NebulaGraph

Nebula Graph Database Manual

2.5.0

Min Wu, Yao Zhou, Cooper Liang, Foesa Yang, Max Zhu

Table of contents

1. About

Danger

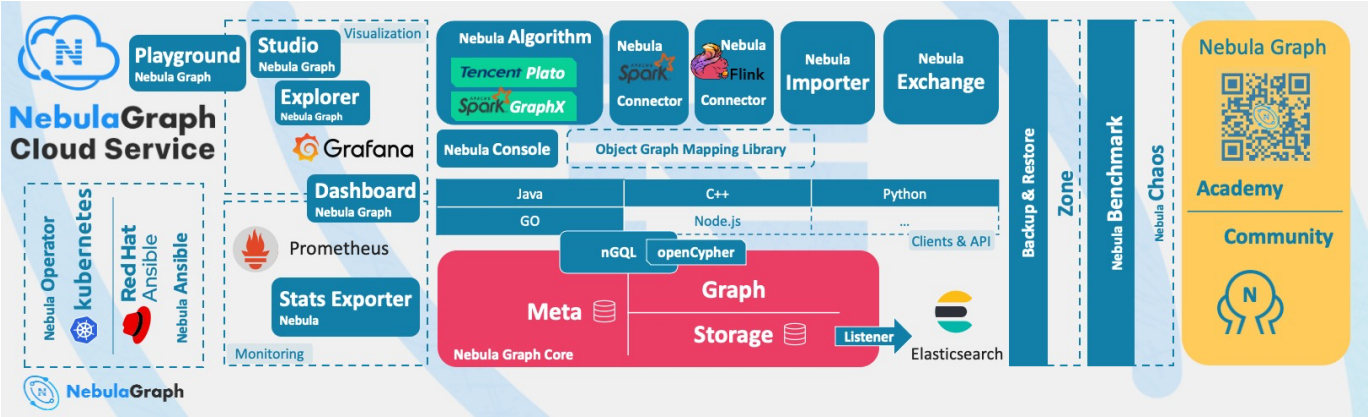
A new version has been released.

Last update: February 23, 2022

2. Introduction

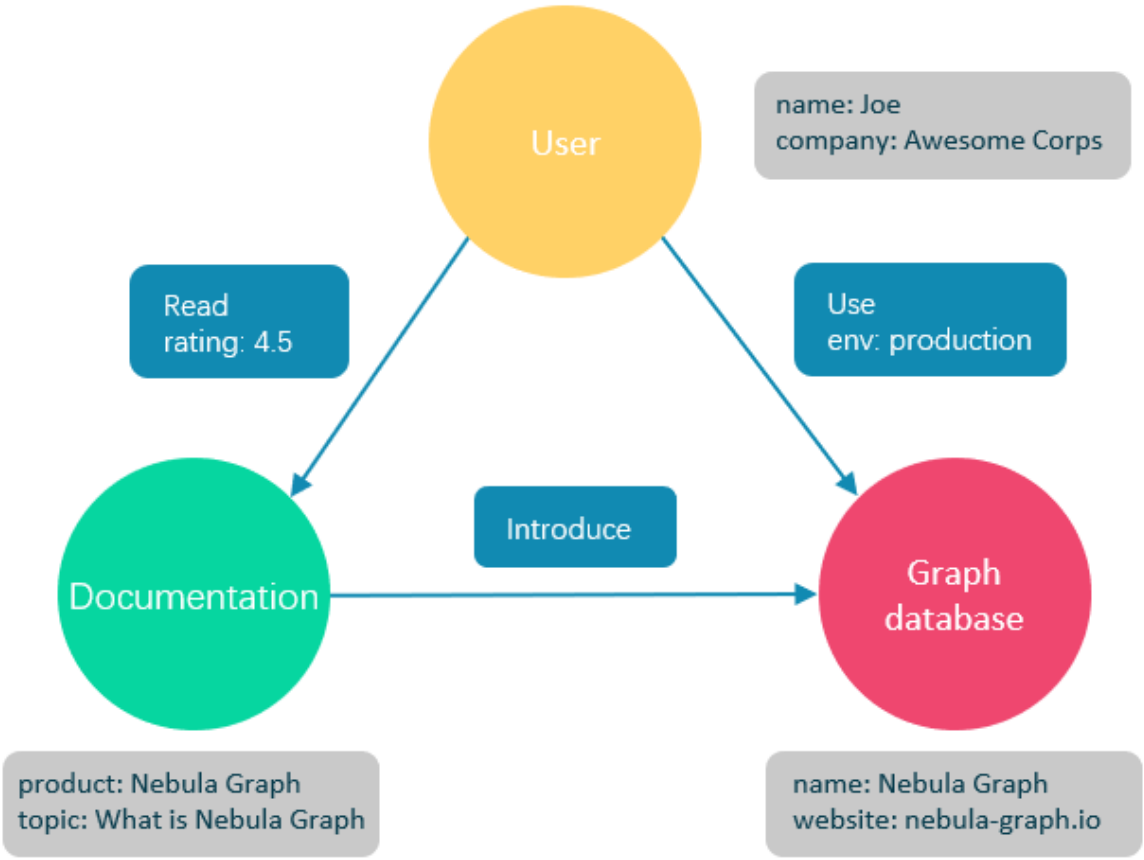
2.1 What is Nebula Graph

Nebula Graph 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 Nebula Graph, 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 the data management time-consuming and cost-ineffective.

Nebula Graph, 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 Benefits of Nebula Graph

Open-source

Nebula Graph 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 Nebula Graph. To join the opening of source code and ideas, surf the Nebula Graph GitHub page.

Outstanding performance

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

High scalability

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

Developer friendly

Nebula Graph supports clients in popular programming languages like Java, Python, C++, and Go, and more are being developed. For more information, see Nebula Graph clients.

Reliable access control

Nebula Graph 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 Nebula Graph have been released, such as Nebula Graph Studio, Nebula Console, and Nebula Exchange.

Besides, Nebula Graph 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. For more information, see Ecosystem development.

OpenCypher-compatible query language

The native Nebula Graph 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. Nebula Graph 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 Nebula Graph 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

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

2.1.3 Use cases

Nebula Graph 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 Nebula Graph.

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 Nebula Graph, fraud rings and other sophisticated scams can be easily detected.

Real-time recommendation

Nebula Graph 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 Nebula Graph. 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 are typical graph data. Nebula Graph 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
- Blog
- Forum
- GitHub

Last update: November 3, 2021

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 Nebula Graph.

2.2.1 Data structures

Nebula Graph 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 Nebula Graph, 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 must have at least one tag or multiple tags.
- **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**
>
> Tag and Edge type are similar to the vertex table and edge table in the relational databases.

2.2.2 Directed property graph

Nebula Graph 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:

$$G = < V, E, P_V, P_E >$$

- **V** is a set of vertices.
- **E** is a set of directed edges.
- P_V is the property of vertices.
- P_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

Nebula Graph supports only directed edges.

Compatibility

Nebula Graph 2.5.0 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.

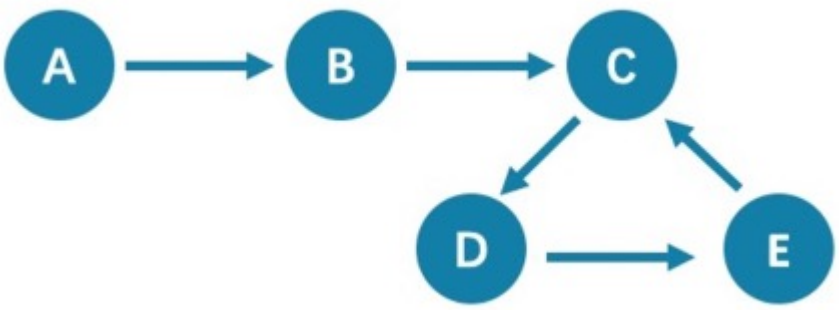
Last update: August 19, 2021

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 picture 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.

In the above picture C, D, and E form a cycle. So, this picture contains infinite paths, such as `A->B->C->D->E`, `A->B->C->D->E->C`, and `A->B->C->D->E->C->D`.

> **Note**
>
> `GO` statements use `walk`.

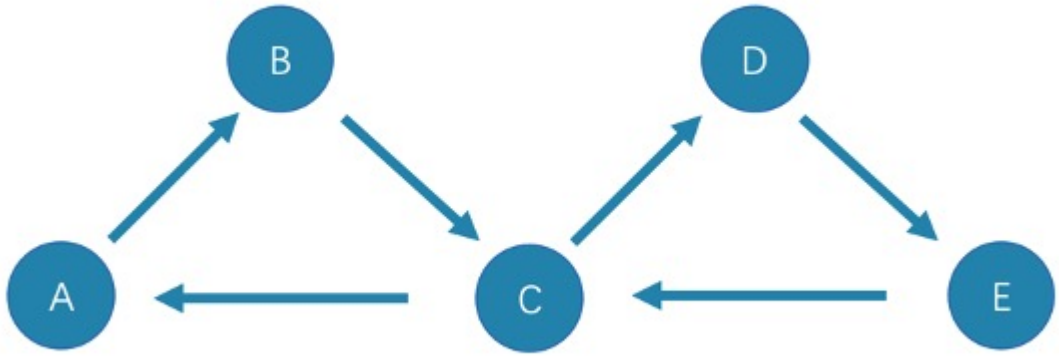
2.3.2 Trail

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

In the above picture, edges cannot be repeatedly visited. So, this picture contains finite paths. The longest path in this picture 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 picture 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 picture consists of 3 edges: `A->B->C->A` or `C->D->E->C`.

- circuit

  A `circuit` refers to a closed `trail`. 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 picture: `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 picture contains finite paths. The longest path in this picture consists of 4 edges: `A->B->C->D->E`.

Last update: September 2, 2021

2.4 VID

In Nebula Graph, 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`; a 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 Nebula Graph 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

- Nebula Graph 1.x only supports `INT64` while Nebula Graph 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 inte64 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 the data type of VIDs

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

2.4.5 Query `start vid` and global scan

In most cases, the execution plan of query statements in Nebula Graph (`MATCH`, `GO`, and `LOOKUP`) must query the `start vid` in a certain way.

There are only two ways to locate `start vid`:

1. For example, `GO 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`.

> **You cannot perform a global scan without `start vid`**
>
> For example, `match (n) return n;` returns an error because `start vid` cannot be located at this time. As a global scan, it is forbidden.

Last update: September 3, 2021

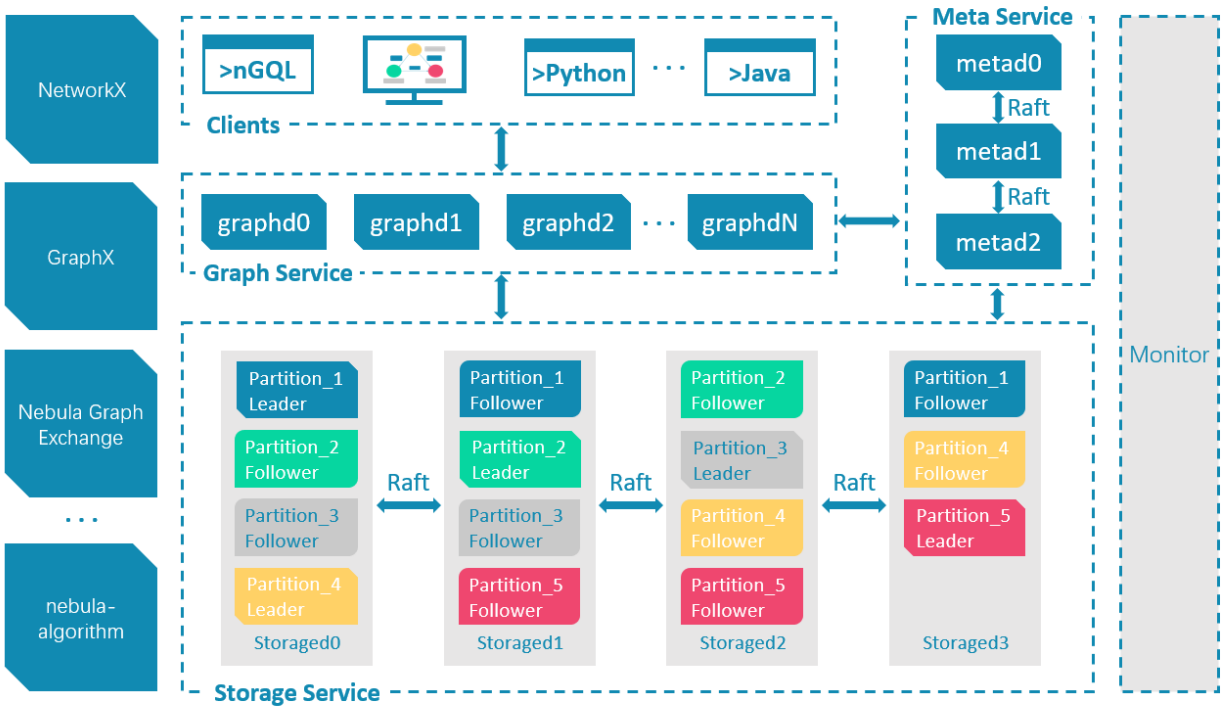
2.5 Nebula Graph architecture

2.5.1 Architecture overview

Nebula Graph 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 Nebula Graph cluster on a single machine or multiple machines using these binaries.

The following figure shows the architecture of a typical Nebula Graph cluster.



The Meta Service

The Meta Service in the Nebula Graph 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

Nebula Graph 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. The cost savings can be more distinct if the Nebula Graph Cloud service is used.

- 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.

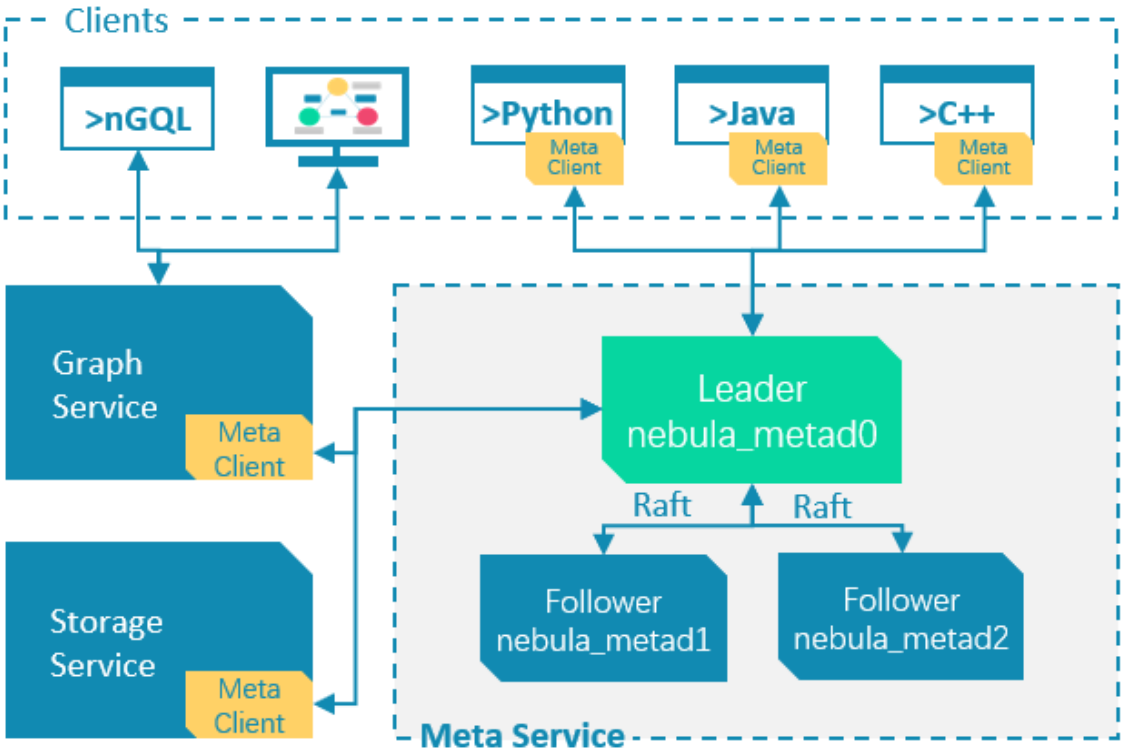
Last update: July 1, 2021

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 Nebula Graph. 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.

> **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 Nebula Graph access control, see Authentication and authorization.

MANAGES PARTITIONS

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

MANAGES GRAPH SPACES

Nebula Graph 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

Nebula Graph 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 Nebula Graph schema, see Data model.

MANAGES TTL-BASED DATA EVICTION

The Meta Service provides automatic data eviction and space reclamation based on TTL (time to live) options for Nebula Graph.

For more information on TTL, see TTL options.

MANAGES JOBS

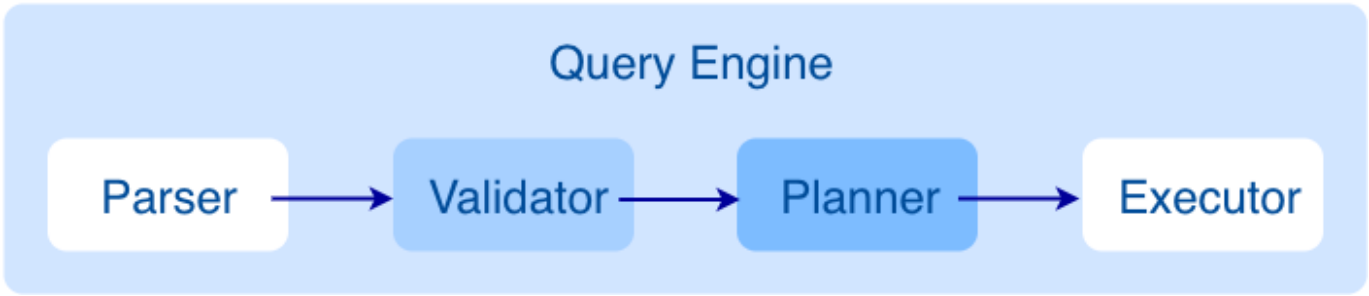
The Job Management module in the Meta Service is responsible for the creation, queuing, querying, and deletion of jobs.

Last update: August 24, 2021

2.5.3 Graph Service

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

The architecture of Graph Service



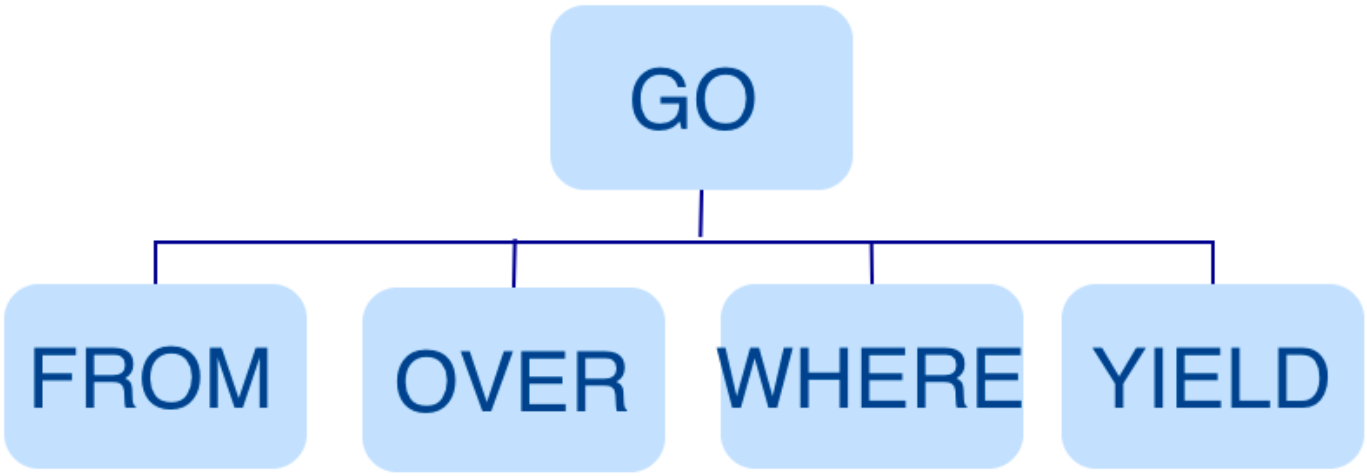
After a query is sent to Graph Service, it will be processed by the following four submodules:

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 operators.

Parser

After receiving a request, the statements will be parsed by the 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 its invalid syntax.

For example, the structure of the AST of `GO FROM "Tim" OVER like WHERE like.likeness > 8.0 YIELD like._dst` is shown in the following picture.



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 `YIELD` 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 like._dst AS ID; GO FROM $var.ID OVER serve YIELD serve._dst`, 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 `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 like._dst, like.likeness, serve._dst` 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 like._dst, like.likeness, serve._dst`.

- Validating input and output

  Validator will check the consistency of the clauses before and after the `|`.

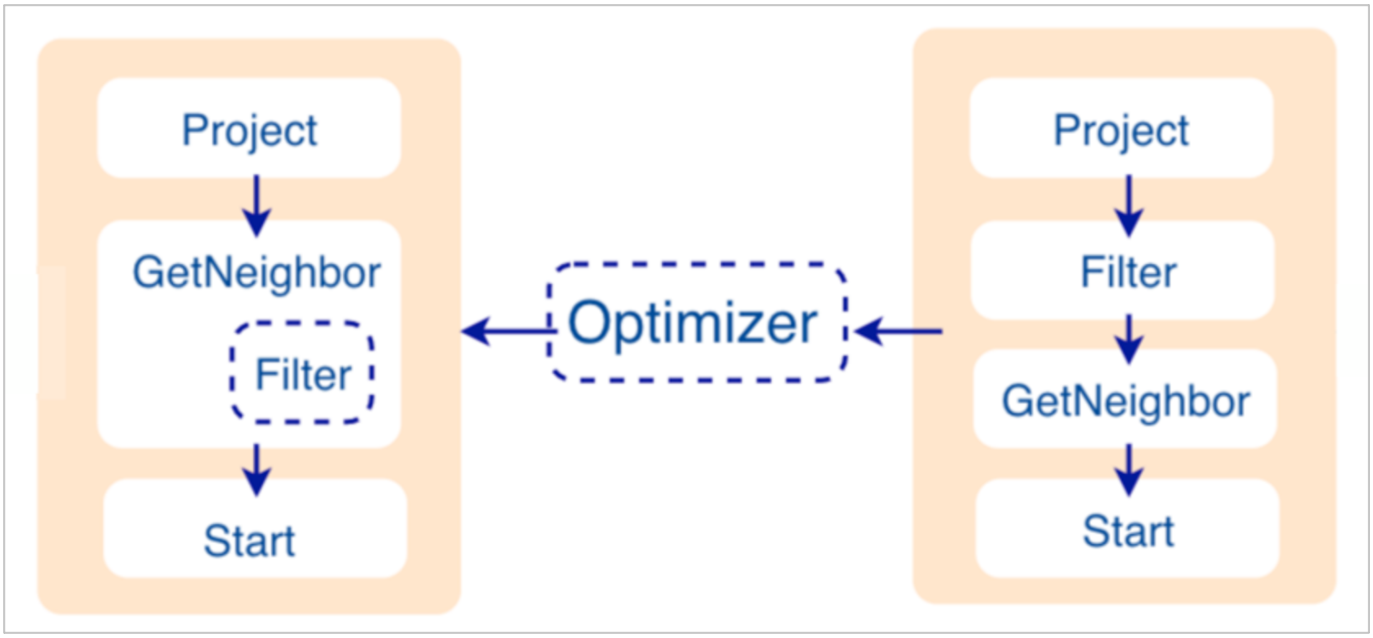
  In the statement `GO FROM "Tim" OVER like YIELD like._dst AS ID | GO FROM $-.ID OVER serve YIELD serve._dst`, Validator will verify whether `$-.ID` is defined in the clause before the `|`.

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 picture, 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 pre-defined 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.

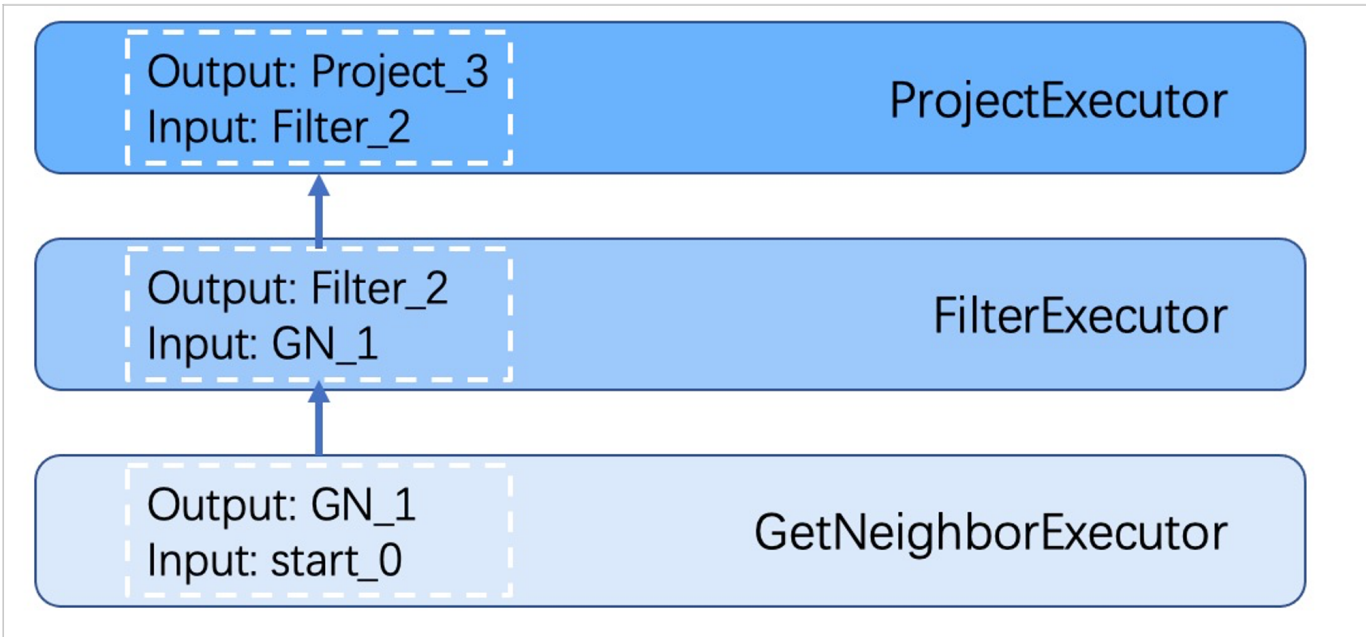
  RBO is a “bottom-up” exploration process. For each rule, the root node of the execution plan (in this case, the `Project` node) is the entry point, and step by step along with the node dependencies, it reaches the node at the bottom to see if it matches the rule.

  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.

> **Note**
>
> Nebula Graph 2.5.0 will not run optimization by default.

