SESSION_INVALID	-1002	Invalid session
E_SESSION_TIMEOUT	-1003	Session timeout
E_SYNTAX_ERROR	-1004	Syntax error
E_EXECUTION_ERROR	-1005	Execution error
E_STATEMENT_EMPTY	-1006	Statement is empty

Error name	Error Code	Description
E_BAD_PERMISSION	-1008	Permission denied
E_SEMANTIC_ERROR	-1009	Semantic error
E_TOO_MANY_CONNECTIONS	-1010	Maximum number of connections exceeded
E_PARTIAL_SUCCEEDED	-1011	Access to storage failed (only some requests succeeded)
E_NO_HOSTS	-2001	Host does not exist
E_EXISTED	-2002	Host already exists
E_INVALID_HOST	-2003	Invalid host
E_UNSUPPORTED	-2004	The current command, statement, or function is not supported
E_NOT_DROP	-2005	Not allowed to drop
E_CONFIG_IMMUTABLE	-2007	Configuration items cannot be changed
E_CONFLICT	-2008	Parameters conflict with meta data
E_INVALID_PARM	-2009	Invalid parameter
E_WRONGCLUSTER	-2010	Wrong cluster
E_ZONE_NOT_ENOUGH	-2011	Listener conflicts
E_ZONE_IS_EMPTY	-2012	Host not exist
E_SCHEMA_NAME_EXISTS	-2013	Schema name already exists
E_RELATED_INDEX_EXISTS	-2014	There are still indexes related to tag or edge, cannot drop it
E_RELATED_SPACE_EXISTS	-2015	There are still some space on the host, cannot drop it
E_STORE_FAILURE	-2021	Failed to store data
E_STORE_SEGMENT_ILLEGAL	-2022	Illegal storage segment
E_BAD_BALANCE_PLAN	-2023	Invalid data balancing plan
E_BALANCED	-2024	The cluster is already in the data balancing status
E_NO_RUNNING_BALANCE_PLAN	-2025	There is no running data balancing plan
E_NO_VALID_HOST	-2026	Lack of valid hosts
E_CORRUPTED_BALANCE_PLAN	-2027	A data balancing plan that has been corrupted
E_IMPROPER_ROLE	-2030	Failed to recover user role
E_INVALID_PARTITION_NUM	-2031	Number of invalid partitions
E_INVALID_REPLICA_FACTOR	-2032	Invalid replica factor
E_INVALID_CHARSET	-2033	Invalid character set
E_INVALID_COLLATE	-2034	Invalid character sorting rules
E_CHARSET_COLLATE_NOT_MATCH	-2035	Character set and character sorting rule mismatch
E_SNAPSHOT_FAILURE	-2040	Failed to generate a snapshot
E_BLOCK_WRITE_FAILURE	-2041	Failed to write block data
E_ADD_JOB_FAILURE	-2044	Failed to add new task
E_STOP_JOB_FAILURE	-2045	Failed to stop task

Error name	Error Code	Description
E_SAVE_JOB_FAILURE	-2046	Failed to save task information
E_BALANCER_FAILURE	-2047	Data balancing failed
E_JOB_NOT_FINISHED	-2048	The current task has not been completed
E_TASK_REPORT_OUT_DATE	-2049	Task report failed
E_JOB_NOT_IN_SPACE	-2050	The current task is not in the graph space
E_JOB_NEED_RECOVER	-2051	The current task needs to be resumed
E_JOB_ALREADY_FINISH	-2052	The job status has already been failed or finished
E_JOB_SUBMITTED	-2053	Job default status
E_JOB_NOT_STOPPABLE	-2054	The given job do not support stop
E_JOB_HAS_NO_TARGET_STORAGE	-2055	The leader distribution has not been reported, so can't send task to storage
E_INVALID_JOB	-2065	Invalid task
E_BACKUP_BUILDING_INDEX	-2066	Backup terminated (index being created)
E_BACKUP_SPACE_NOT_FOUND	-2067	Graph space does not exist at the time of backup
E_RESTORE_FAILURE	-2068	Backup recovery failed
E_SESSION_NOT_FOUND	-2069	Session does not exist
E_LIST_CLUSTER_FAILURE	-2070	Failed to get cluster information
E_LIST_CLUSTER_GET_ABS_PATH_FAILURE	-2071	Failed to get absolute path when getting cluster information
E_LIST_CLUSTER_NO_AGENT_FAILURE	-2072	Unable to get an agent when getting cluster information
E_QUERY_NOT_FOUND	-2073	Query not found
E_AGENT_HB_FAILUE	-2074	Failed to receive heartbeat from agent
E_HOST_CAN_NOT_BE_ADDED	-2082	The host can not be added for it's not a storage host
E_ACCESS_ES_FAILURE	-2090	Failed to access elasticsearch
E_GRAPH_MEMORY_EXCEEDED	-2600	Graph memory exceeded
E_CONSENSUS_ERROR	-3001	Consensus cannot be reached during an election
E_KEY_HAS_EXISTS	-3002	Key already exists
E_DATA_TYPE_MISMATCH	-3003	Data type mismatch
E_INVALID_FIELD_VALUE	-3004	Invalid field value
E_INVALID_OPERATION	-3005	Invalid operation
E_NOT_NULLABLE	-3006	Current value is not allowed to be empty
E_FIELD_UNSET	-3007	Field value must be set if the field value is NOT NULL or has no default value $% \left( {{\left[ {{\left( {{\left( {{\left( {{\left( {{\left( {{\left( {$
E_OUT_OF_RANGE	-3008	The value is out of the range of the current type
E_DATA_CONFLICT_ERROR	-3010	Data conflict
E_WRITE_STALLED	-3011	Writes are delayed
E_IMPROPER_DATA_TYPE	-3021	Incorrect data type