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
   |--context    //contexts for validation and execution
   |--daemons
   |--executor   //execution operators
   |--mock
   |--optimizer  //optimization rules
   |--parser     //lexical analysis and syntax analysis
   |--planner    //structure of the execution plans
   |--scheduler  //scheduler
   |--service
   |--util       //basic components
   |--validator  //validation of the statements
   |--visitor
```

Last update: September 2, 2021

2.5.4 Storage Service

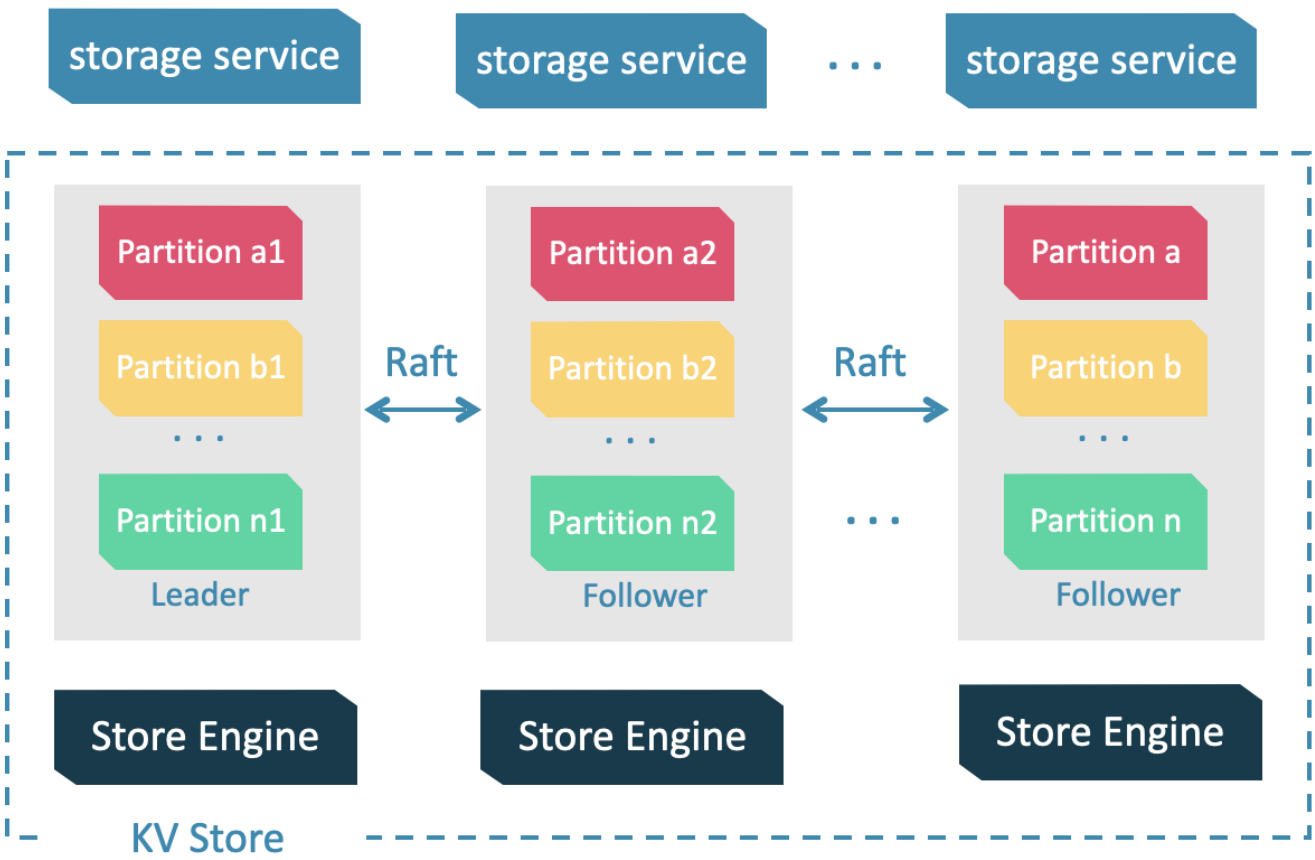
The persistent data of Nebula Graph have two parts. One is the Meta Service that stores the meta-related data.

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 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 Storage Service



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` : query the in-edge or out-edge of a set of vertices, return the edges and the corresponding properties, and support conditional filtering.
  - `insert vertex/edge` : insert a vertex or edge and its properties.
  - `getProps` : get 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. Users can develop their own local store plugins based on their needs.

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

KVStore

Nebula Graph 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 filtering-pushdown purpose. As a strong-typed database, how to provide Schema during pushdown is the key to efficiency for Nebula Graph.
- It has strong data consistency.

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

- For multiple local hard disks, Nebula Graph can make full use of its concurrent capacities through deploying multiple data directories.
- 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.

  > **Note**
  >
  > Nebula Graph does not support auto load balancing because auto data transfer will affect online business.

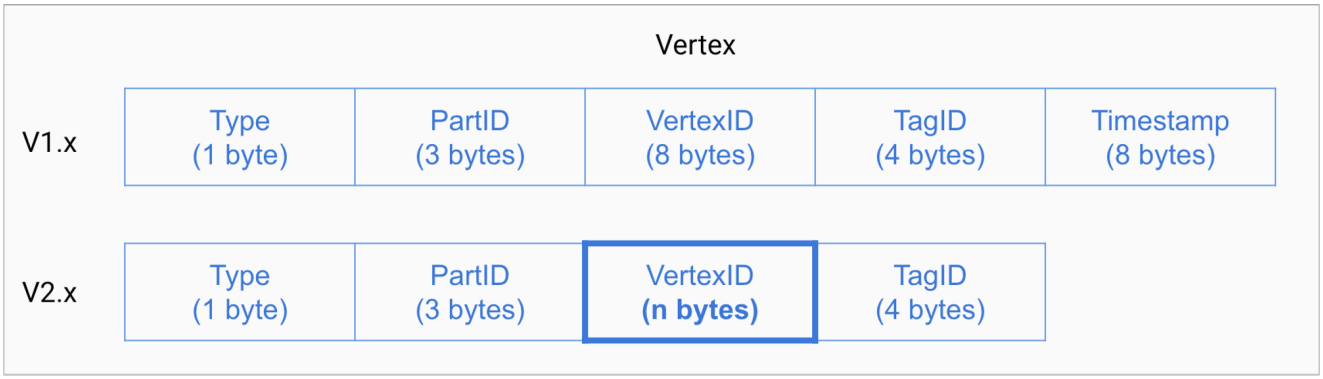
- Nebula Graph provides its own WAL mode so one can customize the WAL. Each partition owns its WAL.
- One Nebula Graph KVStore cluster supports multiple graph spaces, and each graph space has its own partition number and replica copies. Different graph spaces are isolated physically from each other in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

Data storage formats

Nebula Graph stores vertices and edges. Efficient property filtering is critical for a Graph Database. So, Nebula Graph uses keys to store vertices and edges, while uses values to store the related properties.

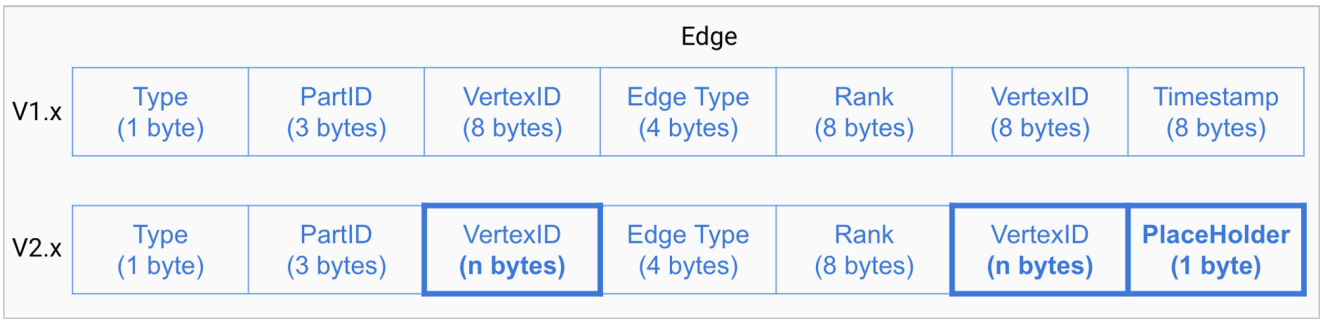
Nebula Graph 2.0 has changed a lot over its releases. The following will introduce the old and new data storage formats and cover their differences.

- Vertex format



Field	Description
`Type`	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`	Used to indicate vertex ID. For an integer VertexID, it occupies eight bytes. However, for a string VertexID, it is changed to `fixed_string` 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.

- Edge Format



Field	Description
`Type`	One byte, used to indicate the key type.
`PartID`	Three bytes, used to indicate the sharding partition. This field can be used to scan the partition data based on the prefix when re-balancing the partition.
`VertexID`	Used to indicate vertex ID. The former VID refers to source VID in out-edge and dest VID in in-edge, while the latter VID refers to dest VID in out-edge and source VID in in-edge.
`Edge Type`	Four bytes, used to indicate edge type. Greater than zero means out-edge, less than zero means in-edge.
`Rank`	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 byte. Reserved.

Legacy version compatibility

The differences between Nebula Graph 1.x and 2.0 are as follows:

- In Nebula Graph 1.x, a vertex and an edge have the same `Type` byte, while in Nebula Graph 2.0, the Type byte differs from each other, which separates vertices and edges physically so that all tags of a vertex can be easily queried.
- Nebula Graph 1.x supports only int IDs, while Nebula Graph 2.0 is compatible with both int IDs and string IDs.
- Nebula Graph 2.0 removes `Timestamp` in both vertex and edge key formats.
- Nebula Graph 2.0 adds `PlaceHolder` to edge key format.
- Nebula Graph 2.0 has changed the formats of indexes for a range query.

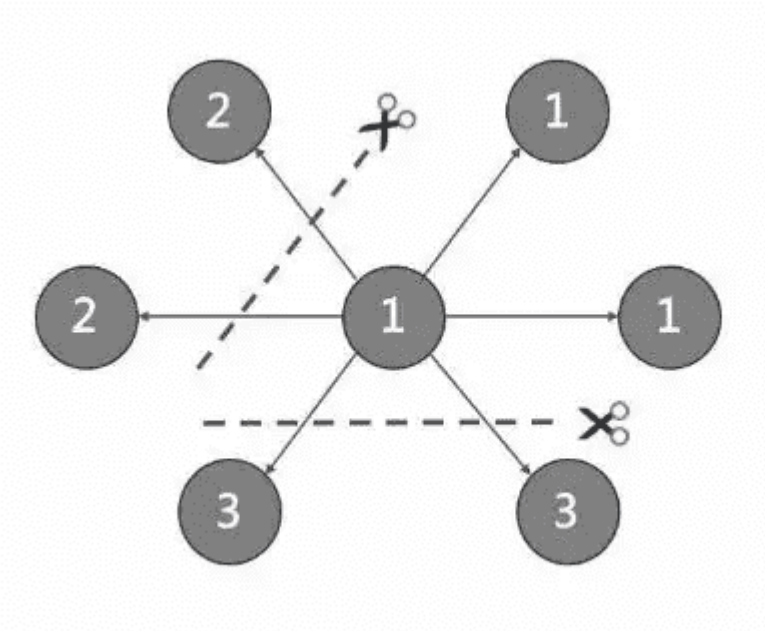
PROPERTY DESCRIPTIONS

Nebula Graph uses strong-typed Schema.

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

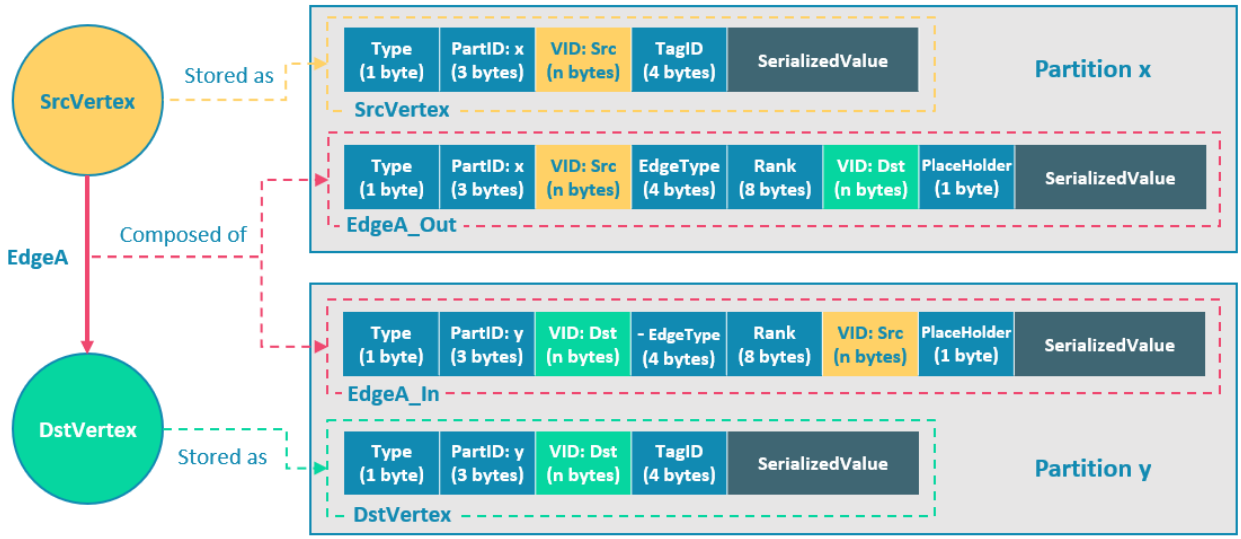
Data partitioning

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, Nebula Graph uses hash to shard the graph elements and store them in different partitions.



EDGE AND STORAGE AMPLIFICATION

In Nebula Graph, 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 picture below.



In this example, ScrVertex connects DstVertex via EdgeA, forming the path of `(SrcVertex)-[EdgeA]->(DstVertex)`. ScrVertex, 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 ScrVertex. 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, Nebula Graph 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

Nebula Graph 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.

> **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 Nebula Graph, 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.

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

Listener: As is a special role in Raft, it cannot vote or keep data consistency. In Nebula Graph, it reads Raft-wal from the Leader and synchronizes it to ElasticSearch cluster.

MULTI GROUP RAFT

Storage Service supports a distributed cluster architecture, so Nebula Graph 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, Nebula Graph achieves strong consistency and high availability. The functions of Raft are as follows.

Nebula Graph 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 Nebula Graph uses WAL to implement some special functions, batches need to be grouped, which is a feature of Nebula Graph.

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.

LISTENER

The Listener is designed for **storage horizontal scaling**. It takes a long time for the newly added machines to be synchronized with data. Therefore, these machines cannot join the group followers, otherwise, the availability of the entire cluster will decrease.

The Listener will write into the command WAL. If the leader finds a command of `add learner` when writing the WAL, it will add the listener to its peers and mark it as a Listener. Listeners cannot join the quorum votes, but logs will still be sent to them as usual. Listeners themselves will not initiate elections.

Raft listener can write the data into Elasticsearch cluster after receiving them from Learner to implement full-text search. For more information, see Deploy Raft Listener.

TRANSFER LEADERSHIP

Transfer leadership is extremely important for balance. When moving a partition from one machine to another, Nebula Graph 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. Nebula Graph 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

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

- 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 Storage Service, only the replicas of the leader can read and write, while in HDFS all the replicas can do so.
- In 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.
- 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, 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: September 10, 2021

3. Quick start

3.1 Quick start workflow

The quick start introduces the simplest workflow to use Nebula Graph, including deploying Nebula Graph, connecting to Nebula Graph, and doing basic CRUD.

3.1.1 Documents

Users can quickly deploy and use Nebula Graph in the following steps.

1. Deploy Nebula Graph

   Users can use the RPM or DEB file to quickly deploy Nebula Graph. For other ways to deploy Nebula Graph and corresponding preparations, see deployment and installation.

2. Start Nebula Graph

   Users need to start Nebula Graph after deployment.

3. Connect to Nebula Graph

   Then users can use clients to connect to Nebula Graph. Nebula Graph supports a variety of clients. This topic will describe how to use Nebula Console to connect to Nebula Graph.

4. CRUD in Nebula Graph

   Users can use nGQL (Nebula Graph Query Language) to run CRUD after connecting to Nebula Graph.

Last update: September 2, 2021

3.2 Step 1: Install Nebula Graph

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

3.2.1 Prerequisites

Prepare the right resources.

> **Note**
>
> The console is not complied or packaged with Nebula Graph server binaries. You can install nebula-console by yourself.

3.2.2 Download the package from cloud service

- Download the released version.

  URL:

```
//Centos 6
https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.el6.x86_64.rpm

//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 release package 2.5.0 for `Centos 7.5`:

```
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.el7.x86_64.rpm.sha256sum.txt
```

  download release package `2.5.0` for `Ubuntu 1804`:

```
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.ubuntu1804.amd64.deb.sha256sum.txt
```

- Download the nightly version.

> **Danger**
>
> - Nightly versions are usually used to test new features. Don't use it for production.
> - Nightly versions may not be build successfully every night. And the names may change from day to day.

URL:

```
//Centos 6
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el6.x86_64.rpm

//Centos 7
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu2004.amd64.deb
```

For example, download the `Centos 7.5` package developed and built in `2021.03.28`:

```
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm.sha256sum.txt
```

For example, download the `Ubuntu 1804` package developed and built in `2021.03.28`:

```
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb.sha256sum.txt
```

3.2.3 Install Nebula Graph

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

```
sudo rpm -ivh --prefix=<installation_path> <package_name>
```

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

```
sudo dpkg -i --instdir==<installation_path> <package_name>
```

> **Note**
>
> The default installation path is `/usr/local/nebula/`.

3.2.4 What's next

- start Nebula Graph
- connect to Nebula Graph

Last update: September 2, 2021

3.3 Step 2: Manage Nebula Graph Service

You can use the `nebula.service` script to start, stop, restart, terminate, and check the Nebula Graph services. This topic takes starting, stopping and checking the Nebula Graph services for examples.

`nebula.service` is stored in the `/usr/local/nebula/` directory by default, which is also the default installation path of Nebula Graph. If you have customized the path, use the actual path in your environment.

3.3.1 Syntax

```
$ sudo /usr/local/nebula/scripts/nebula.service
[-v] [-c <config_file_path>]
<start|stop|restart|status|kill>
<metad|graphd|storaged|all>
```

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 Nebula Graph services as the target services.

3.3.2 Start Nebula Graph

In non-container environment

Run the following command to start Nebula Graph.

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

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to start Nebula Graph.

```
nebula-docker-compose]$ docker-compose up -d
Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Creating network "nebula-docker-compose_nebula-net" with the default driver
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_graphd1_1   ... done
Creating nebula-docker-compose_storaged1_1 ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
```

3.3.3 Stop Nebula Graph

Danger

Don't run `kill -9` to forcibly terminate the processes, otherwise, there is a low probability of data loss.

In non-container environment

Run the following command to stop Nebula Graph.

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

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to stop Nebula Graph.

```
nebula-docker-compose]$ docker-compose down
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad1_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing network nebula-docker-compose_nebula-net
```

If you are using a development or nightly version for testing and have compatibility issues, try to run `docker-compose down -v` to **DELETE** all data stored in Nebula Graph and import data again.

3.3.4 Check the service status

In non-container environment

Run the following command to check the service status of Nebula Graph.

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

- Nebula Graph is running normally if the following information is returned.

```
[INFO] nebula-metad: Running as 26601, Listening on 9559
[INFO] nebula-graphd: Running as 26644, Listening on 9669
[INFO] nebula-storaged: Running as 26709, Listening on 9779
```

- If the return information is similar to the following one, there is a problem.

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

The Nebula Graph 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 return information to troubleshoot problems.

You may also go to the Nebula Graph community for help.

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to check the service status of Nebula Graph.

```
nebula-docker-compose]$ docker-compose ps
            Name                           Command                  State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/tcp, 0.0.0.0:49224->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/tcp, 0.0.0.0:49230->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49212->19559/tcp, 0.0.0.0:49211->19560/tcp, 0.0.0.0:49213->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49209->19559/tcp, 0.0.0.0:49208->19560/tcp, 0.0.0.0:49210->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49206->19559/tcp, 0.0.0.0:49205->19560/tcp, 0.0.0.0:49207->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49218->19779/tcp, 0.0.0.0:49217->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49219->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49215->19779/tcp, 0.0.0.0:49214->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                     0.0.0.0:49216->9779/tcp, 9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49226->19779/tcp, 0.0.0.0:49225->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49227->9779/tcp, 9780/tcp
```

Use the `CONTAINER ID` to log in the container and troubleshoot.

```
nebula-docker-compose]$ docker exec -it 2a6c56c405f5 bash
[root@2a6c56c405f5 nebula]#
```

3.3.5 What's next

Connect to Nebula Graph

Last update: September 2, 2021

3.4 Step 3: Connect to Nebula Graph

Nebula Graph supports multiple types of clients, including a CLI client, a GUI client, and clients developed in popular programming languages. This topic provides an overview of Nebula Graph clients and basic instructions on how to use the native CLI client, Nebula Console.

3.4.1 Nebula Graph clients

You can use supported clients or console to connect to Nebula Graph.

3.4.2 Use Nebula Console to connect to Nebula Graph

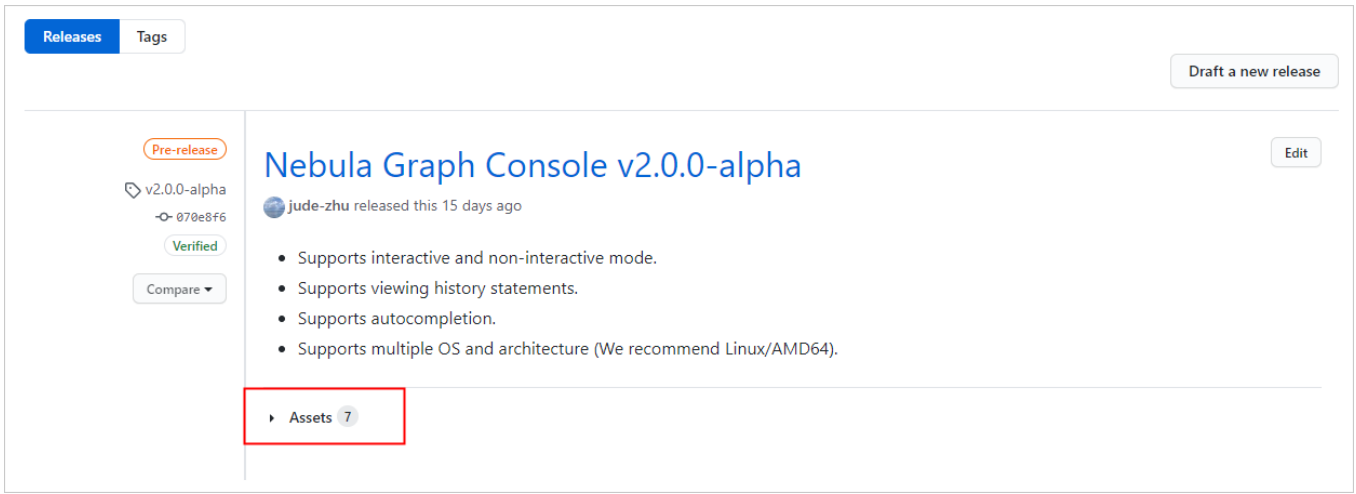
Prerequisites

- You have started the Nebula Graph services. For how to start the services, see Start and Stop Nebula Graph.
- The machine you plan to run Nebula Console on has network access to the Nebula Graph services.

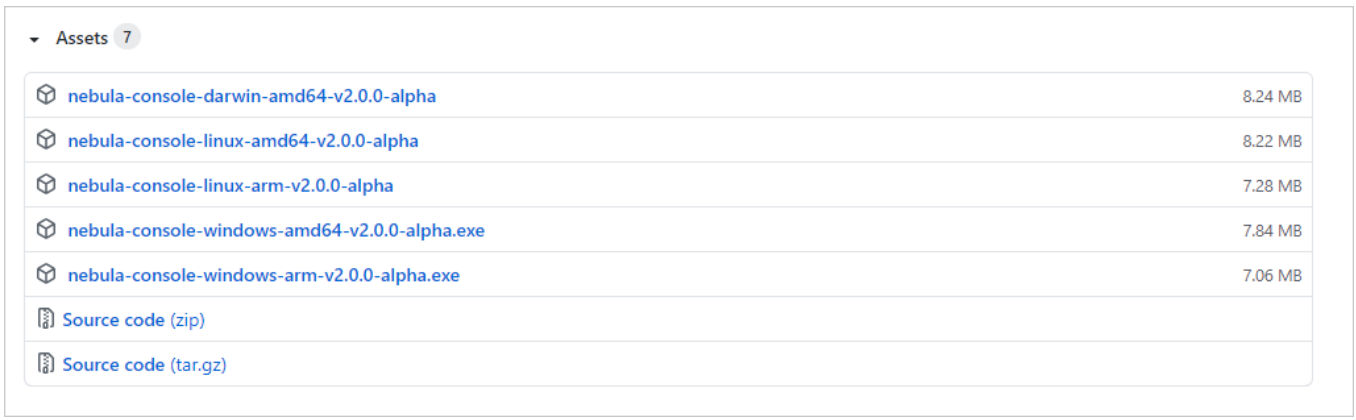
Steps

1. On the nebula-console page, select a Nebula Console version and click **Assets**.

> **Note**
>
> We recommend that you select the **latest** release.



2. In the **Assets** area, find the correct binary file for the machine where you want to run Nebula 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 Nebula 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 Nebula Graph.

   • 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]
```

The description of the parameters is as follows.

Option	Description
`-h`	Shows the help menu.
`-addr`	Sets the IP address of the graphd service. The default address is 127.0.0.1.
`-port`	Sets the port number of the graphd service. The default port number is 9669.
`-u/-user`	Sets the username of your Nebula Graph account. Before enabling authentication, you can use any existing username. The default username is `root`.
`-p/-password`	Sets the password of your Nebula Graph 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 second. 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. You'll get the return messages and the connection stops then.

You can find more details in the Nebula Console Repository.

3.4.3 Nebula Console commands

Nebula Console can export CSV file, DOT file, and import too.

> **Note**
>
> The commands are case insensitive.

Export a CSV file

> **Note**
>
> - A CSV file will be saved in the working directory, i.e., what linux command `pwd` show;
> - This command only works for the next query statement.

The command to export a csv file.

```
nebula> :CSV <file_name.csv>
```

Export a DOT file

> **Note**
>
> - A DOT file will be saved in the working directory, i.e., what linux command `pwd` show;
> - You can copy the contents of DOT file, and paste in GraphvizOnline, to visualize the excution plan;
> - This command only works for the next query statement.

The command to export a DOT file.

```
nebula> :dot <file_name.dot>
```

For example,

```
nebula> :dot a.dot
nebula> PROFILE FORMAT="dot" GO FROM "player100" OVER follow;
```

Importing a testing dataset

The testing dataset is named `nba`. Details about schema and data can be seen by commands `SHOW`.

Using the following command to import the testing dataset,

```
nebula> :play nba
```

Run a command multiple times

Sometimes, you want to run a command multiple times. Run the following command.

```
nebula> :repeat N
```

For example,

```
nebula> :repeat 3
nebula> GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 2602/3214 us)

Fri, 20 Aug 2021 06:36:05 UTC

+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 583/849 us)

Fri, 20 Aug 2021 06:36:05 UTC

+-------------+
| follow._dst |
+-------------+
| "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 to wait

Sleep N seconds.

It is usually used when altering schema. Since schema is altered in async way, and take effects in the next heartbeat cycle.

```
nebula> :sleep N
```

3.4.4 Disconnect Nebula Console from Nebula Graph

You can use `:EXIT` or `:QUIT` to disconnect from Nebula Graph. For convenience, Nebula Console supports using these commands in lower case without the colon (":"), such as `quit`.

```
nebula> :QUIT

Bye root!
```

3.4.5 FAQ

How can I install Nebula Console from the source code

To download and compile the latest source code of Nebula Console, follow the instructions on the nebula console GitHub page.

Last update: September 2, 2021

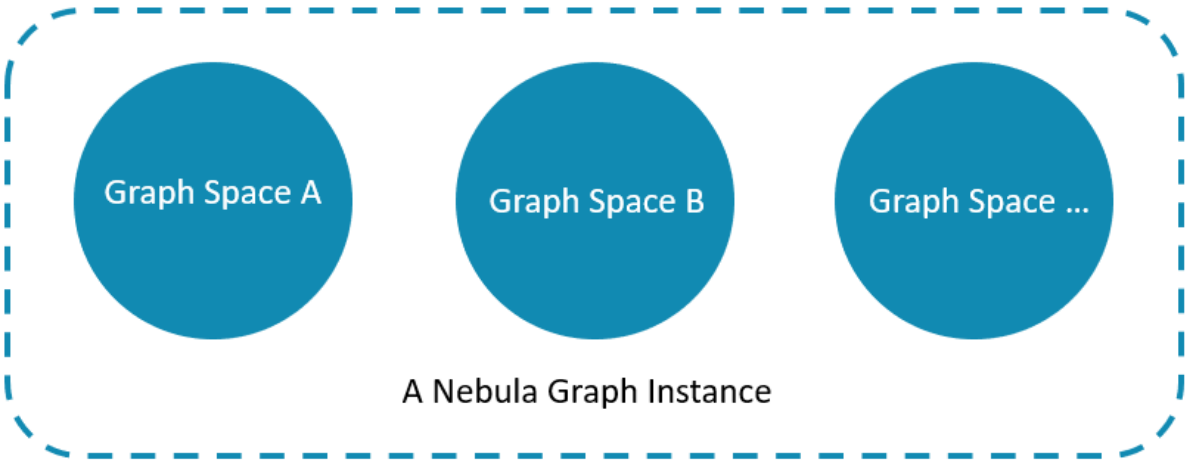
3.5 Step 4: Use nGQL (CRUD)

This topic will describe the basic CRUD operations in Nebula Graph.

For more information, see nGQL guide.

3.5.1 Graph space and Nebula Graph schema

A Nebula Graph 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.

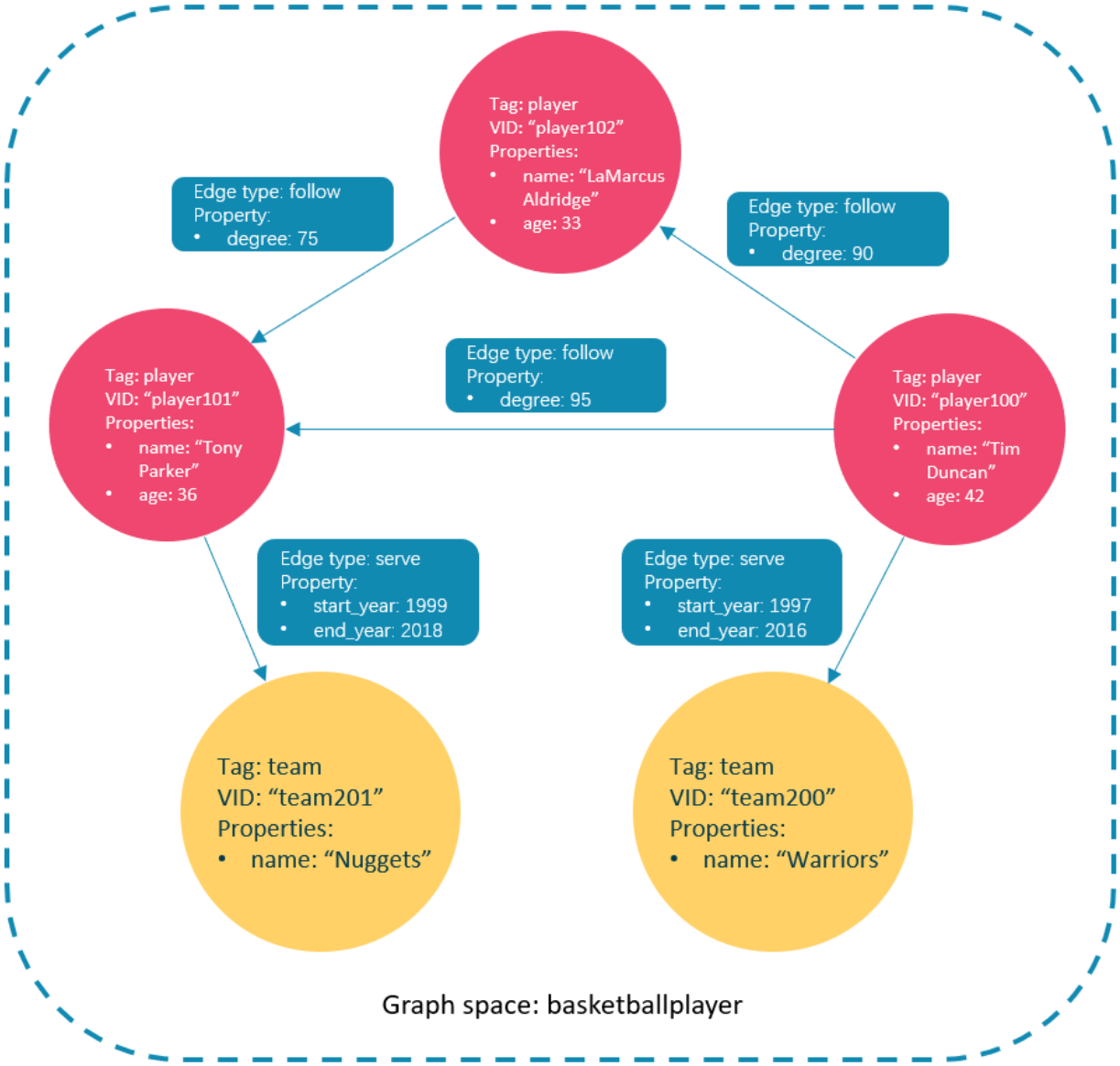


To insert data into a graph space, define a schema for the graph database. Nebula Graph schema is based on the following components.

Schema component	Description
Vertex	Represents an entity in the real world. A vertex can have one or more 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 **directed** 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.



3.5.2 Check the machine status in the Nebula Graph cluster

> **Note**
>
> First, we recommend that you check the machine status to make sure that all the Storage services are connected to the Meta services. Run `SHOW HOSTS` as follows.

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| Host        | Port      | Status    | Leader count | Leader distribution  | Partition distribution |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 0            | "No valid partition" | "No valid partition"   |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
| "Total"     | __EMPTY__ | __EMPTY__ | 0            | __EMPTY__            | __EMPTY__              |
+-------------+-----------+-----------+--------------+----------------------+------------------------+
Got 4 rows (time spent 1061/2251 us)
```

From the **Status** column of the table in the return results, you can see that all the Storage services are online.

Asynchronous implementation of creation and alteration

> **Caution**
>
> Nebula Graph implements the following creation or alteration operations asynchronously in the **next** heartbeat cycle. The operations will not take effect until they finish.

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

> **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.

To make sure the follow-up operations work as expected, take one of the following approaches:

- Run `SHOW` or `DESCRIBE` statements accordingly to check the status of the objects, and make sure the creation or alteration is complete. If it is not, wait a few seconds and try again.
- Wait for two heartbeat cycles, i.e., 20 seconds.

3.5.3 Create and use a graph space

nGQL syntax

- Create a graph space:

```
CREATE SPACE [IF NOT EXISTS] <graph_space_name> (
[partition_num = <partition_number>,]
[replica_factor = <replica_number>,]
vid_type = {FIXED_STRING(<N>) | INT64}
)
[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));
Execution succeeded (time spent 2817/3280 us)
```

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

```
nebula> SHOW HOSTS;
+-------------+-----------+-----------+--------------+----------------------------------+------------------------+
| Host        | Port      | Status    | Leader count | Leader distribution              | Partition distribution |
+-------------+-----------+-----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"             | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"             | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779      | "ONLINE"  | 5            | "basketballplayer:5"             | "basketballplayer:5"   |
+-------------+-----------+-----------+--------------+----------------------------------+------------------------+
| "Total"     |           |           | 15           | "basketballplayer:15"            | "basketballplayer:15"  |
+-------------+-----------+-----------+--------------+----------------------------------+------------------------+
Got 4 rows (time spent 1633/2867 us)
```

   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;
Execution succeeded (time spent 1229/2318 us)
```

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

```
nebula> SHOW SPACES;
+--------------------+
| Name               |
+--------------------+
| "basketballplayer" |
+--------------------+
Got 1 rows (time spent 977/2000 us)
```

3.5.4 Create tags and edge types

nGQL syntax

```
CREATE {TAG | EDGE} {<tag_name> | <edge_type>}(<property_name> <data_type>
[, <property_name> <data_type> ...])
[COMMENT = '<comment>'];
```

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

Examples

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

Component name	Type	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);
Execution succeeded (time spent 20708/22071 us)

nebula> CREATE TAG team(name string);
Execution succeeded (time spent 5643/6810 us)

nebula> CREATE EDGE follow(degree int);
Execution succeeded (time spent 12665/13934 us)

nebula> CREATE EDGE serve(start_year int, end_year int);
Execution succeeded (time spent 5858/6870 us)
```

3.5.5 Insert vertices and edges

Users 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_name> (<property_name>[, <property_name>...])
[, <tag_name> (<property_name>[, <property_name>...]), ...]
{VALUES | VALUE} <vid>: (<property_value>[, <property_value>...])
[, <vid>: (<property_value>[, <property_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> (<property_name>[, <property_name>...])
{VALUES | VALUE} <src_vid> -> <dst_vid>[@<rank>] : (<property_value>[, <property_value>...])
[, <src_vid> -> <dst_vid>[@<rank>] : (<property_name>[, <property_name>...]), ...];
```

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);
Execution succeeded (time spent 28196/30896 us)

nebula> INSERT VERTEX player(name, age) VALUES "player101":("Tony Parker", 36);
Execution succeeded (time spent 2708/3834 us)

nebula> INSERT VERTEX player(name, age) VALUES "player102":("LaMarcus Aldridge", 33);
Execution succeeded (time spent 1945/3294 us)
```

```
nebula> INSERT VERTEX team(name) VALUES "team200":("Warriors"), "team201":("Nuggets");
Execution succeeded (time spent 2269/3310 us)
```

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

```
nebula> INSERT EDGE follow(degree) VALUES "player100" -> "player101":(95);
Execution succeeded (time spent 3362/4542 us)

nebula> INSERT EDGE follow(degree) VALUES "player100" -> "player102":(90);
Execution succeeded (time spent 2974/4274 us)

nebula> INSERT EDGE follow(degree) VALUES "player102" -> "player101":(75);
Execution succeeded (time spent 1891/3096 us)

nebula> INSERT EDGE serve(start_year, end_year) VALUES "player100" -> "team200":(1997, 2016), "player101" -> "team201":(1999,  2018);
Execution succeeded (time spent 6064/7104 us)
```

3.5.6 Read data

- The GO statement can traverse the database based on specific conditions. A `GO` 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 Nebula Graph. Therefore, its performance still needs optimization.

nGQL syntax

- `GO`

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [REVERSELY] [BIDIRECT]
[WHERE <expression> [AND | OR expression ...])]
YIELD [DISTINCT] <return_list>;
```

- `FETCH`
  - Fetch properties on tags:

```
FETCH PROP ON {<tag_name> | <tag_name_list> | *} <vid_list>
[YIELD [DISTINCT] <return_list>];
```

  - Fetch properties on edges:

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

- `LOOKUP`

```
LOOKUP ON {<tag_name> | <edge_type>}
WHERE <expression> [AND expression ...])]
[YIELD <return_list>];
```

- `MATCH`

```
MATCH <pattern> [<WHERE clause>] RETURN <output>;
```

Examples of `GO` statement

- Search for the players that the player with VID `player100` follows.

```
nebula> GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
```

```
| "player102" |
+-------------+
Got 2 rows (time spent 12097/14220 us)
```

- Filter the players that the player with VID `player100` follows whose age is equal to or greater than 35. Rename the corresponding columns in the results with `Teammate` and `Age`.

```
nebula> GO FROM "player100" OVER follow WHERE $$.player.age >= 35 \
        YIELD $$.player.name AS Teammate, $$.player.age AS Age;
+---------------+-----+
| Teammate      | Age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+
Got 1 rows (time spent 8206/9335 us)
```

Clause/Sign	Description
`YIELD`	Specifies what values or results you want to return from the query.
`$$`	Represents the target vertices.
`\`	A line-breaker.

- Search for the players that the player with VID `player100` 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 "player100" OVER follow YIELD follow._dst AS id | \
        GO FROM $-.id OVER serve YIELD $$.team.name AS Team, \
        $^.player.name AS Player;
+-----------+---------------+
| Team      | Player        |
+-----------+---------------+
| "Nuggets" | "Tony Parker" |
+-----------+---------------+
Got 1 rows (time spent 5055/8203 us)
```

Clause/Sign	Description
`$^`	Represents the source vertex of the edge.
`|`	A pipe symbol can combine multiple queries.
`$-`	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 FROM "player100" OVER follow YIELD follow._dst AS id; \
        GO FROM $var.id OVER serve YIELD $$.team.name AS Team, \
        $^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
Got 1 rows (time spent 3103/3711 us)
```

Example of `FETCH` statement

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

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

```
+-------------------------------------------------+
Got 1 rows (time spent 2006/2406 us)
```

> **Note**
>
> The examples of `LOOKUP` and `MATCH` statements are in indexes.

3.5.7 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 <source vid> -> <destination vid> [@rank] OF <edge_type>
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";
Execution succeeded (time spent 3483/3914 us)

nebula> FETCH PROP ON player "player100";
+-----------------------------------------+
| vertices_                               |
+-----------------------------------------+
| ("player100" :player{age: 42, name: "Tim"}) |
```

```
+--------------------------------------------+
Got 1 rows (time spent 2463/3042 us)
```

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

```
nebula> UPDATE EDGE "player100" -> "player101" OF follow SET degree = 96;
Execution succeeded (time spent 3932/4432 us)

nebula> FETCH PROP ON follow "player100" -> "player101";
+-------------------------------------------------+
| edges_                                          |
+-------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 96}] |
+-------------------------------------------------+
Got 1 rows (time spent 2205/2800 us)
```

- Insert a vertex with VID `player111` and `UPSERT` it.

```
nebula> INSERT VERTEX player(name, age) VALUES "player111":("Ben Simmons", 22);
Execution succeeded (time spent 2115/2900 us)

Wed, 21 Oct 2020 11:11:50 UTC

nebula> UPSERT VERTEX "player111" SET player.name = "Dwight Howard", player.age = $^.player.age + 11 \
        WHEN $^.player.name == "Ben Simmons" AND $^.player.age > 20 \
        YIELD $^.player.name AS Name, $^.player.age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| Dwight Howard | 33  |
+---------------+-----+
Got 1 rows (time spent 1815/2329 us)
```

3.5.8 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 "team1", "team2";
Execution succeeded (time spent 4337/4782 us)
```

- Delete edges:

```
nebula> DELETE EDGE follow "team1" -> "team2";
Execution succeeded (time spent 3700/4101 us)
```

3.5.9 About indexes

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

> **Must-read for using indexes**
>
> Both `MATCH` and `LOOKUP` statements depend on the indexes. But indexes can dramatically reduce the write performance (as much as 90% or even more). **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>;
```

Examples

Create and rebuild indexes for the `name` property on all vertices with the tag `player`.

```
nebula> CREATE TAG INDEX player_index_0 on player(name(20));
nebula> REBUILD TAG INDEX player_index_0;
```

> **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_name_0` on the player name property.

```
nebula> CREATE TAG INDEX player_name_0 on player(name(10));
Execution succeeded (time spent 3465/4150 us)
```

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

```
nebula> REBUILD TAG INDEX player_name_0
+------------+
| New Job Id |
+------------+
| 31         |
+------------+
Got 1 rows (time spent 2379/3033 us)
```

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

```
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
+-------------+---------------+------------+
| VertexID    | player.name   | player.age |
+-------------+---------------+------------+
| "player101" | "Tony Parker" | 36         |
+-------------+---------------+------------+
```

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

```
nebula> MATCH (v:player{name:"Tony Parker"}) RETURN v;
+-------------------------------------------------------+
| v                                                     |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------------+
Got 1 rows (time spent 5132/6246 us)
```

Last update: September 2, 2021

4. nGQL guide

4.1 nGQL overview

4.1.1 Nebula Graph Query Language (nGQL)

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

What is nGQL

nGQL is a declarative graph query language for Nebula Graph. 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 Nebula Graph team if you find a new issue of this type. Nebula Graph 2.0 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 Nebula Graph. After downloading the example data, you can import it to Nebula Graph by using the `-f` option in Nebula Graph Console.

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
|	complete alternative elements
...	may be repeated any number of times

For example, create vertices or edges in nGQL syntax:

```
CREATE {TAG | EDGE} {<tag_name> | <edge_type>}(<property_name> <data_type>
[, <property_name> <data_type> ...]);
```

Example statement:

```
nebula> CREATE TAG 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 Nebula Graph. 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`

Undefined behavior

Do not put together `native nGQL` and `openCypher compatible sentences` in one composite statement because this behavior is undefined.

IS NGQL COMPATIBLE WITH OPENCYPHER 9 COMPLETELY?

NO.

nGQL is partially compatible with DQL in openCypher 9

nGQL is designed to be compatible with part of DQL (match) and is not planned to be compatible with any DDL, DML, or DCL.

Multiple known incompatible items are listed in Nebula Graph Issues. Submit an issue with the `incompatible` tag if you find a new issue of this type. Users can search in this manual with the keyword `compatibility` to find major compatibility issues.

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	`^`	`^` 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, and `OPTIONAL MATCH` are not supported.
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 query	support	not supported

Compatibility

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

1. 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.
3. 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 Nebula Graph 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:
      | v                                                |
      | ("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.name AS Name
      """
    Then the result should be, in any order:

      | Type     | Name        |
      | "follow" | "Ray Allen" |
      | "serve"  | "Lakers"    |
      | "serve"  | "Heat"      |
      | "serve"  | "Cavaliers" |
```

```
Feature:  Comparison of where clause

  Background:
    Given a graph with space named "basketballplayer"

    Scenario: push edge props filter down
      When profiling query:
        """
        GO FROM "player100" OVER follow
        WHERE follow.degree IN [v IN [95,99] WHERE v > 0]
        YIELD follow._dst, follow.degree
        """
      Then the result should be, in any order:
        | follow._dst | follow.degree |
        | "player101" | 95            |
        | "player125" | 95            |
      And the execution plan should be:
        | id | name         | dependencies | operator info                                                  |
        | 0  | Project      | 1            |                                                                |
        | 1  | GetNeighbors | 2            | {"filter": "(follow.degree IN [v IN [95,99] WHERE (v>0)])"} |
        | 2  | Start        |              |                                                                |
```

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 Nebula Graph 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 NEBULA GRAPH SUPPORT W3C RDF (SPARQL) OR GRAPHQL?

No. And no plan to support that.

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

Nebula Graph Query Language does not support `SPARQL` nor `GraphQL`.

Last update: August 27, 2021

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 Nebula Graph, 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 `label` 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)
```

> **OpenCypher compatibility**
>
> The `MATCH` statement in nGQL does not support matching multiple tags with `(a:User:Admin)`. If users need to match multiple tags, use filtering conditions, such as `WHERE "User" IN tags(n) AND "Admin" IN tags(n)`.

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

```
(a {name: 'Andres', sport: 'Brazilian Ju-Jitsu'})
```

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

```
(a)-[{blocked: false}]->(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.

The lower bound can be omitted. For example, to describe paths of length 5 or less, use:

```
(a)-[*..5]->(b)
```

> **Note**
>
> The upper bound must be specified. The following are **NOT** accepted.
>
> ```
> (a)-[*3..]->(b)
> (a)-[*]->(b)
> ```

Assigning to path variables

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

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

Users can do this in the `MATCH` statement.

Last update: August 27, 2021

4.1.3 Comments

This topic will describe the comments in nGQL.

Legacy version compatibility

- In Nebula Graph 1.0, there are four comment styles: `#`, `--`, `//`, `/* */`.
- In Nebula Graph 2.0, `--` represents an edge pattern and cannot be used as comments.

Examples

```
nebula> # Do nothing in this line
nebula> RETURN 1+1;     # This comment continues to the end of this line.
nebula> RETURN 1+1;     // This comment continues to the end of this line.
nebula> RETURN 1 /* This is an in-line comment. */ + 1 == 2;
nebula> RETURN 11 +           \
/* Multi-line comment.       \
Use a backslash as a line break.   \
*/ 12;
```

In nGQL statement, the backslash `\` in a line indicates a line break.

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 `\` 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: July 13, 2021

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 my_space (vid_type=FIXED_STRING(30));
nebula> use MY_SPACE;
[ERROR (-8)]: 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

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: July 14, 2021

4.1.5 Keywords

Keywords have significance in nGQL. It can be classified into reserved keywords and non-reserved keywords.

Non-reserved keywords are permitted as identifiers without quoting. To use reserved keywords as identifiers, quote them with backticks such as `AND`.

> **Note**
>
> Keywords are case-insensitive.

```
nebula> CREATE TAG TAG(name string);
[ERROR (-7)]: SyntaxError: syntax error near `TAG'

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

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

- `TAG` is a reserved keyword. To use `TAG` as an identifier, you must quote it with backticks.
- `SPACE` is a non-reserved keyword. You can use it as an identifier without quoting it.

Reserved keywords

```
GO
AS
TO
OR
AND
XOR
USE
SET
FROM
WHERE
MATCH
INSERT
YIELD
RETURN
DESCRIBE
DESC
VERTEX
EDGE
EDGES
UPDATE
UPSERT
WHEN
DELETE
FIND
LOOKUP
ALTER
STEPS
STEP
OVER
UPTO
REVERSELY
INDEX
INDEXES
REBUILD
BOOL
INT8
INT16
INT32
INT64
INT
FLOAT
DOUBLE
STRING
FIXED_STRING
TIMESTAMP
DATE
TIME
DATETIME
TAG
TAGS
UNION
INTERSECT
MINUS
```

```
NO
OVERWRITE
SHOW
ADD
CREATE
DROP
REMOVE
IF
NOT
EXISTS
WITH
CHANGE
GRANT
REVOKE
ON
BY
IN
NOT_IN
DOWNLOAD
GET
OF
ORDER
INGEST
COMPACT
FLUSH
SUBMIT
ASC
ASCENDING
DESCENDING
DISTINCT
FETCH
PROP
BALANCE
STOP
LIMIT
OFFSET
IS
NULL
RECOVER
EXPLAIN
PROFILE
FORMAT
CASE
```

Non-reserved keywords

```
HOST
HOSTS
SPACE
SPACES
VALUE
VALUES
USER
USERS
PASSWORD
ROLE
ROLES
GOD
ADMIN
DBA
GUEST
GROUP
PARTITION_NUM
REPLICA_FACTOR
VID_TYPE
CHARSET
COLLATE
COLLATION
ATOMIC_EDGE
ALL
ANY
SINGLE
NONE
REDUCE
LEADER
UUID
DATA
SNAPSHOT
SNAPSHOTS
ACCOUNT
JOBS
JOB
PATH
BIDIRECT
STATS
STATUS
FORCE
PART
PARTS
DEFAULT
HDFS
```

```
CONFIGS
TTL_DURATION
TTL_COL
GRAPH
META
STORAGE
SHORTEST
NOLOOP
OUT
BOTH
SUBGRAPH
CONTAINS
NOT_CONTAINS
STARTS
STARTS_WITH
NOT_STARTS_WITH
ENDS
ENDS_WITH
NOT_ENDS_WITH
IS_NULL
IS_NOT_NULL
IS_EMPTY
IS_NOT_EMPTY
UNWIND
SKIP
OPTIONAL
THEN
ELSE
END
GROUPS
ZONE
ZONES
INTO
LISTENER
ELASTICSEARCH
FULLTEXT
AUTO
FUZZY
PREFIX
REGEXP
WILDCARD
TEXT
SEARCH
CLIENTS
SIGN
SERVICE
TEXT_SEARCH
RESET
PLAN
COMMENT
SESSIONS
SESSION
SAMPLE
QUERIES
QUERY
KILL
TOP
TRUE
FALSE
```

Last update: August 24, 2021

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.

> **Compatibility**
>
> 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 follow._dst AS id;
   ```

   Recommended:

   ```
   GO FROM "player100" \
   OVER follow REVERSELY \
   YIELD follow._dst AS id;
   ```

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

   Not recommended:

   ```
   GO FROM "player100" OVER follow REVERSELY YIELD follow._dst AS id | GO FROM $-.id \
   OVER serve WHERE $^.player.age > 20 YIELD $^.player.name AS FriendOf, $$.team.name AS Team;
   ```

   Recommended:

   ```
   GO FROM "player100" \
   OVER follow REVERSELY \
   YIELD follow._dst AS id | \
   GO FROM $-.id OVER serve \
   WHERE $^.player.age > 20 \
   YIELD $^.player.name AS FriendOf, $$.team.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.name STARTS WITH "Y" AND v2.age > 35 AND v2.age < v.age) OR (v2.name STARTS WITH "T" AND v2.age < 45 AND v2.age > v.age) \
   RETURN v2;
   ```

   Recommended:

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

> **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:

   ```
   go from "player100" over Follow
   ```

   Recommended:

   ```
   GO FROM "player100" OVER follow
   ```

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]->()<--(v3) \
   RETURN v, e, v2;
   ```

   Recommended:

   ```
   MATCH (v:player{name: "Tim Duncan", age: 42})-[e:follow]-> \
   ()-[e:serve]->()<--(v3) \
   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\"";
```

> **Note**
>
> When single or double quotes need to be nested in a string, use a backslash () to escape. For example:
>
> ```
> RETURN "\"Nebula Graph is amazing,\" the user says.";
> ```

Statement termination

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

   Not recommended:

   ```
   FETCH PROP ON player "player100"
   ```

   Recommended:

   ```
   FETCH PROP ON player "player100";
   ```

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 \
   YIELD follow._dst AS id; | \
   GO FROM $-.id \
   OVER serve \
   YIELD $$.team.name AS Team, $^.player.name AS Player;
   ```

   Supported:

   ```
   GO FROM "player100" \
   OVER follow \
   YIELD follow._dst AS id | \
   GO FROM $-.id \
   OVER serve \
   YIELD $$.team.name AS Team, $^.player.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 = GO FROM "player100" \
   OVER follow \
   YIELD follow._dst AS id \
   GO FROM $var.id \
   OVER serve \
   YIELD $$.team.name AS Team, $^.player.name AS Player;
   ```

Not supported:

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

Supported:

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

Last update: August 27, 2021

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.

Type	Declared keywords	Range
INT64	`INT64` or `INT`	-9,223,372,036,854,775,808 ~ 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.

Type	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`.

> **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, Nebula Graph 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`.

Last update: September 2, 2021

4.2.2 Boolean

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

nGQL 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: August 23, 2021

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 t1 (p1 FIXED_STRING(10));
```

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

```
nebula> CREATE TAG 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 Nebula Graph 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 Nebula Graph will return an error.

Escape characters

Line breaks are not allowed in a string. Escape characters are supported within strings, for example:

- `"\n\t\r\b\f"`
- `"\110ello world"`

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: September 2, 2021

4.2.4 Date and time types

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

While inserting time-type property values, except for `TIMESTAMP`, Nebula Graph transforms them to a UTC time according to the time zone specified with the `timezone_name` parameter in the configuration files. The time-type values returned by nGQL queries are all UTC time.

> **Note**
>
> To change the time zone, modify the `timezone_name` value in the configuration files of all Nebula Graph services.

- `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.

OpenCypher Compatibility

In nGQL:

- Year, month, day, hour, minute, and second are supported, while the millisecond is not supported.
- `localdatetime()` and `duration()` are not supported.
- Most string time formats are not supported. The only exception is `YYYY-MM-DDThh:mm:ss`.

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`.

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.msmsmsususus` format. The supported range is `00:00:00.000000` to `23:59:59.999999`.

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 `32767-12-31T23:59:59.999999`.

The properties of `datetime()` include `year`, `month`, `day`, `hour`, `minute`, and `second`.

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 parameters to get the timestamp of the current time zone and also accepts a string type parameter.

```
# Return the current time.
nebula> return timestamp();
+-------------+
| timestamp() |
+-------------+
| 1625469277  |
+-------------+

# Return the specified time.
nebula> return timestamp("2021-07-05T06:18:43.984000");
+-----------------------------------------+
| timestamp("2021-07-05T06:18:43.984000") |
+-----------------------------------------+
| 1625465923                              |
+-----------------------------------------+
```

- The underlying storage data type is **int64**.

Examples

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

```
nebula> CREATE TAG 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("2021-03-17T17:53:59"));
```

3. Return the month of the property `p1` on `test1`.

```
nebula> CREATE TAG INDEX date1_index ON date1(p1);
nebula> REBUILD TAG INDEX date1_index;
nebula> MATCH (v:date1) RETURN v.p1.month;
+------------+
| v.p1.month |
+------------+
| 3          |
+------------+
```

4. Create a tag named `school` with the property of `TIMESTAMP`.

```
nebula> CREATE TAG school(name string , found_time timestamp);
```

5. 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"));
```

6. 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. For example:

```
nebula> WITH time({hour: 12, minute: 31, second: 14}) AS d RETURN d;
+--------------+
| d            |
+--------------+
```

```
| 12:31:14.000 |
+--------------+

nebula> WITH date({year: 1984, month: 10, day: 11}) AS x RETURN x + 1;
+------------+
| x          |
+------------+
| 1984-10-12 |
+------------+
```

Last update: August 23, 2021

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`, `OR`, `XOR`, and `NOT`.

a	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 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                |
|           | ) 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 `player`. Specify the property `age` as `NOT NULL`. The default value is `18`.

```
nebula> CREATE TAG 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.