Error name	Error Code	Description
E_INVALID_SPACEVIDLEN	-3022	Invalid VID length
E_INVALID_FILTER	-3031	Invalid filter
E_INVALID_UPDATER	-3032	Invalid field update
E_INVALID_STORE	-3033	Invalid KV storage
E_INVALID_PEER	-3034	Peer invalid
E_RETRY_EXHAUSTED	-3035	Out of retries
E_TRANSFER_LEADER_FAILED	-3036	Leader change failed
E_INVALID_STAT_TYPE	-3037	Invalid stat type
E_INVALID_VID	-3038	VID is invalid
E_LOAD_META_FAILED	-3040	Failed to load meta information
E_FAILED_TO_CHECKPOINT	-3041	Failed to generate checkpoint
E_CHECKPOINT_BLOCKED	-3042	Generating checkpoint is blocked
E_FILTER_OUT	-3043	Data is filtered
E_INVALID_DATA	-3044	Invalid data
E_MUTATE_EDGE_CONFLICT	-3045	Concurrent write conflicts on the same edge
E_MUTATE_TAG_CONFLICT	-3046	Concurrent write conflict on the same vertex
E_OUTDATED_LOCK	-3047	Lock is invalid
E_INVALID_TASK_PARA	-3051	Invalid task parameter
E_USER_CANCEL	-3052	The user canceled the task
E_TASK_EXECUTION_FAILED	-3053	Task execution failed
E_PLAN_IS_KILLED	-3060	Execution plan was cleared
E_NO_TERM	-3070	The heartbeat process was not completed when the request was received
E_OUTDATED_TERM	-3071	Out-of-date heartbeat received from the old leader (the new leader has been elected)
E_WRITE_WRITE_CONFLICT	-3073	Concurrent write conflicts with later requests
E_RAFT_UNKNOWN_PART	-3500	Unknown partition
E_RAFT_LOG_GAP	-3501	Raft logs lag behind
E_RAFT_LOG_STALE	-3502	Raft logs are out of date
E_RAFT_TERM_OUT_OF_DATE	-3503	Heartbeat messages are out of date
E_RAFT_UNKNOWN_APPEND_LOG	-3504	Unknown additional logs
E_RAFT_WAITING_SNAPSHOT	-3511	Waiting for the snapshot to complete
E_RAFT_SENDING_SNAPSHOT	-3512	There was an error sending the snapshot
E_RAFT_INVALID_PEER	-3513	Invalid receiver
E_RAFT_NOT_READY	-3514	Raft did not start
E_RAFT_STOPPED	-3515	Raft has stopped

Error name	Error Code	Description
E_RAFT_BAD_ROLE	-3516	Wrong role
E_RAFT_WAL_FAIL	-3521	Write to a WAL failed
E_RAFT_HOST_STOPPED	-3522	The host has stopped
E_RAFT_TOO_MANY_REQUESTS	-3523	Too many requests
E_RAFT_PERSIST_SNAPSHOT_FAILED	-3524	Persistent snapshot failed
E_RAFT_RPC_EXCEPTION	-3525	RPC exception
E_RAFT_NO_WAL_FOUND	-3526	No WAL logs found
E_RAFT_HOST_PAUSED	-3527	Host suspended
E_RAFT_WRITE_BLOCKED	-3528	Writes are blocked
E_RAFT_BUFFER_OVERFLOW	-3529	Cache overflow
E_RAFT_ATOMIC_OP_FAILED	-3530	Atomic operation failed
E_LEADER_LEASE_FAILED	-3531	Leader lease expired
E_RAFT_CAUGHT_UP	-3532	Data has been synchronized on Raft
E_STORAGE_MEMORY_EXCEEDED	-3600	Storage memory exceeded
E_LOG_GAP	-4001	Drainer logs lag behind
E_LOG_STALE	-4002	Drainer logs are out of date
E_INVALID_DRAINER_STORE	-4003	The drainer data storage is invalid
E_SPACE_MISMATCH	-4004	Graph space mismatch
E_PART_MISMATCH	-4005	Partition mismatch
E_DATA_CONFLICT	-4006	Data conflict
E_REQ_CONFLICT	-4007	Request conflict
E_DATA_ILLEGAL	-4008	Illegal data
E_CACHE_CONFIG_ERROR	-5001	Cache configuration error
E_NOT_ENOUGH_SPACE	-5002	Insufficient space
E_CACHE_MISS	-5003	No cache hit
E_CACHE_WRITE_FAILURE	-5005	Write cache failed
E_NODE_NUMBER_EXCEED_LIMIT	-7001	Number of machines exceeded the limit
E_PARSING_LICENSE_FAILURE	-7002	Failed to resolve certificate
E_UNKNOWN	-8000	Unknown error

Last update: October 25, 2023



https://docs.nebula-graph.io/3.6.0