```
nebula> FETCH PROP ON player "Kobe"
+----------------------------------------+
| vertices_                              |
+----------------------------------------+
| ("Kobe" :player{age: 18, name: "Kobe"}) |
+----------------------------------------+
```

Last update: August 23, 2021

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.

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.

> **Note**
>
> - Return empty if the index is out of bounds, while return normally if the index is within the bound.
> - Return empty if `M ≥ 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 [1, 2, 3] AS List;
+-----------+
| List      |
+-----------+
| [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];
+------------------+
| range(1,5)[0..3] |
+------------------+
| [1, 2, 3]        |
+------------------+

# The following query returns the elements whose indexes are greater than 2 in the list [1,2,3,4,5].
```

```
nebula> RETURN range(1,5)[3..] AS a;
+--------+
| a      |
+--------+
| [4, 5] |
+--------+

# The following query returns the elements whose indexes are smaller than 3.
nebula> WITH [1, 2, 3, 4, 5] AS list \
        RETURN list[..3] AS r;
+-----------+
| r         |
+-----------+
| [1, 2, 3] |
+-----------+

# 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> RETURN [n IN range(1,5) WHERE n > 2 | n + 10] AS a;
+--------------+
| a            |
+--------------+
| [13, 14, 15] |
+--------------+

# The following query returns the elements from the first to the penultimate (inclusive) in the list [1, 2, 3].
nebula> YIELD [1, 2, 3][0..-1] AS a;
+--------+
| a      |
+--------+
| [1, 2] |
+--------+

# The following query returns the elements from the first (exclusive) to the third backward in the list [1, 2, 3, 4, 5].
nebula> YIELD [1, 2, 3, 4, 5][-3..-1] AS a;
+--------+
| a      |
+--------+
| [3, 4] |
+--------+

# The following query sets the variables and returns the elements whose indexes are 1 and 2.
nebula> $var = YIELD 1 AS f, 3 AS t; \
        YIELD [1, 2, 3][$var.f..$var.t] AS a;
+--------+
| a      |
+--------+
| [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 [1, 2, 3, 4, 5] [0..10] AS a;
+-----------------+
| a               |
+-----------------+
| [1, 2, 3, 4, 5] |
+-----------------+

nebula> RETURN [1, 2, 3] [-5..5] AS a;
+-----------+
| a         |
+-----------+
| [1, 2, 3] |
+-----------+

# The following query returns empty because there is a [0..0].
nebula> RETURN [1, 2, 3, 4, 5] [0..0] AS a;
+----+
| a  |
+----+
| [] |
+----+

# The following query returns empty because of M ≥ N.
nebula> RETURN [1, 2, 3, 4, 5] [3..1] AS a;
+----+
| a  |
+----+
| [] |
+----+

# When conduct a range query, if `M` or `N` is null, return `null`.
nebula> WITH [1,2,3] AS list \
        RETURN list[0..null] as a;
+----------+
| a        |
+----------+
| __NULL__ |
+----------+

# 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] |
+-----------------+

# The following query takes the elements in the list [1,2,3] as true and return.
nebula> RETURN [n IN range(1, 3) WHERE true | n] AS r;
+-----------+
| r         |
+-----------+
| [1, 2, 3] |
+-----------+

# The following query returns the length of the list [1,2,3].
nebula> RETURN size([1,2,3]);
+---------------+
| size([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> GO FROM "player100" OVER follow WHERE follow.degree NOT IN [x IN [92, 90] | x + $$.player.age] \
        YIELD follow._dst AS id, follow.degree AS degree;
+-------------+--------+
| id          | degree |
+-------------+--------+
| "player101" | 95     |
+-------------+--------+
| "player102" | 90     |
+-------------+--------+

# The following query takes the query result of the MATCH statement as the elements in a list. Then it calculates and returns them.
nebula> MATCH p = (n:player{name:"Tim Duncan"})-[:follow]->(m) \
        RETURN [n IN nodes(p) | n.age + 100] AS r;
+------------+
| r          |
+------------+
| [142, 136] |
+------------+
| [142, 133] |
+------------+
```

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.

  ```
  nebula> RETURN range(0,5)[-12];
  +-------------------+
  | range(0,5)[-(12)] |
  +-------------------+
  | OUT_OF_RANGE      |
  +-------------------+
  ```

- 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: September 2, 2021

4.2.7 Sets

The set is a composite data type.

OpenCypher compatibility

A set is not a data type in openCypher. The behavior of a set in nGQL is not determined yet.

Last update: July 14, 2021

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']`.

Literal maps

```
nebula> YIELD {key: 'Value', listKey: [{inner: 'Map1'}, {inner: 'Map2'}]}
+------------------------------------------------------------+
| {key:Value,listKey:[{inner:Map1},{inner:Map2}]}            |
+------------------------------------------------------------+
| {key: "Value", listKey: [{inner: "Map1"}, {inner: "Map2"}]} |
+------------------------------------------------------------+
```

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.

Last update: July 14, 2021

4.2.9 Type Conversion/Type coercions

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

Legacy version compatibility

- nGQL 1.0 adopts the `C`-style of type conversion (implicitly or explicitly): `(type_name)expression`. For example, the results of `YIELD (int)(TRUE)` is `1`. But it is error-prone to users who are not familiar with the C language.
- nGQL 2.0 chooses the openCypher way of type coercions.

Type coercions functions

Function	Description
toBoolean()	Converts a string value to a boolean value.
toFloat()	Converts an integer or string value to a floating point number.
toInteger()	Converts a floating point or string value to an integer value.
type()	Returns the string representation of the relationship type.

Examples

```
nebula> UNWIND [true, false, 'true', 'false', NULL] AS b RETURN toBoolean(b) AS b;
+----------+
| b        |
+----------+
| true     |
+----------+
| false    |
+----------+
| true     |
+----------+
| false    |
+----------+
| __NULL__ |
+----------+

nebula> RETURN toFloat(1), toFloat('1.3'), toFloat('1e3'), toFloat('not a number');
+------------+----------------+----------------+------------------------+
| toFloat(1) | toFloat("1.3") | toFloat("1e3") | toFloat("not a number") |
+------------+----------------+----------------+------------------------+
| 1.0        | 1.3            | 1000.0         | __NULL__               |
+------------+----------------+----------------+------------------------+

nebula> RETURN toInteger(1), toInteger('1'), toInteger('1e3'), toInteger('not a number');
+--------------+----------------+------------------+--------------------------+
| toInteger(1) | toInteger("1") | toInteger("1e3") | toInteger("not a number") |
+--------------+----------------+------------------+--------------------------+
| 1            | 1              | 1000             | __NULL__                 |
+--------------+----------------+------------------+--------------------------+

nebula> MATCH (a:player)-[e]-() RETURN type(e);
+----------+
| type(e)  |
+----------+
| "follow" |
+----------+
| "follow" |

nebula> MATCH (a:player {name: "Tim Duncan"}) WHERE toInteger(id(a)) == 100 RETURN a;
+-----------------------------------------------+
| a                                             |
+-----------------------------------------------+
| ("100" :player{age: 42, name: "Tim Duncan"}) |
+-----------------------------------------------+

nebula> MATCH (n:player) WITH n LIMIT toInteger(ceil(1.8)) RETURN count(*) AS count;
+-------+
| count |
+-------+
| 2     |
+-------+
```

Last update: July 14, 2021

4.3 Variables and composite queries

4.3.1 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 ... | GO ... | 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.

Undefined behavior

Do not put together `native nGQL` and `openCypher compatible sentences` in one composite statement because this behavior is undefined.

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.

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> GO FROM "player100" OVER follow YIELD follow._dst AS id | \
        GO FROM $-.id OVER serve YIELD $$.team.name AS Team, \
        $^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
```

Last update: August 27, 2021

4.3.2 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:

```
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

The user-defined variable in the preceding query is `v`.

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.

Example

```
nebula> $var = GO FROM "player100" OVER follow YIELD follow._dst AS id; \
        GO FROM $var.id OVER serve YIELD $$.team.name AS Team, \
        $^.player.name AS Player;
+---------+-------------+
| Team    | Player      |
+---------+-------------+
| Nuggets | Tony Parker |
+---------+-------------+
```

Last update: August 27, 2021

4.3.3 Property reference

You can refer to the properties of a vertex or an edge in `WHERE` and `YIELD` syntax.

> **Note**
>
> This function applies to native nGQL only.

Property reference for vertex

FOR SOURCE VERTEX

```
$^.<tag_name>.<prop_name>
```

Parameter	Description
`$^`	is used to get the property of the source vertex.
`tag_name`	is the tag name of the vertex.
`prop_name`	specifies the property name.

FOR DESTINATION VERTEX

```
$$.<tag_name>.<prop_name>
```

Parameter	Description
`$$`	is used to get the property of the destination vertex.
`tag_name`	is the tag name of the vertex.
`prop_name`	specifies the property name.

Property reference for edge

FOR USER-DEFINED EDGE PROPERTY

```
<edge_type>.<prop_name>
```

Parameter	Description
`edge_type`	is the edge type of the edge.
`prop_name`	specifies the property name of the edge type.

FOR BUILT-IN PROPERTIES

Apart from the user-defined edge property, there are four built-in properties in each edge:

Parameter	Description
`_src`	source vertex ID of the edge
`_dst`	destination vertex ID of the edge
`_type`	edge type
`_rank`	the rank value for the edge

Examples

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" | 33     |
+--------------+--------+
```

The following query returns the `degree` property of the edge type `follow`.

```
nebula> GO FROM "player100" OVER follow YIELD follow.degree;
+---------------+
| follow.degree |
+---------------+
| 95            |
+---------------+
| 90            |
+---------------+
```

The following query returns the source vertex, the destination vertex, the edge type, and the edge rank value of the edge type `follow`.

```
nebula> GO FROM "player100" OVER follow YIELD follow._src, follow._dst, follow._type, follow._rank;
+-------------+-------------+-------------+--------------+
| follow._src | follow._dst | follow._type | follow._rank |
+-------------+-------------+-------------+--------------+
| "player100" | "player101" | 136          | 0            |
+-------------+-------------+-------------+--------------+
| "player100" | "player102" | 136          | 0            |
+-------------+-------------+-------------+--------------+
```

Last update: August 27, 2021

4.4 Operators

4.4.1 Comparison operators

Nebula Graph supports the following comparison operators.

Name	Description
`=`	Assigns a value
`+`	Addition operator
`-`	Minus operator
`*`	Multiplication operator
`/`	Division operator
`==`	Equal operator
`!=`, `<>`	Not equal operator
`>`	Greater than operator
`>=`	Greater than or equal operator
`<`	Less than operator
`<=`	Less than or equal operator
`%`	Modulo operator
`-`	Changes the sign of the argument
`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

- The comparison operation of `NULL` is different from openCypher. The behavior may also change. `IS [NOT] NULL` is often used with `OPTIONAL MATCH` in openCypher. But `OPTIONAL MATCH` is not supported in nGQL.
- 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.

> **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 result;
+--------+
| result |
+--------+
| true   |
+--------+
```

>=

```
nebula> RETURN 2 >= "2", 2 >= 2;
+----------+--------+
| (2>="2") | (2>=2) |
+----------+--------+
| __NULL__ | true   |
+----------+--------+
```

<

```
nebula> YIELD 2.0 < 1.9;
+---------+
| (2<1.9) |
+---------+
| false   |
+---------+
```

<=

```
nebula> YIELD 0.11 <= 0.11;
+--------------+
| (0.11<=0.11) |
+--------------+
| true         |
+--------------+
```

!=

```
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(NULL), size(NULL), count(NULL), NULL IS NULL, NULL IS NOT NULL, sin(NULL), NULL + NULL, [1, NULL] IS NULL;
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+
| length(NULL) | size(NULL) | 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            |
+--------------+------------+-------------+--------------+------------------+-----------+-------------+------------------+

nebula> WITH {name: null} AS map \
        RETURN map.name IS NOT NULL;
+----------------------+
| map.name IS NOT NULL |
+----------------------+
| false                |
+----------------------+

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              |
+-------------------+-----------------------+-------------------+

nebula> MATCH (n:player) \
        RETURN n.age IS NULL, n.name IS NOT NULL, n.empty IS NULL;
+---------------+--------------------+-----------------+
| n.age IS NULL | n.name IS NOT NULL | n.empty IS NULL |
+---------------+--------------------+-----------------+
| false         | true               | true            |
+---------------+--------------------+-----------------+
| false         | true               | true            |
+---------------+--------------------+-----------------+
| false         | true               | true            |
+---------------+--------------------+-----------------+
...
```

IS [NOT] EMPTY

```
nebula> RETURN null IS EMPTY;
+---------------+
| NULL IS EMPTY |
+---------------+
| false         |
+---------------+

nebula> RETURN "a" IS NOT EMPTY;
+------------------+
| "a" IS NOT EMPTY |
+------------------+
| true             |
+------------------+

nebula> GO FROM "player100" OVER * WHERE $$.player.name IS NOT EMPTY YIELD follow._dst;
+-------------+
| follow._dst |
+-------------+
| "player125" |
+-------------+
| "player101" |
+-------------+
```

Last update: September 2, 2021

4.4.2 Boolean operators

Nebula Graph 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 `NULL`, refer to NULL.

Legacy version compatibility

- In Nebula Graph 2.0, non-zero numbers cannot be converted to boolean values.

Last update: July 19, 2021

4.4.3 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> GO FROM "player100" OVER follow \
        YIELD follow._dst AS dstid, $$.player.name AS Name | \
        GO FROM $-.dstid OVER follow;

+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
```

If there is no `YIELD` clause to define the output, the destination vertex ID is returned by default. If a YIELD clause is applied, the output is defined by the YIELD clause.

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

Performance tips

In Nebula Graph 2.5.0, 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. Pipe operators need to be serialized and deserialized, which is executed in a single thread.
3. 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 | B` with a single graphd process.

Last update: August 27, 2021

4.4.4 Reference operators

NGQL provides reference operators to represent a property in a `WHERE` or `YIELD` clause, or the output of the statement before the pipe operator in a composite query.

OpenCypher compatibility

Reference operators apply to native nGQL only.

Reference operator List

Reference operator	Description
`$^`	Refers to a source vertex property. For more information, see Property reference.
`$$`	Refers to a destination vertex property. For more information, see Property reference.
`$-`	Refers to the output of the statement before the pipe operator in a composite query. For more information, see Pipe.

Examples

```
# The following example returns the age of the source vertex and the destination vertex.
nebula> GO FROM "player100" OVER follow YIELD $^.player.age AS SrcAge, $$.player.age AS DestAge;
+--------+---------+
| SrcAge | DestAge |
+--------+---------+
| 42     | 36      |
+--------+---------+
| 42     | 41      |
+--------+---------+

# The following example returns the name and team of the players that player100 follows.
nebula> GO FROM "player100" OVER follow \
        YIELD follow._dst AS id | \
        GO FROM $-.id OVER serve \
        YIELD $^.player.name AS Player, $$.team.name AS Team;
+-----------------+-----------+
| Player          | Team      |
+-----------------+-----------+
| "Tony Parker"   | "Spurs"   |
+-----------------+-----------+
| "Tony Parker"   | "Hornets" |
+-----------------+-----------+
| "Manu Ginobili" | "Spurs"   |
+-----------------+-----------+
```

Last update: July 19, 2021

4.4.5 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, Nebula Graph will evaluate them from left to right unless parentheses explicitly specify another order.

OpenCypher compatibility

Set operators apply to native nGQL only.

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> GO FROM "player102" OVER follow \
        UNION \
        GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+

# The following statement returns the union of two query results with duplicated elements.
nebula> GO FROM "player102" OVER follow \
        UNION ALL \
        GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+

# UNION 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.
nebula> GO FROM "player102" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age \
        UNION /* DISTINCT */ \
        GO FROM "player100" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;
+-------------+--------+-----+
| id          | Degree | Age |
+-------------+--------+-----+
| "player101" | 75     | 36  |
+-------------+--------+-----+
| "player101" | 96     | 36  |
+-------------+--------+-----+
| "player102" | 90     | 33  |
+-------------+--------+-----+
```

INTERSECT

```
<left> INTERSECT <right>
```

- Operator `INTERSECT` returns the intersection of two sets A and B (denoted by A ⋂ 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

```
nebula> GO FROM "player102" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age \
        INTERSECT \
        GO FROM "player100" OVER follow \
        YIELD follow._dst AS id, follow.degree AS Degree, $$.player.age AS Age;
Empty set (time spent 2990/3511 us)
```

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

```
nebula> GO FROM "player100" OVER follow \
        MINUS \
        GO FROM "player102" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player102" |
+-------------+

nebula> GO FROM "player102" OVER follow \
        MINUS \
        GO FROM "player100" OVER follow;
Empty set (time spent 2243/3259 us)
```

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

```
nebula> GO FROM "player102" OVER follow \
        YIELD follow._dst AS play_dst  \
        UNION \
        GO FROM "team200" OVER serve REVERSELY \
        YIELD serve._dst AS play_dst \
        | GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;

+-------------+
| play_dst    |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
```

```
nebula> GO FROM "player102" OVER follow YIELD follow._dst AS play_dst  \
UNION \
GO FROM "team200" OVER serve REVERSELY YIELD serve._dst AS play_dst \
| GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;
```

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:

```
nebula> (GO FROM "player102" OVER follow \
        YIELD follow._dst AS play_dst  \
        UNION \
        GO FROM "team200" OVER serve REVERSELY \
        YIELD serve._dst AS play_dst) \
        | GO FROM $-.play_dst OVER follow YIELD follow._dst AS play_dst;
```

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.

Last update: August 27, 2021

4.4.6 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.

> **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 `CONTAINS` operator requires string types on both left and right sides.

```
nebula> MATCH (s:player)-[e:serve]->(t:team) WHERE id(s) == "player101" \
        AND t.name CONTAINS "ets" RETURN s.name, e.start_year, e.end_year, t.name;
+---------------+-------------+------------+-----------+
| s.name        | e.start_year | e.end_year | t.name    |
+---------------+-------------+------------+-----------+
| "Tony Parker" | 2018         | 2019       | "Hornets" |
+---------------+-------------+------------+-----------+

nebula> GO FROM "player101" OVER serve WHERE (STRING)serve.start_year CONTAINS "19" AND \
        $^.player.name CONTAINS "ny" \
        YIELD $^.player.name, serve.start_year, serve.end_year, $$.team.name;
+----------------+-----------------+----------------+--------------+
| $^.player.name | serve.start_year | serve.end_year | $$.team.name |
+----------------+-----------------+----------------+--------------+
| "Tony Parker"  | 1999             | 2018           | "Spurs"      |
+----------------+-----------------+----------------+--------------+

nebula> GO FROM "player101" OVER serve WHERE !($$.team.name CONTAINS "ets") \
        YIELD $^.player.name, serve.start_year, serve.end_year, $$.team.name;
+----------------+-----------------+----------------+--------------+
| $^.player.name | serve.start_year | serve.end_year | $$.team.name |
+----------------+-----------------+----------------+--------------+
| "Tony Parker"  | 1999             | 2018           | "Spurs"      |
+----------------+-----------------+----------------+--------------+
```

(NOT) IN

```
nebula> RETURN  1 IN [1,2,3], "Yao" NOT IN ["Yi", "Tim", "Kobe"], NULL IN ["Yi", "Tim", "Kobe"]
+---------------+-----------------------------------+------------------------------+
| (1 IN [1,2,3]) | ("Yao" NOT IN ["Yi","Tim","Kobe"]) | (NULL IN ["Yi","Tim","Kobe"]) |
+---------------+-----------------------------------+------------------------------+
```

```
| true           | true                              | __NULL__                     |
+----------------+-----------------------------------+------------------------------+
```

(NOT) STARTS WITH

```
nebula> RETURN 'apple' STARTS WITH 'app', 'apple' STARTS WITH 'a', 'apple' STARTS WITH toUpper('a')
+---------------------------+-------------------------+---------------------------------+
| ("apple" STARTS WITH "app") | ("apple" STARTS WITH "a") | ("apple" STARTS WITH toUpper("a")) |
+---------------------------+-------------------------+---------------------------------+
| true                        | true                    | false                           |
+---------------------------+-------------------------+---------------------------------+

nebula> RETURN 'apple' STARTS WITH 'b','apple' NOT STARTS WITH 'app'
+--------------------------+--------------------------------+
| ("apple" STARTS WITH "b") | ("apple" NOT STARTS WITH "app") |
+--------------------------+--------------------------------+
| false                    | false                          |
+--------------------------+--------------------------------+
```

(NOT) ENDS WITH

```
nebula> RETURN 'apple' ENDS WITH 'app', 'apple' ENDS WITH '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.).

Nebula Graph 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.name =~ 'Tony.*' RETURN v.name;
+---------------+
| v.name        |
+---------------+
| "Tony Parker" |
+---------------+
```

Last update: August 27, 2021

4.4.7 List operators

Nebula Graph supports the following list operators:

List 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;
+-----------------------+
| 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,1]) |
+--------------------+
| __NULL__           |
+--------------------+

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      |
+--------+

nebula> WITH ['Anne', 'John', 'Bill', 'Diane', 'Eve'] AS names RETURN names[1] AS result;
+--------+
| result |
+--------+
| "John" |
+--------+
```

Last update: August 27, 2021

4.4.8 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> RETURN 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 <= z` is equivalent to `x < y AND y <= z` in openCypher.

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

Last update: September 2, 2021

4.5 Functions and expressions

4.5.1 Built-in math functions

Function descriptions

Nebula Graph supports the following built-in 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.
collect()	Puts all the collected values into a list.
avg()	Returns the average value of the argument.
count()	Returns the number of records.
max()	Returns the maximum value.
min()	Returns the minimum value.
std()	Returns the population standard deviation.
sum()	Returns the sum value.
bit_and()	Bitwise AND.

Function	Description
bit_or()	Bitwise OR.
bit_xor()	Bitwise XOR.
int size()	Returns the number of elements in a list or a map.
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.
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`.

Note

If the argument is `NULL`, the output is undefined.

Example

```
# The following statement supports aggregate functions.
nebula>  GO FROM "Tim Duncan" OVER like YIELD like._dst AS dst, $$.player.age AS age \
         | GROUP BY $-.dst \
         YIELD \
         $-.dst AS dst, \
         toInteger((sum($-.age)/count($-.age)))+avg(distinct $-.age+1)+1 AS statistics;
+-----------------+------------+
| dst             | statistics |
+-----------------+------------+
| "Tony Parker"   | 74.0       |
+-----------------+------------+
| "Manu Ginobili" | 84.0       |
+-----------------+------------+
Got 2 rows (time spent 4739/5064 us)
```

Last update: August 27, 2021

4.5.2 Built-in string functions

Nebula Graph supports the following built-in string functions:

> **Note**
>
> Like SQL, the position index of nGQL starts from `1`, while in C language it starts from `0`.

Function	Description
int strcasecmp(string a, string b)	Compares string a and b without case sensitivity. When a = b, the return value is 0. When a > b, the return value is greater than 0. When a < b, the return value is less than 0.
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(string a)	Returns the length of the given string in bytes.
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 a. If string a is shorter than `count`, the system returns string a.
string right(string a, int count)	Returns a substring consisting of `count` characters from the right side of string a. If string a is shorter than `count`, the system returns string a.
string lpad(string a, int size, string letters)	Left-pads string a with string `letters` and returns a substring with the length of `size`.
string rpad(string a, int size, string letters)	Right-pads string a with string `letters` and returns a substring with the length of `size`.
string substr(string a, int pos, int count)	Returns a substring extracting `count` characters starting from the specified position `pos` of string a.
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.
string toString()	Takes in any data type and converts it into a string.
int hash()	Takes in any data type and encodes it into a hash value.

> **Note**
>
> If the argument is `NULL`, the return is undefined.

Explanations for the return of `substr()` and `substring()`

- The position index starts from `0`.
- If `pos` is 0, the whole string is returned.
- 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.

> **OpenCypher compatibility**
>
> - In openCypher, if `a` is `null`, `null` is returned.
> - In openCypher, if `pos` is 0, the returned substring starts from the first character, and extend to `count` characters.
> - In openCypher, if either `pos` or `count` is `null` or a negative integer, an issue is raised.

Last update: July 19, 2021

4.5.3 Built-in date and time functions

Nebula Graph supports the following built-in date and time functions:

Function	Description
int now()	Returns the current date and time of the system time zone.
timestamp timestamp()	Returns the current date and time of the system time zone.
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.

The `date()`, `time()`, and `datetime()` functions accept three kind of parameters, namely empty, string, and map. The `timestamp()` function accepts two kind of parameters, namely empty and string.

OpenCypher compatibility

- Time in openCypher is measured in milliseconds.
- Time in nGQL is measured in seconds. The milliseconds are displayed in `000`.

Examples

```
> RETURN now(), timestamp(), date(), time(), datetime();
+------------+-------------+------------+-----------------+----------------------------+
| now()      | timestamp() | date()     | time()          | datetime()                 |
+------------+-------------+------------+-----------------+----------------------------+
| 1625470028 | 1625470028  | 2021-07-05 | 07:27:07.944000 | 2021-07-05T07:27:07.944000 |
+------------+-------------+------------+-----------------+----------------------------+
```

Last update: August 27, 2021

4.5.4 Schema functions

> **Note**
>
> The functions introduced in this topic applies to compatible statements in openCypher only.

Nebula Graph supports the following built-in schema functions:

Function	Description
id(vertex)	Returns the id of a vertex. The data type of the result is the same as the vertex ID.
list tags(vertex)	Returns the tags of a vertex.
list labels(vertex)	Returns the tags of a vertex.
map properties(vertex_or_edge)	Takes in a vertex or an edge and returns its properties.
string type(edge)	Returns the edge type of an edge.
vertex startNode(path)	Takes in an edge or a path and returns its source vertex ID.
string endNode(path)	Takes in an edge or a path and returns its destination vertex ID.
int rank(edge)	Returns the rank value of an edge.

Examples

```
nebula> MATCH (a:player) WHERE id(a) == "player100" \
        RETURN tags(a), labels(a), properties(a);
+------------+------------+------------------------------+
| tags(a)    | labels(a)  | properties(a)                |
+------------+------------+------------------------------+
| ["player"] | ["player"] | {age: 42, name: "Tim Duncan"} |
+------------+------------+------------------------------+

nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) \
        RETURN type(r), rank(r);
+---------+---------+
| type(r) | rank(r) |
+---------+---------+
| "serve" | 0       |
+---------+---------+

nebula> MATCH p = (a :player {name : "Tim Duncan"})-[r:serve]-(t) \
        RETURN startNode(p), endNode(p);
+-------------------------------------------------+---------------------------------+
| startNode(p)                                    | endNode(p)                      |
+-------------------------------------------------+---------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | ("team204" :team{name: "Spurs"}) |
+-------------------------------------------------+---------------------------------+
```

Last update: August 23, 2021

4.5.5 CASE expressions

The `CASE` expression uses conditions to filter the result of an nGQL query statement. It is usually used in the `YIELD` and `RETURN` clauses. 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 `CASE` expression with an `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> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Name, \
        CASE $$.player.age > 35 \
        WHEN true THEN "Yes" \
        WHEN false THEN "No" \
        ELSE "Nah" \
        END \
        AS Age_above_35;
+---------------------+--------------+
| Name                | Age_above_35 |
+---------------------+--------------+
| "Tony Parker"       | "Yes"        |
+---------------------+--------------+
| "LaMarcus Aldridge" | "No"         |
+---------------------+--------------+
```

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

```
nebula> YIELD \
        CASE WHEN 4 > 5 THEN 0 \
        WHEN 3+4==7 THEN 1 \
        ELSE 2 \
        END \
        AS result;
+--------+
| result |
+--------+
| 1      |
+--------+
```

```
nebula> MATCH (v:player) WHERE v.age > 30 \
        RETURN v.name AS Name,  \
        CASE \
        WHEN v.name STARTS WITH "T" THEN "Yes" \
        ELSE "No" \
        END \
        AS Starts_with_T;
+---------------------+---------------+
| Name                | Starts_with_T |
+---------------------+---------------+
| "Tim"               | "Yes"         |
+---------------------+---------------+
| "LaMarcus Aldridge" | "No"          |
+---------------------+---------------+
| "Tony Parker"       | "Yes"         |
+---------------------+---------------+
```

Differences between the simple form and the generic form

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 $$.player.name AS Name, $$.player.age AS Age, \
        CASE $$.player.age \
        WHEN $$.player.age > 35 THEN "Yes" \
        ELSE "No" \
        END \
        AS Age_above_35;
+---------------------+-----+--------------+
| Name                | Age | Age_above_35 |
+---------------------+-----+--------------+
| "Tony Parker"       | 36  | "No"         |
+---------------------+-----+--------------+
| "LaMarcus Aldridge" | 33  | "No"         |
+---------------------+-----+--------------+
```

The preceding `GO` 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.

Last update: July 19, 2021

4.5.6 List functions

Nebula Graph supports the following 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 `[start,end]`. `step` is 1 by default.
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.
coalesce(list)	Returns the first not null value in a list.
reduce()	See reduce() function.

Note

If the argument is `NULL`, the output is undefined.

Examples

```
nebula> WITH [NULL, 4923, 'abc', 521, 487] AS ids \
        RETURN reverse(ids), tail(ids), head(ids), last(ids), coalesce(ids);
+----------------------------------+-------------------------+-----------+-----------+---------------+
| reverse(ids)                     | tail(ids)               | head(ids) | last(ids) | coalesce(ids) |
+----------------------------------+-------------------------+-----------+-----------+---------------+
| [487, 521, "abc", 4923, __NULL__] | [4923, "abc", 521, 487] | __NULL__  | 487       | 4923          |
+----------------------------------+-------------------------+-----------+-----------+---------------+

nebula> MATCH (a:player)-[r]->() \
        WHERE id(a) == "player100" \
        RETURN labels(a),  keys(r);
+------------+----------------------------+
| labels(a)  | keys(r)                    |
+------------+----------------------------+
| ["player"] | ["degree"]                 |
+------------+----------------------------+
| ["player"] | ["degree"]                 |
+------------+----------------------------+
| ["player"] | ["end_year", "start_year"] |
+------------+----------------------------+

nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) \
        WHERE a.name == "Tim Duncan" AND c.name == "Spurs" \
        RETURN nodes(p);
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| nodes(p)                                                                                                                                                    |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker"}), ("team204" :team{name: "Spurs"})]                  |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player125" :player{age: 41, name: "Manu Ginobili"}), ("team204" :team{name: "Spurs"})]                |
+-------------------------------------------------------------------------------------------------------------------------------------------------------------+

nebula> MATCH p = (a:player)-[]->(b)-[]->(c:team) WHERE a.name == "Tim Duncan" AND c.name == "Spurs" RETURN relationships(p);
+-----------------------------------------------------------------------------------------------------------------------------+
| relationships(p)                                                                                                            |
+-----------------------------------------------------------------------------------------------------------------------------+
| [[:follow "player100"->"player101" @0 {degree: 95}], [:serve "player101"->"team204" @0 {end_year: 2018, start_year: 1999}]] |
+-----------------------------------------------------------------------------------------------------------------------------+
```

```
| [[:follow "player100"->"player125" @0 {degree: 95}], [:serve "player125"->"team204" @0 {end_year: 2018, start_year: 2002}]] |
+-------------------------------------------------------------------------------------------------------------------------+
```

Last update: August 27, 2021

4.5.7 count() function

The `count()` function counts the number of the specified values or rows.

- (Native nGQL) You can use `count()` and `GROUP BY` together to group and count the number of specific values. Use `YIELD` to return.
- (OpenCypher style) You can use `count()` and `RETURN`. `GROUP BY` is not necessary.

Syntax

```
count({expr | *})
```

- count(*) returns the number of rows (including NULL).
- count(expr) return the number of non-NULL values that meet the expression.
- `count()` and `size()` are different.

EXAMPLES

```
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                |
+----------+----------+------------------+
```

```
# The statement in the following example searches for the people whom `player101` follows and people who follow `player101`, i.e. a bidirectional query.
nebula> GO FROM "player101" OVER follow BIDIRECT \
        YIELD $$.player.name AS Name \
        | GROUP BY $-.Name YIELD $-.Name, count(*);
+---------------------+----------+
| $-.Name             | count(*) |
+---------------------+----------+
| "Dejounte Murray"   | 1        |
+---------------------+----------+
| "LaMarcus Aldridge" | 2        |
+---------------------+----------+
| "Tim Duncan"        | 2        |
+---------------------+----------+
| "Marco Belinelli"   | 1        |
+---------------------+----------+
| "Manu Ginobili"     | 1        |
+---------------------+----------+
| "Boris Diaw"        | 1        |
+---------------------+----------+
```

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 \
        YIELD $-.playerage as age, count(*) AS number \
        | ORDER BY $-.number DESC, $-.age DESC;
+-----+--------+
| 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.age as age, count(*) as number \
        ORDER BY number DESC, age DESC;
+-----+--------+
| age | number |
+-----+--------+
| 34  | 4      |
+-----+--------+
| 33  | 4      |
+-----+--------+
| 30  | 4      |
+-----+--------+
| 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"}) -- (v2) \
        RETURN count(DISTINCT v2);
+--------------------+
| count(distinct v2) |
+--------------------+
| 11                 |
+--------------------+

# 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) | count(distinct fof) |
+------------+---------------------+
| 4          | 3                   |
+------------+---------------------+
```

Last update: August 27, 2021

4.5.8 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.

The aggregate function `collect()` works like `GROUP BY` in SQL.

Examples

```
nebula> UNWIND [1, 2, 1] AS a \
        RETURN a;
+---+
| a |
+---+
| 1 |
+---+
| 2 |
+---+
| 1 |
+---+

nebula> UNWIND [1, 2, 1] AS a \
        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.œ
nebula> UNWIND ["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.age);
+-------------------------------------------------------------+
| collect(n.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.age AS age, collect(n.name);
+-----+-------------------------------------------------------------------------+
| age | collect(n.name)                                                         |
+-----+-------------------------------------------------------------------------+
| 24  | ["Giannis Antetokounmpo"]                                               |
+-----+-------------------------------------------------------------------------+
| 20  | ["Luka Doncic"]                                                         |
+-----+-------------------------------------------------------------------------+
| 25  | ["Joel Embiid", "Kyle Anderson"]                                        |
+-----+-------------------------------------------------------------------------+
...
```

Last update: July 19, 2021

4.5.9 reduce() function

This topic will describe the `reduce` function.

OpenCypher Compatibility

In openCypher, the `reduce()` function is not defined. nGQL will implement the `reduce()` function in the Cypher way.

Syntax

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. This function iterates each element `e` in the given list, runs the expression on `e`, 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.

```
reduce(<accumulator> = <initial>, <variable> IN <list> | <expression>)
```

Parameter	Description
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`.

Note

The type of the value returned depends on the parameters provided, along with the semantics of the expression.

Examples

```
nebula> RETURN reduce(totalNum = 10, n IN range(1, 3) | totalNum + n) AS r;
+----+
| r  |
+----+
| 16 |
+----+

nebula> RETURN reduce(totalNum = -4 * 5, n IN [1, 2] | totalNum + n * 2) AS r;
+-----+
| r   |
+-----+
| -14 |
+-----+

nebula> MATCH p = (n:player{name:"LeBron James"})<-[:follow]-(m) \
        RETURN  nodes(p)[0].age AS src1,  nodes(p)[1].age AS dst2,  \
        reduce(totalAge = 100, n IN nodes(p) | totalAge + n.age) AS sum;
+------+------+-----+
| src1 | dst2 | sum |
+------+------+-----+
| 34   | 31   | 165 |
+------+------+-----+
| 34   | 29   | 163 |
+------+------+-----+
| 34   | 33   | 167 |
+------+------+-----+
| 34   | 26   | 160 |
+------+------+-----+
| 34   | 34   | 168 |
+------+------+-----+
| 34   | 37   | 171 |
+------+------+-----+

nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        |  GO FROM $-.VertexID over follow \
        WHERE follow.degree != reduce(totalNum = 5, n IN range(1, 3) | $$.player.age + totalNum + n) \
```

```
        YIELD $$.player.name AS id, $$.player.age AS age, follow.degree AS degree;
+---------------------+-----+--------+
| id                  | age | degree |
+---------------------+-----+--------+
| "Tim Duncan"        | 42  | 95     |
+---------------------+-----+--------+
| "LaMarcus Aldridge" | 33  | 90     |
+---------------------+-----+--------+
| "Manu Ginobili"     | 41  | 95     |
+---------------------+-----+--------+
```

Last update: July 19, 2021

4.5.10 hash function

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.

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)
```

Legacy version compatibility

In nGQL 1.0, when nGQL does not support string VIDs, a common practice is to hash the strings first and then use the values as VIDs. But in nGQL 2.0, both string VIDs and integer VIDs are supported, so there is no need to use `hash()` to set VIDs.

Hash a number

```
nebula> YIELD hash(-123);
+--------------+
| hash(-(123)) |
+--------------+
| -123         |
+--------------+
```

Hash a string

```
nebula> YIELD hash("to_be_hashed");
+----------------------+
| hash(to_be_hashed)   |
+----------------------+
| -1098333533029391540 |
+----------------------+
```

Hash a list

```
nebula> YIELD hash([1,2,3]);
+----------------+
| hash([1,2,3])  |
+----------------+
| 11093822460243 |
+----------------+
```

Hash a boolean

```
nebula> YIELD hash(true);
+------------+
| hash(true) |
+------------+
| 1          |
+------------+

nebula> YIELD hash(false);
+-------------+
| hash(false) |
+-------------+
| 0           |
+-------------+
```

Hash NULL

```
nebula> YIELD hash(NULL);
+------------+
| hash(NULL) |
+------------+
| -1         |
+------------+
```

Hash an expression

```
nebula> YIELD hash(toLower("HELLO NEBULA"));
+------------------------------+
| hash(toLower("HELLO NEBULA")) |
+------------------------------+
| -8481157362655072082         |
+------------------------------+
```

Last update: August 27, 2021

4.5.11 concat function

The `concat()` and `concat_ws()` functions return strings concatenated by one or more strings.

concat() function

The `concat()` function requires at least two or more strings. All the parameters are concatenated into one string.

- If there is only one string, the string itself is returned.
- If any one of the strings is `NULL`, `NULL` is returned.

SYNTAX

```
concat(string1,string2,...)
```

EXAMPLES

```
//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__ |
+----------+

nebula> GO FROM "player100" over follow \
        YIELD concat(follow._src, $^.player.age, $$.player.name, follow.degree) AS A;
+------------------------------+
| A                            |
+------------------------------+
| "player10042Tony Parker95"   |
+------------------------------+
| "player10042Manu Ginobili95" |
+------------------------------+
```

concat_ws() function

The `concat_ws()` function connects two or more strings with a predefined separator.

- If the separator is `NULL`, the `concat_ws()` function returns `NULL`.
- If the separator is not `NULL` and there is only one string, the string itself is returned.
- If the separator is not `NULL` and there is a `NULL` in the strings, `NULL` is ignored during the concatenation.

SYNTAX

```
concat_ws(separator,string1,string2,... )
```

EXAMPLES

```
//This example concatenates a, b, and c with the separator +.
nebula> RETURN concat_ws("+","a","b","c") AS r;
+---------+
| r       |
+---------+
| "a+b+c" |
+---------+

//In this example, the separator is NULL.
neubla> RETURN concat_ws(NULL,"a","b","c") AS r;
+----------+
| r        |
+----------+
| __NULL__ |
+----------+
```

```
//In this example, the separator is + and there is a NULL in the strings.
nebula> RETURN concat_ws("+","a",NULL,"b","c") AS r;
+---------+
| r       |
+---------+
| "a+b+c" |
+---------+

//In this example, the separator is + and there is only one string.
nebula> RETURN concat_ws("+","a") AS r;
+-----+
| r   |
+-----+
| "a" |
+-----+

nebula> GO FROM "player100" over follow \
        YIELD concat_ws(" ",follow._src, $^.player.age, $$.player.name, follow.degree) AS A;
+---------------------------------+
| A                               |
+---------------------------------+
| "player100 42 Tony Parker 95"   |
+---------------------------------+
| "player100 42 Manu Ginobili 95" |
+---------------------------------+
```

Last update: September 13, 2021

4.5.12 Predicate functions

Predicate functions return `true` or `false`. They are most commonly used in `WHERE` clauses.

Nebula Graph 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.

Compatibility

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] \
        WHERE n > 2) AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> RETURN single(n IN range(1, 5) \
        WHERE n == 3) AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> RETURN none(n IN range(1, 3) \
        WHERE n == 0) AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> WITH [1, 2, 3, 4, 5, NULL] AS a \
        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) \
        RETURN nodes(p)[0].name AS n1, nodes(p)[1].name AS n2, \
        all(n IN nodes(p) WHERE n.name NOT STARTS WITH "D") AS b;
+----------------+------------------+-------+
```

```
| n1             | n2                | b     |
+----------------+-------------------+-------+
| "LeBron James" | "Danny Green"     | false |
+----------------+-------------------+-------+
| "LeBron James" | "Dejounte Murray" | false |
+----------------+-------------------+-------+
| "LeBron James" | "Chris Paul"      | true  |
+----------------+-------------------+-------+
| "LeBron James" | "Kyrie Irving"    | true  |
+----------------+-------------------+-------+
| "LeBron James" | "Carmelo Anthony" | true  |
+----------------+-------------------+-------+
| "LeBron James" | "Dwyane Wade"     | false |
+----------------+-------------------+-------+

nebula> MATCH p = (n:player{name:"LeBron James"})-[:follow]->(m) \
        RETURN single(n IN nodes(p) WHERE n.age > 40) AS b;
+------+
| b    |
+------+
| true |
+------+

nebula> MATCH (n:player) \
        RETURN exists(n.id), n IS NOT NULL;
+--------------+---------------+
| exists(n.id) | n IS NOT NULL |
+--------------+---------------+
| false        | true          |
+--------------+---------------+
...

nebula> MATCH (n:player) \
        WHERE exists(n['name']) RETURN n;
+-------------------------------------------------------------------------------------------------------------+
| n                                                                                                           |
+-------------------------------------------------------------------------------------------------------------+
| ("Grant Hill" :player{age: 46, name: "Grant Hill"})                                                         |
+-------------------------------------------------------------------------------------------------------------+
| ("Marc Gasol" :player{age: 34, name: "Marc Gasol"})                                                         |
+-------------------------------------------------------------------------------------------------------------+
...
```

Last update: July 19, 2021

4.5.13 User-defined functions

OpenCypher compatibility

User-defined functions (UDF) and storage processes are not yet supported nor designed in Nebula Graph 2.5.0.

Last update: July 19, 2021

4.6 General queries statements

4.6.1 MATCH

The `MATCH` statement supports searching based on pattern matching.

A `MATCH` statement defines a search pattern and uses it to match data stored in Nebula Graph and to retrieve them in the form defined in the `RETURN` clause.

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 `GO` or `LOOKUP`. But generally, it can be summarized as follows.

```
MATCH <pattern> [<WHERE clause>] RETURN <output>
```

The workflow of MATCH

1. The `MATCH` statement uses a native index to locate a source vertex or an edge. The source vertex or the edge can be in any position in the pattern. In other words, in a valid `MATCH` statement, **there must be an indexed property, a tag, or an edge type. Or the VID of a specific vertex must be specified with the id() function in the `WHERE` clause.** For how to create an index, see create native index.
2. The `MATCH` statement searches through the pattern to match edges or vertices.

   > **Note**
   >
   > The path type of the `MATCH` statement is `trail`. That is, only vertices can be repeatedly visited in the graph traversal. Edges cannot be repeatedly visited. For details, see path.

3. The `MATCH` statement retrieves data according to the `RETURN` clause.

OpenCypher compatibility

- For now, nGQL does not support traversing all vertices and edges with `MATCH`, such as `MATCH (v) RETURN v`. However, after the index of a certain tag is created, all corresponding vertices can be traversed, such as `MATCH (v:T1) RETURN v`.
- Graph pattern is not supported in the `WHERE` clause.

Using patterns in MATCH statements

PREREQUISITES

Make sure there is at least one index in the `MATCH` statement, or there is a specified VID. If you want to create an index, but there are already related vertices, edges, or properties, you must rebuild indexes after creating the index to make it valid.

> **Caution**
>
> Correct use of indexes can speed up queries, but indexes can dramatically reduce the write performance. The performance reduction can be as much as 90% or even more. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.

```
# The following example creates an index on both the name property of the tag player and the edge type follow.
nebula> CREATE TAG INDEX name ON player(name(20));
```

```
nebula> CREATE EDGE INDEX follow_index on follow();

# The following example rebuilds the index.
nebula> REBUILD TAG INDEX name;
+------------+
| New Job Id |
+------------+
| 121        |
+------------+

nebula> REBUILD EDGE INDEX follow_index
+------------+
| New Job Id |
+------------+
| 122        |
+------------+


# The following example makes sure the index is rebuilt successfully.
nebula> SHOW JOB 121;
+---------------+---------------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time              | Stop Time               |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 121           | "REBUILD_TAG_INDEX" | "FINISHED" | 2021-05-27T02:18:02.000 | 2021-05-27T02:18:02.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 0             | "storaged1"         | "FINISHED" | 2021-05-27T02:18:02.000 | 2021-05-27T02:18:02.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 1             | "storaged0"         | "FINISHED" | 2021-05-27T02:18:02.000 | 2021-05-27T02:18:02.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 2             | "storaged2"         | "FINISHED" | 2021-05-27T02:18:02.000 | 2021-05-27T02:18:02.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+

nebula> SHOW JOB 122;
+---------------+---------------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time              | Stop Time               |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 122           | "REBUILD_EDGE_INDEX" | "FINISHED" | 2021-05-27T02:18:11.000 | 2021-05-27T02:18:11.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 0             | "storaged1"         | "FINISHED" | 2021-05-27T02:18:11.000 | 2021-05-27T02:18:21.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 1             | "storaged0"         | "FINISHED" | 2021-05-27T02:18:11.000 | 2021-05-27T02:18:21.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
| 2             | "storaged2"         | "FINISHED" | 2021-05-27T02:18:11.000 | 2021-05-27T02:18:21.000 |
+---------------+---------------------+------------+-------------------------+-------------------------+
```

MATCH VERTICES

You can use a user-defined variable in a pair of parentheses to represent a vertex in a pattern. For example: `(v)`.

MATCH TAGS

> **Note**
>
> The prerequisite for matching a tag is that the tag itself has an index or a certain property of the tag has an index. Otherwise, you cannot execute the `MATCH` statement based on the tag.

You can specify a tag with `:<tag_name>` after the vertex in a pattern.

```
nebula> MATCH (v:player) RETURN v;
+------------------------------------------------------------+
| v                                                          |
+------------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"})  |
+------------------------------------------------------------+
| ("player106" :player{age: 25, name: "Kyle Anderson"})      |
+------------------------------------------------------------+
| ("player115" :player{age: 40, name: "Kobe Bryant"})        |
+------------------------------------------------------------+
...
```

MATCH VERTEX PROPERTIES

> **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.

```
# The following example uses the name property to match a vertex.
nebula> MATCH (v:player{name:"Tim Duncan"}) RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

The `WHERE` clause can do the same thing:

```
nebula> MATCH (v:player) WHERE v.name == "Tim Duncan" RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+
```

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).

MATCH VIDS

You can use the VID to match a vertex. The `id()` function can retrieve the VID of a vertex.

```
nebula> MATCH (v) WHERE id(v) == 'player101' RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------+
```

To match multiple VIDs, use `WHERE id(v) IN [vid_list]`.

```
nebula> MATCH (v:player { name: 'Tim Duncan' })--(v2) \
        WHERE id(v2) IN ["player101", "player102"] RETURN v2;
+-------------------------------------------------------+
| v2                                                    |
+-------------------------------------------------------+
| ("player101" :player{name: "Tony Parker", age: 36})   |
+-------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-------------------------------------------------------+
| ("player101" :player{name: "Tony Parker", age: 36})   |
+-------------------------------------------------------+
```

MATCH CONNECTED VERTICES

You can use the `--` symbol to represent edges of both directions and match vertices connected by these edges.

Legacy 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.

```
nebula> MATCH (v:player{name:"Tim Duncan"})--(v2) \
        RETURN v2.name AS Name;
+--------------------+
| Name               |
+--------------------+
| "Tony Parker"      |
+--------------------+
| "LaMarcus Aldridge" |
+--------------------+
| "Marco Belinelli"  |
+--------------------+
| "Danny Green"      |
+--------------------+
| "Aron Baynes"      |
+--------------------+
...
```

You can add a `>` or `<` to the `--` symbol to specify the direction of an edge.

In the following example, `-->` 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) \
        RETURN v2.name AS Name;
+-----------------+
| Name            |
+-----------------+
| "Spurs"         |
+-----------------+
| "Tony Parker"   |
+-----------------+
| "Manu Ginobili" |
+-----------------+
```

To extend the pattern, you can add more vertices and edges.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->(v2)<--(v3) \
        RETURN v3.name AS Name;
+---------------------+
| Name                |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "Tiago Splitter"    |
+---------------------+
| "Dejounte Murray"   |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
...
```

If you do not need to refer to a vertex, you can omit the variable representing it in the parentheses.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-->()<--(v3) \
        RETURN v3.name AS Name;
+---------------------+
| Name                |
+---------------------+
| "Tony Parker"       |
+---------------------+
| "LaMarcus Aldridge" |
+---------------------+
| "Rudy Gay"          |
+---------------------+
| "Danny Green"       |
+---------------------+
| "Kyle Anderson"     |
+---------------------+
...
```

MATCH PATHS

Connected vertices and edges form a path. You can use a user-defined variable to name a path as follows.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-->(v2) \
        RETURN p;
+-----------------------------------------+
| p                                       |
+-----------------------------------------+
| <("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"})> |
+-----------------------------------------+
```

> **OpenCypher compatibility**
>
> In nGQL, the `@` symbol represents the rank of an edge, but openCypher has no such concept.

MATCH EDGES

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]-`.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:follow "player101"->"player100" @0 {degree: 95}]                     |
+------------------------------------------------------------------------+
| [:follow "player102"->"player100" @0 {degree: 75}]                     |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]  |
+------------------------------------------------------------------------+
...
```

MATCH EDGE TYPES

Just like vertices, you can specify edge types with `:<edge_type>` in a pattern. For example: `-[e:follow]-`.

```
nebula> MATCH ()-[e:follow]-() \
        RETURN e;
+--------------------------------------------------+
| e                                                |
+--------------------------------------------------+
| [:follow "player113"->"player119" @0 {degree: 99}]  |
+--------------------------------------------------+
| [:follow "player130"->"player149" @0 {degree: 80}]  |
+--------------------------------------------------+
| [:follow "player149"->"player130" @0 {degree: 80}]  |
+--------------------------------------------------+
| [:follow "player136"->"player117" @0 {degree: 90}]  |
+--------------------------------------------------+
| [:follow "player142"->"player117" @0 {degree: 90}]  |
+--------------------------------------------------+
...
```

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"})-[e:follow{degree:95}]->(v2) \
        RETURN e;
+-------------------------------------------------------+
| e                                                     |
+-------------------------------------------------------+
| [:follow "player100"->"player101" @0 {degree: 95}]    |
+-------------------------------------------------------+
| [:follow "player100"->"player125" @0 {degree: 95}]    |
+-------------------------------------------------------+
```

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                                                        |
+----------------------------------+-----------------------------------------------------------+
```

```
| ("player204" :team{name: "Spurs"}) | ("player101" :player{name: "Tony Parker", age: 36})       |
+------------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+------------------------------------+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"}) | ("player103" :player{age: 32, name: "Rudy Gay"})          |
+------------------------------------+-----------------------------------------------------------+
...
```

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.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*2]->(v2) \
        RETURN DISTINCT v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42})    |
+-------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-------------------------------------------------------+
| ("player125" :player{name: "Manu Ginobili", age: 41})     |
+-------------------------------------------------------+
```

If `hop` is 0, the pattern will match the source vertex of the path.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) -[*0]-> (v2) \
        RETURN v2;
+-------------------------------------------------+
| v2                                              |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
```

MATCH VARIABLE-LENGTH PATHS

You can use the `:<edge_type>*[minHop]..<maxHop>` pattern to match variable-length paths.

Parameter	Description
`minHop`	Optional. It represents the minimum length of the path. `minHop` must be a non-negative integer. The default value is 1.
`maxHop`	Required. It represents the maximum length of the path. `maxHop` must be a non-negative integer. It has no default value.

> **OpenCypher compatibility**
>
> In openCypher, `maxHop` is optional and defaults to infinity. When no bounds are given, `..` can be omitted. However, in nGQL, `maxHop` is required. And `..` cannot be omitted.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2) \
        RETURN v2 AS Friends;
+-------------------------------------------------------+
| Friends                                               |
+-------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
```

You can use the `DISTINCT` keyword to aggregate duplicate results.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*1..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-------------------------------------------------------+-----------+
| Friends                                               | count(v2) |
+-------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"}) | 3         |
+-------------------------------------------------------+-----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
+-------------------------------------------------------+-----------+
| ("player100" :player{age: 42, name: "Tim Duncan"})    | 4         |
```

```
+-----------------------------------------------------------+-----------+
| ("player101" :player{age: 36, name: "Tony Parker"})       | 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.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow*0..3]->(v2:player) \
        RETURN DISTINCT v2 AS Friends, count(v2);
+-----------------------------------------------------------+-----------+
| Friends                                                   | count(v2) |
+-----------------------------------------------------------+-----------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})     | 3         |
+-----------------------------------------------------------+-----------+
| ("player101" :player{age: 36, name: "Tony Parker"})       | 3         |
+-----------------------------------------------------------+-----------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) | 1         |
+-----------------------------------------------------------+-----------+
| ("player100" :player{age: 42, name: "Tim Duncan"})        | 5         |
+-----------------------------------------------------------+-----------+
```

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.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e:follow|serve*2]->(v2) \
        RETURN DISTINCT v2;
+-----------------------------------------------------------+
| v2                                                        |
+-----------------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42})        |
+-----------------------------------------------------------+
| ("player102" :player{name: "LaMarcus Aldridge", age: 33}) |
+-----------------------------------------------------------+
| ("player125" :player{name: "Manu Ginobili", age: 41})     |
+-----------------------------------------------------------+
| ("player204" :team{name: "Spurs"})                        |
+-----------------------------------------------------------+
| ("player215" :team{name: "Hornets"})                      |
+-----------------------------------------------------------+
```

Common retrieving operations

RETRIEVE VERTEX OR EDGE INFORMATION

Use `RETURN {<vertex_name> | <edge_name>}` to retrieve all the information of a vertex or an edge.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{name: "Tim Duncan", age: 42}) |
+-------------------------------------------------+

nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(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}]   |
+-------------------------------------------------------------------------+
```

RETRIEVE VIDS

Use the `id()` function to retrieve VIDs.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN id(v);
+-------------+
| id(v)       |
+-------------+
| "player100" |
+-------------+
```

RETRIEVE TAGS

Use the `labels()` function to retrieve the list of tags on a vertex.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN labels(v);
+------------+
| labels(v)  |
+------------+
| ["player"] |
+------------+
```

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 retrieve the first tag in the list.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN labels(v)[0];
+--------------+
| labels(v)[0] |
+--------------+
| "player"     |
+--------------+
```

RETRIEVE A SINGLE PROPERTY ON A VERTEX OR AN EDGE

Use `RETURN {<vertex_name> | <edge_name>}.<property>` to retrieve a single property.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v.age;
+-------+
| v.age |
+-------+
| 42    |
+-------+
```

Use `AS` to specify an alias for a property.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN v.age AS Age;
+-----+
| Age |
+-----+
| 42  |
+-----+
```

RETRIEVE ALL PROPERTIES ON A VERTEX OR AN EDGE

Use the `properties()` function to retrieve all properties on a vertex or an edge.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN properties(v2);
+-----------------------------------+
| properties(v2)                    |
+-----------------------------------+
| {"name":"Spurs"}                  |
+-----------------------------------+
| {"name":"Tony Parker", "age":36}  |
+-----------------------------------+
| {"age":41, "name":"Manu Ginobili"} |
+-----------------------------------+
```

RETRIEVE EDGE TYPES

Use the `type()` function to retrieve the matched edge types.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[e]->() \
        RETURN DISTINCT type(e);
+----------+
| type(e)  |
+----------+
| "follow" |
+----------+
| "serve"  |
+----------+
```

RETRIEVE PATHS

Use `RETURN <path_name>` to retrieve all the information of the matched paths.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*3]->() \
        RETURN p;
+-----------------------------------------------------------------------------------------+
| p                                                                                       |
+-----------------------------------------------------------------------------------------+
| <("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"})-[:serve@0 {end_year: 2019, start_year: 2015}]->("team204" :team{name: "Spurs"})>           |
+-----------------------------------------------------------------------------------------+
| <("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"})-[:serve@0 {end_year: 2015, start_year: 2006}]->("team203" :team{name: "Trail Blazers"})>   |
+-----------------------------------------------------------------------------------------+
| <("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"})-[:follow@0 {degree: 75}]->("player101" :player{age: 36, name: "Tony Parker"})>             |
+-----------------------------------------------------------------------------------------+
...
```

RETRIEVE VERTICES IN A PATH

Use the `nodes()` function to retrieve all vertices in a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN nodes(p);
+-----------------------------------------------------------------------------------------------------------------------+
| nodes(p)                                                                                                              |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player204" :team{name: "Spurs"})]                      |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player101" :player{name: "Tony Parker", age: 36})]   |
+-----------------------------------------------------------------------------------------------------------------------+
| [("player100" :star{} :player{age: 42, name: "Tim Duncan"}), ("player125" :player{name: "Manu Ginobili", age: 41})] |
+-----------------------------------------------------------------------------------------------------------------------+
```

RETRIEVE EDGES IN A PATH

Use the `relationships()` function to retrieve all edges in a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[]->(v2) \
        RETURN relationships(p);
+-------------------------------------------------------------------------+
| relationships(p)                                                        |
+-------------------------------------------------------------------------+
| [[:follow "player100"->"player101" @0 {degree: 95}]]                    |
+-------------------------------------------------------------------------+
| [[:follow "player100"->"player125" @0 {degree: 95}]]                    |
+-------------------------------------------------------------------------+
| [[:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}]]     |
+-------------------------------------------------------------------------+
```

RETRIEVE PATH LENGTH

Use the `length()` function to retrieve the length of a path.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})-[*..2]->(v2) \
        RETURN p AS Paths, length(p) AS Length;
+-----------------------------------------------------------------------+--------+
| Paths                                                                 | Length |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu Ginobili"})-[:serve@0 {end_year: 2018,
start_year: 2002}]->("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" :player{age: 42, name: "Tim Duncan"})>      | 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"})-[: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"})-[:follow@0 {degree: 95}]-
>("player125" :player{age: 41, name: "Manu Ginobili"})>     | 2      |
+-----------------------------------------------------------------------+--------+
| <("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      |
+-----------------------------------------------------------------------+--------+
| <("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      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:serve@0 {end_year: 2016, start_year: 1997}]->("team204" :team{name:
"Spurs"})>                                                              | 1      |
+-----------------------------------------------------------------------+--------+
| <("player100" :player{age: 42, name: "Tim Duncan"})-[:follow@0 {degree: 95}]->("player125" :player{age: 41, name: "Manu
Ginobili"})>                                                            | 1      |
+-----------------------------------------------------------------------+--------+
```

Performance

In Nebula Graph 2.5.0, the `MATCH` statement has initially optimized for resource usage and performance.

Simpler operations can be replaced by `GO`, `LOOKUP`, `|`, and `FETCH`.

Last update: September 2, 2021

4.6.2 LOOKUP

The `LOOKUP` statement traverses data based on indexes. You can use `LOOKUP` for the following purposes:

- Search for the specific data based on conditions defined by the `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 reduction can be 90% or even more. **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, Nebula Graph 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 `age` have indexes, but the property `name` has no indexes. When running `LOOKUP ON player WHERE player.age == 36 YIELD player.name;`, Nebula Graph randomly uses one of the indexes of the tag `player` and the property `age`.

  > **Legacy version compatibility**
  >
  > In the previous releases, if the specified property is not indexed when using the `LOOKUP` statement, Nebula Graph 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 <return_list>];

<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 <return_list>`: specifies the results to be returned and the format of the results.
- If there is a `WHERE` clause but no `YIELD` clause:
  - The Vertex ID is returned when you `LOOKUP` a tag.
  - The source vertex ID, destination vertex ID, and rank of the edge are returned when `LOOKUP` an edge type.

Limitations of using `WHERE` in `LOOKUP`

The `WHERE` clause in a `LOOKUP` statement does not support the following operations:

- `$-` and `$^`.
- 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.
- Range scan is not supported in the string-type index.
- The `XOR` and `NOT` operations are not supported.

Retrieve Vertices

The following example returns vertices whose `name` is `Tony Parker` and the tag is `player`.

```
nebula> CREATE TAG INDEX 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";
============
| VertexID |
============
| 101      |
------------

nebula> LOOKUP ON player \
        WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
======================================
| VertexID | player.name | player.age |
======================================
| 101      | Tony Parker | 36         |
--------------------------------------

nebula> LOOKUP ON player \
        WHERE player.name == "Kobe Bryant" \
        YIELD player.name AS name |\
        GO FROM $-.VertexID OVER serve \
        YIELD $-.name, serve.start_year, serve.end_year, $$.team.name;
================================================================
| $-.name     | serve.start_year | serve.end_year | $$.team.name |
================================================================
| Kobe Bryant | 1996             | 2016           | Lakers       |
----------------------------------------------------------------
```

Retrieve Edges

The following example returns edges whose `degree` is `90` and the edge type is `follow`.

```
nebula> CREATE EDGE INDEX index_follow ON follow(degree);

nebula> REBUILD EDGE INDEX index_follow;
+------------+
| New Job Id |
+------------+
| 62         |
+------------+

nebula> LOOKUP ON follow \
        WHERE follow.degree == 90;
+-------------+-------------+---------+
| SrcVID      | DstVID      | Ranking |
+-------------+-------------+---------+
| "player101" | "player102" | 0       |
+-------------+-------------+---------+
| "player133" | "player114" | 0       |
+-------------+-------------+---------+
| "player133" | "player144" | 0       |
+-------------+-------------+---------+
...

nebula> LOOKUP ON follow \
        WHERE follow.degree == 90 \
        YIELD follow.degree;
```

```
+-------------+-------------+---------+---------------+
| SrcVID      | DstVID      | Ranking | follow.degree |
+-------------+-------------+---------+---------------+
| "player101" | "player102" | 0       | 90            |
+-------------+-------------+---------+---------------+
| "player133" | "player114" | 0       | 90            |
+-------------+-------------+---------+---------------+
| "player133" | "player144" | 0       | 90            |
+-------------+-------------+---------+---------------+
...

nebula> LOOKUP ON follow \
        WHERE follow.degree == 60 \
        YIELD follow.degree AS Degree |\
        GO FROM $-.DstVID OVER serve \
        YIELD $-.DstVID, serve.start_year, serve.end_year, $$.team.name;
+-------------+------------------+----------------+--------------+
| $-.DstVID   | serve.start_year | serve.end_year | $$.team.name |
+-------------+------------------+----------------+--------------+
| "player105" | 2010             | 2018           | "Spurs"      |
+-------------+------------------+----------------+--------------+
| "player105" | 2009             | 2010           | "Cavaliers"  |
+-------------+------------------+----------------+--------------+
| "player105" | 2018             | 2019           | "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`.

```
nebula> CREATE TAG player(name string,age int);

nebula> CREATE TAG INDEX player_index on player();

nebula> REBUILD TAG INDEX player_index;
+------------+
| New Job Id |
+------------+
| 66         |
+------------+

nebula> INSERT VERTEX player(name,age) \
        VALUES "player100":("Tim Duncan", 42), "player101":("Tony Parker", 36);

The following statement retrieves the VID of all vertices with the tag `player`. It is similar to `MATCH (n:player) RETURN id(n) /*, n */`.

nebula> LOOKUP ON player;
+-------------+
| _vid        |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
```

- The following example shows how to retrieve the source Vertex IDs, destination vertex IDs, and ranks of all edges of the `like` edge type.

```
nebula> CREATE EDGE like(likeness int);

nebula> CREATE EDGE INDEX like_index on like();

nebula> REBUILD EDGE INDEX like_index;
+------------+
| New Job Id |
+------------+
| 88         |
+------------+

nebula> INSERT EDGE like(likeness) \
        VALUES "player100"->"player101":(95);

The following statement retrieves all edges with the edge type `like`. It is similar to `MATCH (s)-[e:like]->(d) RETURN id(s), rank(e), id(d) /*, type(e) */`.

nebula)> LOOKUP ON like;
+-------------+----------+-------------+
| _src        | _ranking | _dst        |
+-------------+----------+-------------+
| "player100" | 0        | "player101" |
+-------------+----------+-------------+
```

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 `like` edge type.

```
nebula> LOOKUP ON player |\
        YIELD COUNT(*) AS Player_Number;
+---------------+
| Player_Number |
+---------------+
| 2             |
+---------------+

nebula> LOOKUP ON like | \
        YIELD COUNT(*) AS Like_Number;
+-------------+
| Like_Number |
+-------------+
| 1           |
+-------------+
```

> **Note**
>
> You can also use `show-stats` to count the numbers of vertices or edges.

Last update: September 9, 2021

4.6.3 GO

`GO` traverses in a graph with specified filters and returns results.

OpenCypher compatibility

This topic applies to native nGQL only.

Syntax

```
GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
[YIELD [DISTINCT] <return_list>]
[| ORDER BY <expression> [{ASC | DESC}]]
[| LIMIT [<offset_value>,] <number_rows>]

GO [[<M> TO] <N> STEPS ] FROM <vertex_list>
OVER <edge_type_list> [{REVERSELY | BIDIRECT}]
[ WHERE <conditions> ]
[| GROUP BY {col_name | expr | position} YIELD <col_name>]

<vertex_list> ::=
    <vid> [, <vid> ...]

<edge_type_list> ::=
   edge_type [, edge_type ...]
   | *
```

```
<return_list> ::=
    <col_name> [AS <col_alias>] [, <col_name> [AS <col_alias>] ...]
```

- `<N> STEPS`: specifies the hop number. If not specified, the default value for `N` is `one`. When `N` is `zero`, Nebula Graph does not traverse any edges and returns nothing.

  > **Note**
  >
  > The path type of the `GO` statement is `walk`, which means both vertices and edges can be repeatedly visited in graph traversal. For more information, see Path.

- `M TO N 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 0 TO 2` and `GO 1 TO 2` are the same.
- `<vertex_list>`: represents a list of vertex IDs separated by commas, or a special place holder `$-.id`. For more information, see Pipe.
- `<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.
- `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.

  > **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.

- `YIELD [DISTINCT] <return_list>`: specifies the desired output. For more information, see YIELD. When not specified, the destination vertex IDs will be returned by default.
- `ORDER BY`: sorts outputs with specified orders. For more information, see ORDER BY.

  > **Note**
  >
  > When the sorting method is not specified, the output orders can be different for the same query.

- `LIMIT`: limits the number of rows of the output. For more information, see LIMIT.
- `GROUP BY`: groups outputs into subgroups based on values of the specified properties. For more information, see GROUP BY.

Examples

```
# The following example returns the teams that player 102 serves.
nebula> GO FROM "player102" \
        OVER serve;
+------------+
| serve._dst |
+------------+
| "team203"  |
+------------+
| "team204"  |
+------------+
```

```
# The following example returns the friends of player 102 with 2 hops.
nebula> GO 2 STEPS FROM "player102" \
        OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
```

```
+-------------+
...
```

```
# The following example adds a filter for the traversal.
nebula> GO FROM "player100", "player102" \
        OVER serve \
        WHERE serve.start_year > 1995 \
        YIELD DISTINCT $$.team.name AS team_name, serve.start_year AS start_year, $^.player.name AS player_name;
+-----------------+------------+--------------------+
| team_name       | start_year | player_name        |
+-----------------+------------+--------------------+
| "Spurs"         | 1997       | "Tim Duncan"       |
+-----------------+------------+--------------------+
| "Trail Blazers" | 2006       | "LaMarcus Aldridge" |
+-----------------+------------+--------------------+
| "Spurs"         | 2015       | "LaMarcus Aldridge" |
+-----------------+------------+--------------------+
```

```
# The following example traverses along with multiple edge types. If there is no value for a property, the output is __EMPTY__.
nebula> GO FROM "player100" \
        OVER follow, serve \
        YIELD follow.degree, serve.start_year;
+---------------+-----------------+
| follow.degree | serve.start_year |
+---------------+-----------------+
| 95            | __EMPTY__       |
+---------------+-----------------+
| 95            | __EMPTY__       |
+---------------+-----------------+
| __EMPTY__     | 1997            |
+---------------+-----------------+
```

```
# The following example returns the incoming edges of player 100.
nebula> GO FROM "player100" \
        OVER follow REVERSELY \
        YIELD follow._dst AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
...

# This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v)<-[e:follow]- (v2) \
        WHERE id(v) == 'player100' \
        RETURN id(v2) AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
+-------------+
| "player102" |
+-------------+
...
```

```
# The following example retrieves the friends of player 100 and the teams that they serve.
nebula> GO FROM "player100" \
        OVER follow REVERSELY \
        YIELD follow._dst AS id |\
        GO FROM $-.id OVER serve \
        WHERE $^.player.age > 20 \
        YIELD $^.player.name AS FriendOf, $$.team.name AS Team;
+--------------------+-----------------+
| FriendOf           | Team            |
+--------------------+-----------------+
| "Tony Parker"      | "Spurs"         |
+--------------------+-----------------+
| "Tony Parker"      | "Hornets"       |
+--------------------+-----------------+
...

# This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v)<-[e:follow]- (v2)-[e2:serve]->(v3)  \
        WHERE id(v) == 'player100' \
        RETURN v2.name AS FriendOf, v3.name AS Team;
+--------------------+-----------------+
| FriendOf           | Team            |
+--------------------+-----------------+
| "Tony Parker"      | "Spurs"         |
+--------------------+-----------------+
| "Tony Parker"      | "Hornets"       |
+--------------------+-----------------+
...
```

```
# The following example returns the outgoing edges and the incoming edges of player 102.
nebula> GO FROM "player102" \
        OVER follow BIDIRECT \
```

```
        YIELD follow._dst AS both;
+-------------+
| both        |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
...

# This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v) -[e:follow]-(v2) \
        WHERE id(v)== "player102" \
        RETURN id(v2) AS both;
+-------------+
| both        |
+-------------+
| "player101" |
+-------------+
| "player103" |
+-------------+
...
```

```
# The following example retrieves the friends of player 100 within 1 or 2 hops.
nebula> GO 1 TO 2 STEPS FROM "player100" \
        OVER follow \
        YIELD follow._dst AS destination;
+-------------+
| destination |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
...

# This MATCH query shares the same semantics with the preceding GO query.
nebula> MATCH (v) -[e:follow*1..2]->(v2) \
        WHERE id(v) == "player100" \
        RETURN id(v2) AS destination;
+-------------+
| destination |
+-------------+
| "player100" |
+-------------+
| "player102" |
+-------------+
```

```
# The following example the outputs according to age.
nebula> GO 2 STEPS FROM "player100" \
        OVER follow \
        YIELD follow._src AS src, follow._dst AS dst, $$.player.age AS age |\
        GROUP BY $-.dst \
        YIELD $-.dst AS dst, collect_set($-.src) AS src, collect($-.age) AS age;
+-------------+---------------------------+----------+
| dst         | src                       | age      |
+-------------+---------------------------+----------+
| "player125" | ["player101"]             | [41]     |
+-------------+---------------------------+----------+
| "player100" | ["player125", "player101"] | [42, 42] |
+-------------+---------------------------+----------+
| "player102" | ["player101"]             | [33]     |
+-------------+---------------------------+----------+
```

```
# The following example groups the outputs and restricts the number of rows of the outputs.
nebula> $a = GO FROM "player100" \
        OVER follow YIELD follow._src AS src, follow._dst AS dst; \
        GO 2 STEPS FROM $a.dst OVER follow \
        YIELD $a.src AS src, $a.dst, follow._src, follow._dst |\
        ORDER BY $-.src |\
        OFFSET 1 LIMIT 2;
+-------------+-------------+-------------+-------------+
| src         | $a.dst      | follow._src | follow._dst |
+-------------+-------------+-------------+-------------+
| "player100" | "player125" | "player100" | "player101" |
+-------------+-------------+-------------+-------------+
| "player100" | "player101" | "player100" | "player125" |
+-------------+-------------+-------------+-------------+
```

```
# The following example determines if $$.player.name IS NOT EMPTY.
nebula> GO FROM "player100" \
        OVER * \
        WHERE $$.player.name IS NOT EMPTY \
        YIELD follow._dst;
+-------------+
| follow._dst |
+-------------+
| "player125" |
+-------------+
| "player101" |
+-------------+
```

Last update: August 3, 2021

4.6.4 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 <output>]
```

Parameter	Description
`tag_name`	The name of the tag.
`*`	Represents all the tags in the current graph space.
`vid`	The vertex ID.
`output`	Specifies the information to be returned. For more information, see `YIELD`. If there is no `YIELD` clause, `FETCH` returns all the matched information.

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";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+
```

FETCH SPECIFIC PROPERTIES OF A VERTEX

Use a `YIELD` clause to specify the properties to be returned.

```
nebula> FETCH PROP ON player "player100" \
        YIELD player.name;
+-------------+-------------+
| VertexID    | player.name  |
+-------------+-------------+
| "player100" | "Tim Duncan" |
+-------------+-------------+
```

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";
+-------------------------------------------------------+
| vertices_                                             |
+-------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})   |
+-------------------------------------------------------+
| ("player102" :player{age: 33, name: "LaMarcus Aldridge"}) |
+-------------------------------------------------------+
| ("player103" :player{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 t1(a string, b int);
```

```
# The following example attaches t1 to the vertex "player100".
nebula> INSERT VERTEX t1(a, b) VALUE "player100":("Hello", 100);

# The following example fetches the properties of vertex "player100" by the tags player and t1.
nebula> FETCH PROP ON player, t1 "player100";
+------------------------------------------------------------------------------+
| vertices_                                                                    |
+------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+------------------------------------------------------------------------------+
```

You can combine multiple tags with multiple VIDs in a `FETCH` statement.

```
nebula> FETCH PROP ON player, t1 "player100", "player103";
+------------------------------------------------------------------------------+
| vertices_                                                                    |
+------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+------------------------------------------------------------------------------+
| ("player103" :player{age: 32, 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";
+------------------------------------------------------------------------------+
| vertices_                                                                    |
+------------------------------------------------------------------------------+
| ("player106" :player{age: 25, name: "Kyle Anderson"})                        |
+------------------------------------------------------------------------------+
| ("team200" :team{name: "Warriors"})                                          |
+------------------------------------------------------------------------------+
| ("player100" :t1{a: "Hello", b: 100} :player{age: 42, name: "Tim Duncan"}) |
+------------------------------------------------------------------------------+
```

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.
`output`	Specifies the information to be returned. For more information, see `YIELD`. If there is no `YIELD` clause, `FETCH` returns all the matched information.

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";
+------------------------------------------------------------------------------+
| edges_                                                                       |
+------------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {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 PROP ON serve "player100" -> "team204"    \
        YIELD serve.start_year;
+-------------+------------+-------------+------------------+
```

```
| serve._src  | serve._dst | serve._rank | serve.start_year |
+-------------+------------+-------------+------------------+
| "player100" | "team204"  | 0           | 1997             |
+-------------+------------+-------------+------------------+
```

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.

```
nebula> FETCH PROP ON serve "player100" -> "team204", "player133" -> "team202";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
+------------------------------------------------------------------------+
| [:serve "player133"->"team202" @0 {end_year: 2011, start_year: 2002}] |
+------------------------------------------------------------------------+
```

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.

```
# The following example inserts edges with different ranks and property values.
nebula> insert edge serve(start_year,end_year) \
        values "player100"->"team204"@1:(1998, 2017);

nebula> insert edge serve(start_year,end_year) \
        values "player100"->"team204"@2:(1990, 2018);

# By default, the FETCH statement returns the edge whose rank is 0.
nebula> FETCH PROP ON serve "player100" -> "team204";
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] |
+------------------------------------------------------------------------+

# To fetch on an edge whose rank is not 0, set its rank in the FETCH statement.
nebula> FETCH PROP ON serve "player100" -> "team204"@1;
+------------------------------------------------------------------------+
| edges_                                                                 |
+------------------------------------------------------------------------+
| [:serve "player100"->"team204" @1 {end_year: 2017, start_year: 1998}] |
+------------------------------------------------------------------------+
```

Use FETCH in composite queries

A common way to use `FETCH` is to combine it with native nGQL such as `GO`.

The following statement returns the `degree` values of the `follow` edges that start from vertex `"player101"`.

```
nebula> GO FROM "player101" OVER follow \
        YIELD follow._src AS s, follow._dst AS d |\
        FETCH PROP ON follow $-.s -> $-.d \
        YIELD follow.degree;
+-------------+-------------+--------------+---------------+
| follow._src | follow._dst | follow._rank | follow.degree |
+-------------+-------------+--------------+---------------+
| "player101" | "player100" | 0            | 95            |
+-------------+-------------+--------------+---------------+
| "player101" | "player102" | 0            | 90            |
+-------------+-------------+--------------+---------------+
| "player101" | "player125" | 0            | 95            |
+-------------+-------------+--------------+---------------+
```

Or you can use user-defined variables to construct similar queries.

```
nebula> $var = GO FROM "player101" OVER follow \
        YIELD follow._src AS s, follow._dst AS d; \
        FETCH PROP ON follow $var.s -> $var.d \
        YIELD follow.degree;
+-------------+-------------+--------------+---------------+
| follow._src | follow._dst | follow._rank | follow.degree |
+-------------+-------------+--------------+---------------+
| "player101" | "player100" | 0            | 95            |
+-------------+-------------+--------------+---------------+
| "player101" | "player102" | 0            | 90            |
+-------------+-------------+--------------+---------------+
```

```
| "player101" | "player125" | 0            | 95           |
+-------------+-------------+--------------+--------------+
```

For more information about composite queries, see Composite queries (clause structure).

Last update: August 3, 2021

4.6.5 UNWIND

The `UNWIND` statement splits a list into separated rows.

`UNWIND` can function as an individual statement or a clause in a statement.

Syntax

```
UNWIND <list> AS <alias> <RETURN clause>;
```

Split a list

The following example splits the list `[1,2,3]` into three rows.

```
nebula> UNWIND [1,2,3] AS n RETURN n;
+---+
| n |
+---+
| 1 |
+---+
| 2 |
+---+
| 3 |
+---+
```

Return a list with distinct items

Use `WITH DISTINCT` in the `UNWIND` statement to return a list with distinct items.

EXAMPLE 1

The following statement:

1. Splits the list `[1,1,2,2,3,3]` into rows.
2. Removes duplicated rows.
3. Sorts the rows.
4. Transforms the rows to a list.

```
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]  |
+------------+
```

Example 2

The following statement:

1. Outputs the vertices on the matched path into a list.
2. Splits the list into rows.
3. Removes duplicated rows.
4. Transforms the rows to a list.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--(v2) \
        WITH nodes(p) AS n \
        UNWIND n AS r \
        WITH DISTINCT r AS r \
        RETURN collect(r);
+-------------------------------------------------------------------------------------------------------------------+
| collect(r)                                                                                                        |
+-------------------------------------------------------------------------------------------------------------------+
```

```
| [("player100" :player{age: 42, name: "Tim Duncan"}), ("player101" :player{age: 36, name: "Tony Parker"}),
("team204" :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: 47, name: "Shaquile O'Neal"}), ("player105" :player{age: 31, name: "Danny Green"}),
("player113" :player{age: 29, name: "Dejounte Murray"}), ("player107" :player{age: 32, name: "Aron Baynes"}),
("player109" :player{age: 34, name: "Tiago Splitter"}), ("player108" :player{age: 36, name: "Boris Diaw"})] |
+-----------------------------------------------------------------------------------------------------------------+
```

Last update: August 31, 2021

4.6.6 SHOW

SHOW CHARSET

The `SHOW CHARSET` statement shows the available character sets.

Currently available types are `utf8` and `utf8mb4`. The default charset type is `utf8`. Nebula Graph extends the `uft8` to support four-byte characters. Therefore `utf8` and `utf8mb4` are equivalent.

SYNTAX

```
SHOW CHARSET;
```

EXAMPLE

```
nebula> SHOW CHARSET;
+---------+-----------------+-------------------+--------+
| Charset | Description     | Default collation | Maxlen |
+---------+-----------------+-------------------+--------+
| "utf8"  | "UTF-8 Unicode" | "utf8_bin"        | 4      |
+---------+-----------------+-------------------+--------+
```

Parameter	Description
`Charset`	The name of the character set.
`Description`	The description of the character set.
`Default collation`	The default collation of the character set.
`Maxlen`	The maximum number of bytes required to store one character.

Last update: July 27, 2021

SHOW COLLATION

The `SHOW COLLATION` statement shows the collations supported by Nebula Graph.

Currently available types are: `utf8_bin`, `utf8_general_ci`, `utf8mb4_bin`, and `utf8mb4_general_ci`.

- When the character set is `utf8`, the default collate is `utf8_bin`.
- When the character set is `utf8mb4`, the default collate is `utf8mb4_bin`.
- Both `utf8mb4_bin` and `utf8mb4_general_ci` are case-insensitive.

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.

Last update: July 27, 2021

SHOW CREATE SPACE

The `SHOW CREATE SPACE` 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 SPACE basketballplayer;
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| Space              | Create Space                                                                                                                                                    |
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
| "basketballplayer" | "CREATE SPACE `basketballplayer` (partition_num = 10, replica_factor = 1, charset = utf8, collate = utf8_bin, vid_type = FIXED_STRING(32))" |
+--------------------+-----------------------------------------------------------------------------------------------------------------------------------------------------------------+
```

Last update: July 27, 2021

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" | "CREATE TAG `player` (           |
|          |  `name` string NULL,             |
|          |  `age` int64 NULL                |
|          | ) ttl_duration = 0, ttl_col = "" |
+----------+----------------------------------+

nebula> SHOW CREATE EDGE follow;
+----------+----------------------------------+
| Edge     | Create Edge                      |
+----------+----------------------------------+
| "follow" | "CREATE EDGE `follow` (          |
|          |  `degree` int64 NULL             |
|          | ) ttl_duration = 0, ttl_col = """ |
+----------+----------------------------------+
```

Last update: August 4, 2021

SHOW HOSTS

The `SHOW HOSTS` statement shows the host and version information of Graph Service, Storage Service, and Meta Service.

SYNTAX

```
SHOW HOSTS [GRAPH | STORAGE | META];
```

If you return `SHOW HOSTS` without the service name, it will show the host information of Storage Service, as well as the leader number, leader distribution, and partition distribution.

EXAMPLES

```
nebula> SHOW HOSTS;
+-------------+-------+----------+--------------+---------------------------------+----------------------------+
| Host        | Port  | Status   | Leader count | Leader distribution             | Partition distribution     |
+-------------+-------+----------+--------------+---------------------------------+----------------------------+
| "storaged0" | 9779  | "ONLINE" | 8            | "docs:5, basketballplayer:3"    | "docs:5, basketballplayer:3"|
+-------------+-------+----------+--------------+---------------------------------+----------------------------+
| "storaged1" | 9779  | "ONLINE" | 9            | "basketballplayer:4, docs:5"    | "docs:5, basketballplayer:4"|
+-------------+-------+----------+--------------+---------------------------------+----------------------------+
| "storaged2" | 9779  | "ONLINE" | 8            | "basketballplayer:3, docs:5"    | "docs:5, basketballplayer:3"|
+-------------+-------+----------+--------------+---------------------------------+----------------------------+

nebula> SHOW HOSTS GRAPH;
+-----------+------+----------+---------+--------------+-----------------------------------------+
| Host      | Port | Status   | Role    | Git Info Sha | Version                                 |
+-----------+------+----------+---------+--------------+-----------------------------------------+
| "graphd"  | 9669 | "ONLINE" | "GRAPH" | "c397299c"   | "2.5.0, Build Time: Aug 19 2021 11:20:18" |
+-----------+------+----------+---------+--------------+-----------------------------------------+
| "graphd1" | 9669 | "ONLINE" | "GRAPH" | "c397299c"   | "2.5.0, Build Time: Aug 19 2021 11:20:18" |
+-----------+------+----------+---------+--------------+-----------------------------------------+
| "graphd2" | 9669 | "ONLINE" | "GRAPH" | "c397299c"   | "2.5.0, Build Time: Aug 19 2021 11:20:18" |
+-----------+------+----------+---------+--------------+-----------------------------------------+

nebula> SHOW HOSTS STORAGE;
+-------------+------+----------+-----------+--------------+-----------------------------------------+
| Host        | Port | Status   | Role      | Git Info Sha | Version                                 |
+-------------+------+----------+-----------+--------------+-----------------------------------------+
| "storaged0" | 9779 | "ONLINE" | "STORAGE" | "5b83e5cb"   | "2.5.0, Build Time: Aug 19 2021 10:30:28" |
+-------------+------+----------+-----------+--------------+-----------------------------------------+
| "storaged1" | 9779 | "ONLINE" | "STORAGE" | "5b83e5cb"   | "2.5.0, Build Time: Aug 19 2021 10:30:28" |
+-------------+------+----------+-----------+--------------+-----------------------------------------+
| "storaged2" | 9779 | "ONLINE" | "STORAGE" | "5b83e5cb"   | "2.5.0, Build Time: Aug 19 2021 10:30:28" |
+-------------+------+----------+-----------+--------------+-----------------------------------------+

nebula> SHOW HOSTS META;
+----------+------+----------+--------+--------------+-----------------------------------------+
| Host     | Port | Status   | Role   | Git Info Sha | Version                                 |
+----------+------+----------+--------+--------------+-----------------------------------------+
| "metad2" | 9559 | "ONLINE" | "META" | "5b83e5cb"   | "2.5.0, Build Time: Aug 19 2021 10:30:28" |
+----------+------+----------+--------+--------------+-----------------------------------------+
| "metad0" | 9559 | "ONLINE" | "META" | "5b83e5cb"   | "2.5.0, Build Time: Aug 19 2021 10:30:28" |
+----------+------+----------+--------+--------------+-----------------------------------------+
| "metad1" | 9559 | "ONLINE" | "META" | "5b83e5cb"   | "2.5.0, Build Time: Aug 19 2021 10:30:28" |
+----------+------+----------+--------+--------------+-----------------------------------------+
```

Last update: August 31, 2021

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 |
+---------------+-------------+
| "like_index_0" | "FINISHED"   |
+---------------+-------------+
| "like1"        | "FINISHED"   |
+---------------+-------------+

nebula> SHOW EDGE INDEX STATUS;
+---------------+-------------+
| Name          | Index Status |
+---------------+-------------+
| "index_follow" | "FINISHED"   |
+---------------+-------------+
```

RELATED TOPICS

- Job manager and the JOB statements
- REBUILD NATIVE INDEX

Last update: July 27, 2021

SHOW INDEXES

The `SHOW INDEXES` statement shows the names of existing native indexes.

SYNTAX

```
SHOW {TAG | EDGE} INDEXES;
```

EXAMPLES

```
nebula> SHOW TAG INDEXES;
+------------------+-------------+-----------------+
| Index Name       | By Tag      | Columns         |
+------------------+-------------+-----------------+
| "fix"            | "fix_string" | ["p1"]          |
+------------------+-------------+-----------------+
| "player_index_0" | "player"     | ["name"]        |
+------------------+-------------+-----------------+
| "player_index_1" | "player"     | ["name", "age"] |
+------------------+-------------+-----------------+
| "var"            | "var_string" | ["p1"]          |
+------------------+-------------+-----------------+

nebula> SHOW EDGE INDEXES;
+---------------+----------+---------+
| Index Name     | By Edge  | Columns |
+---------------+----------+---------+
| "follow_index" | "follow" | []      |
+---------------+----------+---------+
```

Legacy version compatibility

In Nebula Graph 2.0.1, `SHOW TAG/EDGE INDEXES` only returns `Names`.

Last update: August 31, 2021

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;
+--------------+-------------------+-------------------+-------+
| 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 address and the port of the leader.
`Peers`	The IP addresses and the ports of all the replicas.
`Losts`	The IP addresses and the ports of replicas at fault.

Last update: August 23, 2021

SHOW ROLES

The `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, Nebula Graph 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, Nebula Graph shows the user's own role in this graph space.
- If the user does not have access to the specified graph space, Nebula Graph 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"   |
+---------+-----------+
```

Last update: July 27, 2021

SHOW SNAPSHOTS

The `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 `GOD` role can use the `SHOW 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"    |
+-------------------------------+---------+-------------------------------------------------+
| "SNAPSHOT_2020_12_16_11_14_10" | "VALID" | "storaged0:9779, storaged1:9779, storaged2:9779"    |
+-------------------------------+---------+-------------------------------------------------+
```

Last update: July 27, 2021

SHOW SPACES

The `SHOW SPACES` statement shows existing graph spaces in Nebula Graph.

For how to create a graph space, see CREATE SPACE.

SYNTAX

```
SHOW SPACES;
```

EXAMPLE

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

Last update: July 27, 2021

SHOW STATS

The `SHOW STATS` statement shows the statistics of the graph space collected by the latest `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

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) | Command(Dest) | Status     | Start Time | Stop Time  |
+---------------+---------------+------------+------------+------------+
| 98             | "STATS"       | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
| 0              | "storaged2"   | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
| 1              | "storaged0"   | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+
| 2              | "storaged1"   | "FINISHED" | 1606552675 | 1606552675 |
+---------------+---------------+------------+------------+------------+

# Show the statistics of the graph space.
nebula> SHOW STATS;
+---------+------------+-------+
| Type    | Name       | Count |
+---------+------------+-------+
| "Tag"   | "player"   | 51    |
+---------+------------+-------+
| "Tag"   | "team"     | 30    |
+---------+------------+-------+
| "Edge"  | "like"     | 81    |
+---------+------------+-------+
| "Edge"  | "serve"    | 152   |
+---------+------------+-------+
| "Space" | "vertices" | 81    |
+---------+------------+-------+
| "Space" | "edges"    | 233   |
+---------+------------+-------+
```

Last update: July 27, 2021

SHOW TAGS/EDGES

The `SHOW TAGS` statement shows all the tags in the current graph space.

The `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    |
+---------+
| "like"  |
+---------+
| "serve" |
+---------+
```

Last update: August 31, 2021

SHOW USERS

The `SHOW USERS` statement shows the user information.

ROLE REQUIREMENT

Only the `root` user who has the `GOD` role can use the `SHOW USERS` statement.

SYNTAX

```
SHOW USERS;
```

EXAMPLE

```
nebula> SHOW USERS;
+---------+
| Account |
+---------+
| "root"  |
+---------+
| "user1" |
+---------+
```

Last update: July 27, 2021

SHOW SESSIONS

The `SHOW SESSIONS` statement shows the information of all the sessions. It can also show a specified session with its ID.

PRECAUTIONS

When you log in to the database using Nebula Console, a session will be created. The client will execute 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 unexpected ways with the `session_idle_timeout_secs` in nebula-graphd.conf undetermined, the session will not be released automatically.

For those sessions that are not automatically released, you need to delete them manually (TODO: coding).

SYNTAX

```
SHOW SESSIONS;
SHOW SESSION <Session_Id>;
```

EXAMPLES

```
nebula> SHOW SESSIONS;
+------------------+----------+--------------------+----------------------------+----------------------------+--------------+----------+----------------+
| SessionId        | UserName | SpaceName          | CreateTime                 | UpdateTime                 | GraphAddr    | Timezone | ClientIp       |
+------------------+----------+--------------------+----------------------------+----------------------------+--------------+----------+----------------+
| 1623305056644097 | "user1"  | ""                 | 2021-06-10T06:04:16.644097 | 2021-06-10T06:04:16.638039 | "graphd:9669" | 0        | "172.22.xx.xx" |
+------------------+----------+--------------------+----------------------------+----------------------------+--------------+----------+----------------+
| 1623304491050858 | "root"   | "basketballplayer" | 2021-06-10T05:54:51.50858  | 2021-06-10T06:17:31.5417   | "graphd:9669" | 0        | "172.22.xx.xx" |
+------------------+----------+--------------------+----------------------------+----------------------------+--------------+----------+----------------+

nebula> SHOW SESSION 1623304491050858;
+--------------+----------------------------+
| VariableName | Value                      |
+--------------+----------------------------+
| "SessionID"  | 1623304491050858           |
+--------------+----------------------------+
| "UserName"   | "root"                     |
+--------------+----------------------------+
| "SpaceName"  | "basketballplayer"         |
+--------------+----------------------------+
| "CreateTime" | 2021-06-10T05:54:51.50858  |
+--------------+----------------------------+
| "UpdateTime" | 2021-06-10T06:17:34.866137 |
+--------------+----------------------------+
| "GraphAddr"  | "graphd:9669"              |
+--------------+----------------------------+
| "Timezone"   | 0                          |
+--------------+----------------------------+
| "ClientIp"   | "172.22.xx.xx"             |
+--------------+----------------------------+
```

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 address and port of the Graph server that hosts the session.
`Timezone`	A reserved parameter that has no specified meaning for now.
`ClientIp`	The IP address of the client.

Last update: July 27, 2021

SHOW QUERIES

The `SHOW QUERIES` statement shows the information of working queries in the current session.

> **Note**
>
> To terminate queries, see Kill Query.

PRECAUTIONS

- The `SHOW QUERIES` statement gets the status of queries in the current session from the local cache with almost no latency.
- The `SHOW ALL 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

```
SHOW [ALL] QUERIES;
```

EXAMPLES

```
nebula> SHOW QUERIES;
+------------------+-----------------+--------+---------------------+---------------------------+---------------+-----------+-----------------+
| SessionID        | ExecutionPlanID | User   | Host                | StartTime                 | DurationInUSec | Status    | Query           |
+------------------+-----------------+--------+---------------------+---------------------------+---------------+-----------+-----------------+
| 1625463842921750 | 46              | "root" | ""192.168.x.x":9669" | 2021-07-05T05:44:19.502903 | 0             | "RUNNING" | "SHOW QUERIES;" |
+------------------+-----------------+--------+---------------------+---------------------------+---------------+-----------+-----------------+

nebula> SHOW ALL QUERIES;
+------------------+-----------------+---------+---------------------+---------------------------+---------------+-----------
+-----------------------------------------------------+
| SessionID        | ExecutionPlanID | User    | Host                | StartTime                 | DurationInUSec | Status    |
Query                                                 |
+------------------+-----------------+---------+---------------------+---------------------------+---------------+-----------
+-----------------------------------------------------+
| 1625456037718757 | 54              | "user1" | ""192.168.x.x":9669" | 2021-07-05T05:51:08.691318 | 1504502       | "RUNNING" | "MATCH p=(v:player)-[*1..4]-(v2)
RETURN v2 AS Friends;" |
+------------------+-----------------+---------+---------------------+---------------------------+---------------+-----------
+-----------------------------------------------------+

# The following statement returns the top 10 queries that have the longest duration.
nebula> SHOW ALL QUERIES | ORDER BY $-.DurationInUSec DESC | LIMIT 10;
+------------------+-----------------+---------+---------------------+---------------------------+---------------+-----------
+-----------------------------------------------------+
| SessionID        | ExecutionPlanID | User    | Host                | StartTime                 | DurationInUSec | Status    |
Query                                                 |
+------------------+-----------------+---------+---------------------+---------------------------+---------------+-----------
+-----------------------------------------------------+
| 1625471375320831 | 98              | "user2" | ""192.168.x.x":9669" | 2021-07-05T07:50:24.461779 | 2608176       | "RUNNING" | "MATCH (v:player)-[*1..4]-(v2)
RETURN v2 AS Friends;" |
+------------------+-----------------+---------+---------------------+---------------------------+---------------+-----------
+-----------------------------------------------------+
| 1625456037718757 | 99              | "user1" | ""192.168.x.x":9669" | 2021-07-05T07:50:24.910616 | 2159333       | "RUNNING" | "MATCH (v:player)-[*1..4]-(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: July 27, 2021

4.7 Clauses and options

4.7.1 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.name AS Name, count(*) as cnt ORDER BY cnt DESC;
+----------------------+-----+
| 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> GO FROM "player100" OVER follow BIDIRECT \
        YIELD $$.player.name as Name \
        | GROUP BY $-.Name \
        YIELD $-.Name as Player, count(*) AS Name_Count;
+--------------------+------------+
| Player             | Name_Count |
+--------------------+------------+
| "Tiago Splitter"   | 1          |
+--------------------+------------+
| "Aron Baynes"      | 1          |
+--------------------+------------+
| "Boris Diaw"       | 1          |
+--------------------+------------+
| "Manu Ginobili"    | 2          |
+--------------------+------------+
| "Dejounte Murray"  | 1          |
+--------------------+------------+
| "Danny Green"      | 1          |
+--------------------+------------+
| "Tony Parker"      | 2          |
+--------------------+------------+
| "Shaquille O'Neal"  | 1         |
+--------------------+------------+
| "LaMarcus Aldridge" | 1         |
+--------------------+------------+
| "Marco Belinelli"  | 1          |
+--------------------+------------+
```

Group and calculate with functions

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> GO FROM "player100" OVER follow \
        YIELD follow._src AS player, follow.degree AS degree \
        | GROUP BY $-.player \
        YIELD sum($-.degree);
+----------------+
| sum($-.degree) |
+----------------+
| 190            |
+----------------+
```

For more information about the `sum()` function, see Built-in math functions.

Last update: August 23, 2021

4.7.2 LIMIT AND SKIP

The `LIMIT` clause constrains the number of rows in the output.

- Native nGQL: A pipe `|` must be used. And an offset can be ignored.
- OpenCypher style: No pipes are permitted. 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.

Native nGQL syntax

In native nGQL, `LIMIT` works the same as in `SQL`, and must be used with pipe `|`. The `LIMIT` clause accepts one or two parameters. The values of both arguments must be non-negative integers.

```
YIELD <var>
[| LIMIT [<offset_value>,] <number_rows>];
```

Parameter	Description
`var`	The columns or calculations that you wish to sort.
`offset_value`	The offset value. It defines from which row 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.

EXAMPLES

```
# 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 \
        YIELD $$.player.name AS Friend, $$.player.age AS Age \
        | ORDER BY $-.Age, $-.Friend \
        | LIMIT 1, 3;
+------------------+-----+
| Friend           | Age |
+------------------+-----+
| "Danny Green"    | 31  |
+------------------+-----+
| "Aron Baynes"    | 32  |
+------------------+-----+
| "Marco Belinelli" | 32  |
+------------------+-----+
```

OpenCypher syntax

```
RETURN <var>
[SKIP <offset>]
[LIMIT <number_rows>];
```

Parameter	Description
`var`	The columns or calculations that you wish to sort.
`offset`	The offset value. It defines from which row 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

```
nebula> MATCH (v:player) RETURN v.name AS Name, v.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  |
+-------------------------+-----+

nebula> MATCH (v:player) RETURN v.name AS Name, v.age AS Age \
        ORDER BY Age LIMIT rand32(5);
+-------------------------+-----+
| Name                    | Age |
+-------------------------+-----+
| "Luka Doncic"           | 20  |
+-------------------------+-----+
| "Ben Simmons"           | 22  |
+-------------------------+-----+
| "Kristaps Porzingis"    | 23  |
+-------------------------+-----+
| "Giannis Antetokounmpo" | 24  |
+-------------------------+-----+
```

EXAMPLES OF SKIP

You can use `SKIP <offset>` to skip the top N rows of the output and return the rest of the output. So, there is no need to add `LIMIT <number_rows>`.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Manu Ginobili" | 41  |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+

nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1+1;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 36  |
+---------------+-----+
```

You can use `SKIP <offset>` and `LIMIT <number_rows>` together to return the data of the middle N rows.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age \
        ORDER BY Age DESC SKIP 1 LIMIT 1;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

Last update: August 30, 2021

4.7.3 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] ...];
```

> **Compatibility**
>
> In the native nGQL syntax, `$-.` must be used after `ORDER BY` . But it is not required in releases prior to 2.5.0.

EXAMPLES

```
nebula> FETCH PROP ON player "player100", "player101", "player102", "player103" \
        YIELD player.age AS age, player.name AS name \
        | ORDER BY $-.age ASC, $-.name DESC;
+-------------+-----+--------------------+
| VertexID    | age | name               |
+-------------+-----+--------------------+
| "player103" | 32  | "Rudy Gay"         |
+-------------+-----+--------------------+
| "player102" | 33  | "LaMarcus Aldridge" |
+-------------+-----+--------------------+
| "player101" | 36  | "Tony Parker"      |
+-------------+-----+--------------------+
| "player100" | 42  | "Tim Duncan"       |
+-------------+-----+--------------------+
```

OpenCypher Syntax

```
<RETURN clause>
ORDER BY <expression> [ASC | DESC] [, <expression> [ASC | DESC] ...];
```

EXAMPLES

```
nebula> MATCH (v:player) RETURN v.name AS Name, v.age AS Age  \
        ORDER BY Name DESC;
+-----------------+-----+
| Name            | Age |
+-----------------+-----+
| "Yao Ming"      | 38  |
+-----------------+-----+
| "Vince Carter"  | 42  |
+-----------------+-----+
| "Tracy McGrady" | 39  |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
| "Tim Duncan"    | 42  |
+-----------------+-----+
...

# 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.age AS Age, v.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.

```
nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age  \
        ORDER BY Age;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Tony Parker"   | 36       |
+-----------------+----------+
| "Manu Ginobili" | 41       |
+-----------------+----------+
| "Spurs"         | __NULL__ |
+-----------------+----------+

nebula> MATCH (v:player{name:"Tim Duncan"}) --> (v2) \
        RETURN v2.name AS Name, v2.age AS Age  \
        ORDER BY Age DESC;
+-----------------+----------+
| Name            | Age      |
+-----------------+----------+
| "Spurs"         | __NULL__ |
+-----------------+----------+
| "Manu Ginobili" | 41       |
+-----------------+----------+
| "Tony Parker"   | 36       |
+-----------------+----------+
```

Last update: August 30, 2021

4.7.4 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 `YIELD`.

`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);
```

Legacy version compatibility

- In nGQL 1.x, `RETURN` works with native nGQL with the `RETURN <var_ref> IF <var_ref> IS NOT NULL` syntax.
- In nGQL 2.0, `RETURN` does not work with native nGQL.

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

```
nebula> MATCH (v:player) \
        RETURN v;
+-------------------------------------------------------------+
| v                                                           |
+-------------------------------------------------------------+
| ("player104" :player{age: 32, name: "Marco Belinelli"})     |
+-------------------------------------------------------------+
| ("player107" :player{age: 32, name: "Aron Baynes"})         |
+-------------------------------------------------------------+
| ("player116" :player{age: 34, name: "LeBron James"})        |
+-------------------------------------------------------------+
| ("player120" :player{age: 29, name: "James Harden"})        |
+-------------------------------------------------------------+
| ("player125" :player{age: 41, name: "Manu Ginobili"})       |
+-------------------------------------------------------------+
...
```

Return edges

```
nebula> MATCH (v:player)-[e]->() \
        RETURN e;
+------------------------------------------------------------------------------+
| e                                                                            |
+------------------------------------------------------------------------------+
| [:follow "player104"->"player100" @0 {degree: 55}]                           |
+------------------------------------------------------------------------------+
| [:follow "player104"->"player101" @0 {degree: 50}]                           |
+------------------------------------------------------------------------------+
| [:follow "player104"->"player105" @0 {degree: 60}]                           |
+------------------------------------------------------------------------------+
| [:serve "player104"->"team200" @0 {end_year: 2009, start_year: 2007}]        |
+------------------------------------------------------------------------------+
| [:serve "player104"->"team208" @0 {end_year: 2016, start_year: 2015}]        |
+------------------------------------------------------------------------------+
...
```

Return properties

To return a vertex or edge property, use the `{<vertex_name>|<edge_name>}.<property>` syntax.

```
nebula> MATCH (v:player) \
        RETURN v.name, v.age \
        LIMIT 3;
+------------------+-------+
| v.name           | v.age |
+------------------+-------+
| "Rajon Rondo"    | 33    |
+------------------+-------+
| "Rudy Gay"       | 32    |
+------------------+-------+
| "Dejounte Murray" | 29    |
+------------------+-------+
```

Return all elements

To return all the elements that this pattern matches, use an asterisk (*).

```
nebula> MATCH (v:player{name:"Tim Duncan"}) \
        RETURN *;
+-------------------------------------------------+
| v                                               |
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) |
+-------------------------------------------------+

nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
        RETURN *;
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
| v                                               | e                                                                          |
v2                                                |
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:follow "player100"->"player101" @0 {degree: 95}]                   | ("player101" :player{age: 36, name:
"Tony Parker"})   |
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:follow "player100"->"player125" @0 {degree: 95}]                   | ("player125" :player{age: 41, name:
"Manu Ginobili"}) |
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"}) | [:serve "player100"->"team204" @0 {end_year: 2016, start_year: 1997}] | ("team204" :team{name:
"Spurs"})                    |
+-------------------------------------------------+------------------------------------------------------------------------------
+-------------------------------------------------+
```

Rename a field

Use the `AS <alias>` syntax to rename a field in the output.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[:serve]->(v2) \
        RETURN v2.name AS Team;
+---------+
| Team    |
+---------+
| "Spurs" |
+---------+
```

```
nebula> RETURN "Amber" AS Name;
+---------+
| Name    |
+---------+
| "Amber" |
+---------+
```

Return a non-existing property

If a property matched does not exist, `NULL` is returned.

```
nebula> MATCH (v:player{name:"Tim Duncan"})-[e]->(v2) \
        RETURN v2.name, type(e), v2.age;
+-----------------+----------+----------+
| v2.name         | type(e)  | v2.age   |
+-----------------+----------+----------+
| "Tony Parker"   | "follow" | 36       |
+-----------------+----------+----------+
| "Manu Ginobili" | "follow" | 41       |
+-----------------+----------+----------+
| "Spurs"         | "serve"  | __NULL__ |
+-----------------+----------+----------+
```

Return expression results

To return the results of expressions such as literals, functions, or predicates, set them in a `RETURN` clause.

```
nebula> MATCH (v:player{name:"Tony Parker"})-->(v2:player) \
        RETURN DISTINCT v2.name, "Hello"+" graphs!", v2.age > 35;
+--------------------+-----------------+-------------+
| v2.name            | (Hello+ graphs!) | (v2.age>35) |
+--------------------+-----------------+-------------+
| "Tim Duncan"       | "Hello graphs!"  | true        |
+--------------------+-----------------+-------------+
| "LaMarcus Aldridge" | "Hello graphs!"  | false       |
+--------------------+-----------------+-------------+
| "Manu Ginobili"    | "Hello graphs!"  | true        |
+--------------------+-----------------+-------------+

nebula> RETURN 1+1;
+-------+
| (1+1) |
+-------+
| 2     |
+-------+

nebula> RETURN 3 > 1;
+-------+
| (3>1) |
+-------+
| true  |
+-------+

RETURN 1+1, rand32(1, 5);
+-------+-------------+
| (1+1) | rand32(1,5) |
+-------+-------------+
| 2     | 1           |
+-------+-------------+
```

Return unique fields

Use `DISTINCT` to remove duplicate fields in the result set.

```
# Before using DISTINCT.
nebula> MATCH (v:player{name:"Tony Parker"})--(v2:player) \
        RETURN v2.name, v2.age;
+--------------------+--------+
| v2.name            | v2.age |
+--------------------+--------+
| "Tim Duncan"       | 42     |
+--------------------+--------+
| "LaMarcus Aldridge" | 33     |
+--------------------+--------+
| "Marco Belinelli"  | 32     |
+--------------------+--------+
| "Boris Diaw"       | 36     |
+--------------------+--------+
| "Dejounte Murray"  | 29     |
+--------------------+--------+
| "Tim Duncan"       | 42     |
+--------------------+--------+
| "LaMarcus Aldridge" | 33     |
```

```
+--------------------+--------+
| "Manu Ginobili"    | 41     |
+--------------------+--------+

# After using DISTINCT.
nebula> MATCH (v:player{name:"Tony Parker"})--(v2:player) \
        RETURN DISTINCT v2.name, v2.age;
+--------------------+--------+
| v2.name            | v2.age |
+--------------------+--------+
| "Tim Duncan"       | 42     |
+--------------------+--------+
| "LaMarcus Aldridge" | 33     |
+--------------------+--------+
| "Marco Belinelli"  | 32     |
+--------------------+--------+
| "Boris Diaw"       | 36     |
+--------------------+--------+
| "Dejounte Murray"  | 29     |
+--------------------+--------+
| "Manu Ginobili"    | 41     |
+--------------------+--------+
```

Last update: August 30, 2021

4.7.5 TTL

TTL (Time To Live) specifies a timeout for a property. Once timed out, the property expires.

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.

Data expiration and deletion

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.

Nebula Graph automatically deletes the expired data and reclaims the disk space during the next compaction.

> **Note**
>
> If TTL is disabled, the corresponding data deleted after the last compaction can be queried again.

TTL options

The native nGQL TTL feature has the following options.

Option	Description
`ttl_col`	Specifies the property to set a timeout 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.

Use TTL options

You must use the TTL options together to set a valid timeout on a property.

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 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());
```

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 t2(a int, b int, c string) ttl_duration= 100, ttl_col = "a";

# Insert a vertex with tag t2. The timeout timestamp is 1612778164774 (1612778164674 + 100).
nebula> INSERT VERTEX t2(a, b, c) values "102":(1612778164674, 30, "Hello");
```

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: August 5, 2021

4.7.6 WHERE

The `WHERE` clause filters the output by conditions.

The `WHERE` clause usually works in the following queries:

- Native nGQL: such as `GO` and `LOOKUP`.
- OpenCypher syntax: such as `MATCH` and `WITH`.

OpenCypher compatibility

- Using patterns in `WHERE` is not supported (TODO: planning), for example `WHERE (v)-->(v2)`.
- 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.name == "Tim Duncan" \
        XOR (v.age < 30 AND v.name == "Yao Ming") \
        OR NOT (v.name == "Yao Ming" OR v.name == "Tim Duncan") \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Marco Belinelli"       | 32    |
+-------------------------+-------+
| "Aron Baynes"           | 32    |
+-------------------------+-------+
| "LeBron James"          | 34    |
+-------------------------+-------+
| "James Harden"          | 29    |
+-------------------------+-------+
| "Manu Ginobili"         | 41    |
+-------------------------+-------+
...
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE follow.degree > 90 \
        OR $$.player.age != 33 \
        AND $$.player.name != "Tony Parker";
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
```

FILTER ON PROPERTIES

Use vertex or edge properties to define conditions in `WHERE` clauses.

- Filter on a vertex property:

```
nebula> MATCH (v:player)-[e]->(v2) \
        WHERE v2.age < 25 \
        RETURN v2.name, v2.age;
+----------------------+--------+
| v2.name              | v2.age |
+----------------------+--------+
| "Luka Doncic"        | 20     |
+----------------------+--------+
| "Kristaps Porzingis" | 23     |
+----------------------+--------+
| "Ben Simmons"        | 22     |
+----------------------+--------+
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE $^.player.age >= 42;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
```

- Filter on an edge property:

```
nebula> MATCH (v:player)-[e]->() \
        WHERE e.start_year < 2000 \
        RETURN DISTINCT v.name, v.age;
+--------------------+-------+
| v.name             | v.age |
+--------------------+-------+
| "Shaquille O'Neal" | 47    |
+--------------------+-------+
| "Steve Nash"       | 45    |
+--------------------+-------+
| "Ray Allen"        | 43    |
+--------------------+-------+
| "Grant Hill"       | 46    |
+--------------------+-------+
| "Tony Parker"      | 36    |
+--------------------+-------+
...
```

```
nebula> GO FROM "player100" \
        OVER follow \
        WHERE follow.degree > 90;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
```

FILTER ON DYNAMICALLY-CALCULATED PROPERTIES

```
nebula> MATCH (v:player) \
        WHERE v[toLower("AGE")] < 21 \
        RETURN v.name, v.age;
+---------------+-------+
| v.name        | v.age |
+---------------+-------+
| "Luka Doncic" | 20    |
+---------------+-------+
```

FILTER ON EXISTING PROPERTIES

```
nebula> MATCH (v:player) \
        WHERE exists(v.age) \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Boris Diaw"            | 36    |
+-------------------------+-------+
| "DeAndre Jordan"        | 30    |
+-------------------------+-------+
```

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 test (vid_type=FIXED_STRING(30));
nebula> USE test;
nebula> CREATE EDGE e1(p1 int);
nebula> CREATE TAG 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"@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> GO FROM "1" \
        OVER e1 \
        WHERE e1._rank>2 \
        YIELD e1._src, e1._dst, e1._rank AS Rank, e1.p1 | \
        ORDER BY $-.Rank DESC;
==================================
| e1._src | e1._dst | Rank | e1.p1 |
==================================
| 1       | 2       | 6    | 16    |
----------------------------------
| 1       | 2       | 5    | 15    |
----------------------------------
| 1       | 2       | 4    | 14    |
----------------------------------
| 1       | 2       | 3    | 13    |
----------------------------------
```

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:player) \
        WHERE v.name STARTS WITH "T" \
        RETURN v.name, v.age;
+------------------+-------+
| v.name           | v.age |
+------------------+-------+
| "Tracy McGrady"  | 39    |
+------------------+-------+
| "Tony Parker"    | 36    |
+------------------+-------+
| "Tim Duncan"     | 42    |
+------------------+-------+
| "Tiago Splitter" | 34    |
+------------------+-------+
```

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.name STARTS WITH "t" \
        RETURN v.name, v.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.name ENDS WITH "r" \
        RETURN v.name, v.age;
+------------------+-------+
```

```
| v.name           | v.age |
+------------------+-------+
| "Vince Carter"   | 42    |
+------------------+-------+
| "Tony Parker"    | 36    |
+------------------+-------+
| "Tiago Splitter" | 34    |
+------------------+-------+
```

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:player) \
        WHERE v.name CONTAINS "Pa" \
        RETURN v.name, v.age;
+---------------+-------+
| v.name        | v.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.name ENDS WITH "R" \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Rajon Rondo"           | 33    |
+-------------------------+-------+
| "Rudy Gay"              | 32    |
+-------------------------+-------+
| "Dejounte Murray"       | 29    |
+-------------------------+-------+
| "Chris Paul"            | 33    |
+-------------------------+-------+
| "Carmelo Anthony"       | 34    |
+-------------------------+-------+
...
```

Filter on lists

MATCH VALUES IN A LIST

Use the `IN` operator to check if a value is in a specific list.

```
nebula> MATCH (v:player) \
        WHERE v.age IN range(20,25) \
        RETURN v.name, v.age;
+-------------------------+-------+
| v.name                  | v.age |
+-------------------------+-------+
| "Ben Simmons"           | 22    |
+-------------------------+-------+
| "Kristaps Porzingis"    | 23    |
+-------------------------+-------+
| "Luka Doncic"           | 20    |
+-------------------------+-------+
| "Kyle Anderson"         | 25    |
+-------------------------+-------+
| "Giannis Antetokounmpo" | 24    |
+-------------------------+-------+
| "Joel Embiid"           | 25    |
+-------------------------+-------+

nebula> LOOKUP ON player WHERE player.age IN [25,28]  YIELD player.name, player.age;
+-------------+-----------------+------------+
| VertexID    | player.name     | player.age |
+-------------+-----------------+------------+
| "player135" | "Damian Lillard" | 28         |
+-------------+-----------------+------------+
| "player131" | "Paul George"    | 28         |
+-------------+-----------------+------------+
```

```
| "player130" | "Joel Embiid"    | 25         |
+-------------+------------------+------------+
| "player123" | "Ricky Rubio"    | 28         |
+-------------+------------------+------------+
| "player106" | "Kyle Anderson"  | 25         |
+-------------+------------------+------------+
```

MATCH VALUES NOT IN A LIST

Use `NOT` before `IN` to rule out the values in a list.

```
nebula> MATCH (v:player) \
        WHERE v.age NOT IN range(20,25) \
        RETURN v.name AS Name, v.age AS Age \
        ORDER BY Age;
+--------------------+-----+
| Name               | Age |
+--------------------+-----+
| "Kyrie Irving"     | 26  |
+--------------------+-----+
| "Cory Joseph"      | 27  |
+--------------------+-----+
| "Damian Lillard"   | 28  |
+--------------------+-----+
| "Paul George"      | 28  |
+--------------------+-----+
| "Ricky Rubio"      | 28  |
+--------------------+-----+
...
```

Last update: August 30, 2021

4.7.7 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 `GO`, `FETCH`, or `LOOKUP`.
- 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.

  > **Note**
  >
  > NGQL does not support `CALL[…YIELD]` yet.

- In nGQL, `YIELD` works like `RETURN` in openCypher.

> **Note**
>
> In the following examples, `$$` and `$-` are reference operators. For more information, see Operators.

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, `col` 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`:

```
nebula> GO FROM "player100" OVER follow \
        YIELD $$.player.name AS Friend, $$.player.age AS Age;
+-----------------+-----+
| Friend          | Age |
+-----------------+-----+
| "Tony Parker"   | 36  |
+-----------------+-----+
```

```
| "Manu Ginobili" | 41  |
+-----------------+-----+
```

- Use `YIELD` with `FETCH`:

```
nebula> FETCH PROP ON player "player100" \
        YIELD player.name;
+-------------+--------------+
| VertexID    | player.name  |
+-------------+--------------+
| "player100" | "Tim Duncan" |
+-------------+--------------+
```

- Use `YIELD` with `LOOKUP`:

```
nebula> LOOKUP ON player WHERE player.name == "Tony Parker" \
        YIELD player.name, player.age;
======================================
| VertexID | player.name | player.age |
======================================
| 101      | Tony Parker | 36         |
--------------------------------------
```

YIELD statements

SYNTAX

```
YIELD [DISTINCT] <col> [AS <alias>] [, <col> [AS <alias>] ...]
[WHERE <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, `col` 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.
`conditions`	Conditions set in a `WHERE` clause to filter the output. For more information, see `WHERE`.

USE A YIELD STATEMENT IN A COMPOSITE QUERY

In a composite query, a `YIELD` 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> GO FROM "player100" OVER follow \
        YIELD follow._dst AS ID \
        | FETCH PROP ON player $-.ID \
        YIELD player.age AS Age \
        | YIELD AVG($-.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.

```
nebula> $var1 = GO FROM "player101" OVER follow \
        YIELD follow.degree AS Degree, follow._dst as ID; \
        YIELD $var1.ID AS ID WHERE $var1.Degree > 90;
+-------------+
| ID          |
+-------------+
| "player100" |
+-------------+
| "player125" |
+-------------+
```

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;
+-------------+------------+
| string1     | 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      |
+--------+
```

Last update: August 6, 2021

4.7.8 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.

```
nebula> MATCH p=(v:player{name:"Tim Duncan"})--() \
        WITH nodes(p) AS n \
        UNWIND n AS n1 \
        RETURN DISTINCT n1;
+----------------------------------------------------------------------+
| n1                                                                   |
+----------------------------------------------------------------------+
| ("player100" :star{} :person{} :player{age: 42, name: "Tim Duncan"}) |
+----------------------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"})                  |
+----------------------------------------------------------------------+
| ("team204" :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: 47, name: "Shaquile O'Neal"})              |
+----------------------------------------------------------------------+
| ("player105" :player{age: 31, name: "Danny Green"})                  |
+----------------------------------------------------------------------+
| ("player113" :player{age: 29, name: "Dejounte Murray"})              |
+----------------------------------------------------------------------+
| ("player107" :player{age: 32, name: "Aron Baynes"})                  |
+----------------------------------------------------------------------+
| ("player109" :player{age: 34, name: "Tiago Splitter"})               |
+----------------------------------------------------------------------+
| ("player108" :player{age: 36, name: "Boris Diaw"})                   |
+----------------------------------------------------------------------+
```

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   |
+----------+
| "star"   |
+----------+
| "player" |
+----------+
| "person" |
+----------+
```

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.age AS Age \
        ORDER BY Age \
        WHERE Age<25 \
        RETURN v2.name AS 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.

```
nebula> MATCH (v:player) \
        WITH v.name AS Name \
        ORDER BY Name DESC \
        LIMIT 3 \
        RETURN collect(Name);
+------------------------------------------------+
| collect(Name)                                  |
+------------------------------------------------+
| ["Yao Ming", "Vince Carter", "Tracy McGrady"] |
+------------------------------------------------+
```

Use with RETURN

Set an alias using a `WITH` clause, and then output the result through a `RETURN` clause.

```
nebula> WITH [1, 2, 3] AS list  RETURN 3 IN list AS r;
+------+
| r    |
+------+
| true |
+------+

nebula> WITH 4 AS one, 3 AS two RETURN one > two AS result;
+--------+
| result |
+--------+
| true   |
+--------+
```

Last update: August 6, 2021

4.8 Space statements

4.8.1 CREATE SPACE

Graph spaces are used to store data in a physically isolated way in Nebula Graph, which is similar to the database concept in MySQL. The `CREATE SPACE` statement creates a new graph space with the given name.

Prerequisites

Only the God role can use the `CREATE SPACE` statement. For more information, see AUTHENTICATION.

Syntax

```
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
`IF NOT EXISTS`	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>`	Uniquely identifies a graph space in a Nebula Graph instance. The name of the graph space is case-sensitive and allows letters, numbers, or underlines. Keywords and reserved words are not allowed.
`partition_num`	Specifies the number of partitions in each replica. The suggested number is five times the number of the hard disks in the cluster. For example, if you have 3 hard disks in the cluster, we recommend that you set 15 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` characters, Nebula Graph throws an error.
`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 Nebula Graph Storage Service with the BALANCE statement.

Caution

Restrictions on VID type change and VID length

1. In Nebula Graph 1.x, the VID type can only be `INT64` and does not support string. In Nebula Graph 2.x, the VID type can be both `INT64` and `FIXED_STRING(<N>)`. You should specify the VID type when creating a graph space and keep consistency when using the `INSERT` statement. Otherwise, Nebula Graph throws `Wrong vertex id type: 1001`.
2. The length of the VID should not be longer than `N` characters. If it exceeds `N`, Nebula Graph throws `The VID must be a 64-bit integer or a string fitting space vertex id length limit.`.

Legacy version compatibility

In the 2.x releases before 2.5.0, `vid_type` is not a required parameter and its default value is `FIXED_STRING(8)`.

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`.

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 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 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 my_space_3 (partition_num=15, replica_factor=1, vid_type=FIXED_STRING(30)) comment="Test the graph space";
```

Implementation of the operation

Caution

Trying to use a newly created graph space may fail because the creation is implemented asynchronously.

Nebula Graph implements the creation in the next heartbeat cycle. To make sure the creation is successful, take one of the following approaches:

- Find the new graph space in the result of `SHOW SPACES` or `DESCRIBE SPACE`. If you cannot, wait a few seconds and try again.

- 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 peers 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       |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "storaged0" | 9779 | "ONLINE" | 8            | "basketballplayer:3, test:5" | "basketballplayer:10, test:10" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "storaged1" | 9779 | "ONLINE" | 9            | "basketballplayer:4, test:5" | "basketballplayer:10, test:10" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"         | "basketballplayer:10, test:10" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "Total"     |      |          | 20           | "basketballplayer:10, test:10" | "basketballplayer:30, test:30" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
```

To balance the request loads, use the following command.

```
nebula> BALANCE LEADER;
+-------------+------+----------+--------------+------------------------------+------------------------------+
| Host        | Port | Status   | Leader count | Leader distribution          | Partition distribution       |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "storaged0" | 9779 | "ONLINE" | 7            | "basketballplayer:3, test:4" | "basketballplayer:10, test:10" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "storaged1" | 9779 | "ONLINE" | 7            | "basketballplayer:4, test:3" | "basketballplayer:10, test:10" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
| "storaged2" | 9779 | "ONLINE" | 6            | "basketballplayer:3, test:3" | "basketballplayer:10, test:10" |
+-------------+------+----------+--------------+------------------------------+------------------------------+
```

```
| "Total"     |      |          | 20           | "basketballplayer:10, test:10" | "basketballplayer:30, test:30" |
+-------------+------+----------+--------------+--------------------------------+--------------------------------+
```

Last update: August 30, 2021

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, Nebula Graph throws an error.

Syntax

```
USE <graph_space_name>;
```

Examples

```
# The following example specifies space1 as the current working graph space.
nebula> USE space1;

# The following example traverses in space1.
nebula> GO FROM 1 OVER edge1;

# The following example specifies space2 as the current working graph space.
nebula> USE space2;

# The following example traverses in space2. Hereafter, you cannot read any data from space1, because these vertices and edges being traversed have no relevance with
space1.
nebula> GO FROM 2 OVER edge2;
```

Caution

You cannot use two graph spaces in one statement.

Different from Fabric Cypher, graph spaces in Nebula Graph 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 Nebula Graph, you can only use one graph space in one statement.

Last update: August 13, 2021

4.8.3 SHOW SPACES

`SHOW SPACES` lists all the graph spaces in the Nebula Graph examples.

Syntax

```
SHOW SPACES;
```

Example

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

To create graph spaces, see CREATE SPACE.

Last update: August 13, 2021

4.8.4 DESCRIBE SPACE

`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 basketballplayer;
+----+--------------------+------------------+----------------+---------+-----------+--------------------+-------------+-----------+---------+
| ID | Name               | Partition Number | Replica Factor | Charset | Collate   | Vid Type           | Atomic Edge | Group     | Comment |
+----+--------------------+------------------+----------------+---------+-----------+--------------------+-------------+-----------+---------+
| 1  | "basketballplayer" | 10               | 1              | "utf8"  | "utf8_bin" | "FIXED_STRING(32)" | false       | "default" |         |
+----+--------------------+------------------+----------------+---------+-----------+--------------------+-------------+-----------+---------+
```

Last update: August 30, 2021

4.8.5 DROP SPACE

`DROP SPACE` deletes everything in the specified graph space.

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.

The `DROP SPACE` statement does not immediately remove all the files and directories from the disk. You can specify another graph space with the `USE` statement and `submit job compact`.

> **Caution**
>
> **BE CAUTIOUS** about running the `DROP SPACE` statement.

Last update: August 13, 2021

4.9 Tag statements

4.9.1 CREATE TAG

`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, Nebula Graph 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>`	The tag name must be **unique** in a graph space. Once the tag name is set, it can not be altered. The rules for permitted tag names are the same as those for graph space names. For prohibited names, see Keywords and reserved words.
`<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>`	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 | 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 Nebula Graph. 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 player(name string, age int);

# The following example creates a tag with no properties.
nebula> CREATE TAG no_property();

# The following example creates a tag with a default value.
nebula> CREATE TAG 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 woman(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.

Nebula Graph implements the creation of the tag in the next heartbeat cycle. To make sure the creation is successful, take one of the following approaches:

- Find the new tag in the result of `SHOW TAGS`. If you cannot, wait a few seconds and try again.
- 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: August 13, 2021

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. But its edges are available. The vertex will be deleted in the next compaction.
- 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.

Prerequisites

- Running the `DROP TAG` statement requires some privileges for the graph space. Otherwise, Nebula Graph throws an error.
- Before you drop a tag, make sure that the tag does not have any indexes. Otherwise, the conflict error (`[ERROR (-8)]: 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 test(p1 string, p2 int);
nebula> DROP TAG test;
```

> **Note**
>
> In nGQL, there is no such statement to drop a certain tag of a vertex with the given name.
>
> - In openCypher, you can use the statement `REMOVE v:LABEL` to drop the tag `LABLE` of the vertex `v`.
> - In nGQL, after `CREATE TAG` and `INSERT VERTEX`, you can add a `TAG` on the vertex. But there is no way to drop the `TAG` afterward.
>
> We recommend you to add a field to identify the logical deletion in the schema. For example, add `removed` to the schema of each tag.

Last update: September 2, 2021

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.

Prerequisites

- Running the `ALTER TAG` statement requires some privileges for the graph space. Otherwise, Nebula Graph 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 (-8)]: Conflict!` will occur when you `ALTER TAG`. For more information on dropping an index, see DROP INDEX.

Syntax

```
ALTER TAG <tag_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]
    [COMMENT = '<comment>'];

alter_definition:
| ADD    (prop_name data_type)
| DROP   (prop_name)
| CHANGE (prop_name data_type)

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.

Examples

```
nebula> CREATE TAG t1 (p1 string, p2 int);
nebula> ALTER TAG t1 ADD (p3 int, p4 string);
nebula> ALTER TAG t1 TTL_DURATION = 2, TTL_COL = "p2";
nebula> ALTER TAG t1 COMMENT = 'test1';
```

Implementation of the operation

Trying to use a newly altered tag may fail because the alteration of the tag is implemented asynchronously.

Nebula Graph implements the alteration of the tag in the next heartbeat cycle. To make sure the alteration is successful, take one of the following approaches:

- Use `DESCRIBE TAG` to confirm that the tag information is updated. If it is not, wait a few seconds and try again.
- 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: August 30, 2021

4.9.4 SHOW TAGS

The `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     |
+----------+
| "player" |
+----------+
| "team"   |
+----------+
```

Last update: August 13, 2021

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, Nebula Graph 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" |         |         |
+--------+----------+-------+---------+---------+
```

Last update: August 13, 2021

4.9.6 DELETE TAG

`DELETE TAG` deletes a tag with the given name on a specified vertex.

> **Enterpriseonly**
>
> This feature is only available in the Enterprise Edition.

A vertex can have one or more tags.

- If a vertex has only one tag, the vertex **CANNOT** be accessed after you delete the tag. But its edges are available. The vertex will be deleted in the next compaction.
- If a vertex has multiple tags, the vertex is still accessible after you delete one of them. But all the properties defined by this deleted tag **CANNOT** be accessed.

Prerequisites

Running the `DELETE TAG` statement requires some privileges for the graph space. Otherwise, Nebula Graph throws an error.

Syntax

```
DELETE TAG <tag_name_list> FROM <VID>;
```

- `tag_name_list` : Specifies the name of the tag. Multiple tags are separated with commas (,). `*` means all tags.
- `VID` : Specifies the VID of the tag to delete.

Example

```
nebula> CREATE TAG test1(p1 string, p2 int);
nebula> CREATE TAG test2(p3 string, p4 int);
nebula> INSERT VERTEX test1(p1, p2),test2(p3, p4) VALUES "test":("123", 1, "456", 2);
nebula> FETCH PROP ON * "test";
+------------------------------------------------------------+
| vertices_                                                  |
+------------------------------------------------------------+
| ("test" :test2{p3: "456", p4: 2} :test1{p1: "123", p2: 1}) |
+------------------------------------------------------------+

nebula> DELETE TAG test1 FROM "test";
nebula> FETCH PROP ON * "test";
+----------------------------------+
| vertices_                        |
+----------------------------------+
| ("test" :test2{p3: "456", p4: 2}) |
+----------------------------------+

nebula> DELETE TAG * FROM "test";
nebula> FETCH PROP ON * "test";
+-----------+
| vertices_ |
+-----------+
+-----------+
```

> **Compatibility**
>
> - In openCypher, you can use the statement `REMOVE v:LABEL` to delete the tag `LABLE` of the vertex `v`.
> - `DELETE TAG` and `DROP TAG` have the same semantics but different syntax. In nGQL, use `DELETE TAG`.

Last update: September 10, 2021

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, Nebula Graph 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>
    (
      <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 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>`	The edge type name must be **unique** in a graph space. Once the edge type name is set, it can not be altered. The rules for permitted edge type names are the same as those for graph space names. For prohibited names, see Keywords and reserved words.
`<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>`	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 | 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 Nebula Graph. 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 `TTL_COL`. For more information on TTL, see TTL options.

EXAMPLES

```
nebula> CREATE EDGE follow(degree int);

# The following example creates an edge type with no properties.
nebula> CREATE EDGE no_property();

# The following example creates an edge type with a default value.
nebula> CREATE EDGE 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 e1(p1 string, p2 int, p3 timestamp) \
        TTL_DURATION = 100, TTL_COL = "p2";
```

Implementation of the operation

Trying to use a newly created edge type may fail because the creation of the edge type is implemented asynchronously.

Nebula Graph implements the creation of the edge type in the next heartbeat cycle. To make sure the creation is successful, take the following approaches:

- Find the new edge type in the result of `SHOW EDGES`. If you cannot, wait a few seconds and try again.
- 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: August 13, 2021

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, Nebula Graph 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 (-8)]: Conflict!`) will be returned. To drop an index, see DROP INDEX.

Syntax

```
DROP EDGE [IF EXISTS] <edge_type_name>
```

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 e1(p1 string, p2 int);
nebula> DROP EDGE e1;
```

Last update: August 13, 2021

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.

Prerequisites

- Running the `ALTER EDGE` statement requires some privileges for the graph space. Otherwise, Nebula Graph 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 (-8)]: Conflict!` will occur when you `ALTER EDGE`. For more information on dropping an index, see DROP INDEX.

Syntax

```
ALTER EDGE <edge_type_name>
    <alter_definition> [, alter_definition] ...]
    [ttl_definition [, ttl_definition] ... ]
    [COMMENT = '<comment>'];

alter_definition:
| ADD    (prop_name data_type)
| DROP   (prop_name)
| CHANGE (prop_name data_type)

ttl_definition:
    TTL_DURATION = ttl_duration, TTL_COL = prop_name
```

- `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.

Example

```
nebula> CREATE EDGE 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.

Nebula Graph implements the alteration of the edge type in the next heartbeat cycle. To make sure the alteration is successful, take one of the following approaches:

- Use `DESCRIBE EDGE` to confirm that the edge type information is updated. If it is not, wait a few seconds and try again.
- 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: September 2, 2021

4.10.4 SHOW EDGES

`SHOW EDGES` 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"  |
+----------+
```

Last update: August 13, 2021

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, Nebula Graph 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" |         |         |
+----------+---------+-------+---------+---------+
```

Last update: August 13, 2021

4.11 Vertex statements

4.11.1 INSERT VERTEX

The `INSERT VERTEX` statement inserts one or more vertices into a graph space in Nebula Graph.

Prerequisites

Running the `INSERT VERTEX` statement requires some privileges for the graph space. Otherwise, Nebula Graph throws an error.

Syntax

```
INSERT VERTEX [IF NOT EXISTS] <tag_name> (<prop_name_list>) [, <tag_name> (<prop_name_list>), ...]
     {VALUES | VALUE} VID: (<prop_value_list>[, <prop_value_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.

  > **Note**
  >
  > - `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.
- `prop_name_list` contains the names of the properties on the tag.
- `VID` is the vertex ID. In Nebula Graph 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`. If the property values do not match the data type in the tag, 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 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

```
# The following examples create tag t1 with no property and inserts vertex "10" with no property.
nebula> CREATE TAG t1();
nebula> INSERT VERTEX t1() VALUE "10":();
```

```
nebula> CREATE TAG 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 t3(p1 int);
nebula> CREATE TAG 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.

```
# The following examples insert vertex "11" with new values for multiple times.
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n2", 13);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n3", 14);
nebula> INSERT VERTEX t2 (name, age) VALUES "11":("n4", 15);
nebula> FETCH PROP ON t2 "11";
+---------------------------------+
| vertices_                       |
+---------------------------------+
| ("11" :t2{age: 15, name: "n4"}) |
+---------------------------------+
```

```
nebula> CREATE TAG t5(p1 fixed_string(5) NOT NULL, p2 int, p3 int DEFAULT NULL);
nebula> INSERT VERTEX t5(p1, p2, p3) VALUES "001":("Abe", 2, 3);

# 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 (-8)]: Storage Error: The not null field cannot be null.

# In the following example, the value of p3 is the default NULL.
nebula> INSERT VERTEX t5(p1, p2) VALUES "003":("cd", 5);
nebula> FETCH PROP ON t5 "003";
+-------------------------------------------+
| vertices_                                 |
+-------------------------------------------+
| ("003" :t5{p1: "cd", p2: 5, p3: __NULL__}) |
+-------------------------------------------+

# In the following example, the allowed maximum length of p1 is 5.
nebula> INSERT VERTEX t5(p1, p2) VALUES "004":("shalalalala", 4);
nebula> FETCH PROP on t5 "004";
+-----------------------------------------------+
| vertices_                                     |
+-----------------------------------------------+
| ("004" :t5{p1: "shala", p2: 4, p3: __NULL__}) |
+-----------------------------------------------+
```

If you insert a vertex that already exists with `IF NOT EXISTS`, there will be no modification.

```
# 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";
+---------------------------------+
| vertices_                       |
+---------------------------------+
| ("1" :t2{age: 13, name: "n2"}) |
+---------------------------------+
```

Last update: September 2, 2021

4.11.2 DELETE VERTEX

The `DELETE VERTEX` statement deletes vertices and the related incoming and outgoing edges of the vertices.

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.

Syntax

```
DELETE VERTEX <vid> [, <vid> ...];
```

Examples

This query deletes the vertex whose ID is "team1".

```
nebula> DELETE VERTEX "team1";
```

This query shows that you can use `DELETE VERTEX` together with pipe to delete vertices.

```
nebula> GO FROM "player100" OVER serve WHERE serve.start_year == "2021" YIELD serve._dst AS id | DELETE VERTEX $-.id;
```

Delete the process and the related edges

Nebula Graph traverses the incoming and outgoing edges related to the vertices and deletes them all. Then Nebula Graph 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: August 17, 2021

4.11.3 UPDATE VERTEX

The `UPDATE VERTEX` statement updates properties on tags of a vertex.

In Nebula Graph, `UPDATE VERTEX` supports compare-and-set (CAS).

> **Note**
>
> An `UPDATE VERTEX` statement can only update properties on **ONE 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>`	Yes	Specifies the tag of the vertex. The properties to be updated must be on this tag.	`ON player`
`<vid>`	Yes	Specifies the ID of the vertex to be updated.	`"player100"`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET age = age +1`
`WHEN <condition>`	No	Specifies the filter conditions. If `<condition>` evaluates to `false`, the `SET` clause will not take effect.	`WHEN name == "Tim"`
`YIELD <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";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"}) |
+-------------------------------------------------+

// This query updates the age property and returns name and the new age.
nebula> UPDATE VERTEX ON player "player101" \
        SET age = age + 2 \
        WHEN name == "Tony Parker" \
        YIELD name AS Name, age AS Age;
+---------------+-----+
| Name          | Age |
+---------------+-----+
| "Tony Parker" | 38  |
+---------------+-----+
```

Last update: August 17, 2021

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.

> **Note**
>
> An `UPSERT VERTEX` statement can only update the properties on **ONE TAG** of a vertex.

The performance of `UPSERT` is much lower than that of `INSERT` because `UPSERT` is a read-modify-write serialization operation at the partition level.

> **Danger**
>
> 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>`	Yes	Specifies the tag of the vertex. The properties to be updated must be on this tag.	`ON player`
`<vid>`	Yes	Specifies the ID of the vertex to be updated or inserted.	`"player100"`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET age = age +1`
`WHEN <condition>`	No	Specifies the filter conditions.	`WHEN name == "Tim"`
`YIELD <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";
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" \
        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 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 `WHEN` conditions are met, the vertex is updated.

```
nebula> FETCH PROP ON player "player101";
+-------------------------------------------------+
| vertices_                                       |
+-------------------------------------------------+
| ("player101" :player{age: 42, 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" | 44  |
+---------------+-----+
```

If the vertex exists and the `WHEN` conditions are not met, the update does not take effect.

```
nebula> FETCH PROP ON player "player101";
+-------------------------------------------------------+
| vertices_                                             |
+-------------------------------------------------------+
| ("player101" :player{age: 44, 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" | 44  |
+---------------+-----+
```

Last update: August 17, 2021

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 Nebula Graph.

When inserting an edge that already exists, `INSERT VERTEX` **overrides** the edge.

Syntax

```
INSERT EDGE [IF NOT EXISTS] <edge_type> ( <prop_name_list> ) {VALUES | VALUE}
<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> ] ...]
```

- `IF NOT EXISTS` detects if the edge that you want to insert exists. If it does not exist, a new one will be inserted.

  > **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.

- `<edge_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. 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.

  > **OpenCypher 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 e1 with no property and inserts an edge from vertex "10" to vertex "11" with no property.
nebula> CREATE EDGE 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 e2 (name string, age int);
nebula> INSERT EDGE e2 (name, age) VALUES "11"->"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";
+-----------------------------------------+
| edges_                                  |
+-----------------------------------------+
| [: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.

```
# The following example inserts edge e2 from vertex "14" to vertex "15".
nebula> INSERT EDGE e2 (name, age) VALUES "14"->"15"@1:("n1", 12);
# The following example alters the edge with IF NOT EXISTS. But there will be no alteration because edge e2 already exists.
nebula> INSERT EDGE IF NOT EXISTS e2 (name, age) VALUES "14"->"15"@1:("n2", 13);
nebula> FETCH PROP ON e2 "14"->"15"@1;
+-----------------------------------------+
| edges_                                  |
+-----------------------------------------+
| [:e2 "14"->"15" @1 {age: 12, name: "n1"}] |
+-----------------------------------------+
```

Note

- Nebula Graph 2.5.0 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.
- 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: September 2, 2021

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.

> **Note**
>
> Atomic operation is not guaranteed during the entire process for now, so please retry when a failure occurs.

Syntax

```
DELETE EDGE <edge_type> <src_vid> -> <dst_vid>[@<rank>] [, <src_vid> -> <dst_vid>[@<rank>] ...]
```

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> GO FROM "player100" OVER follow \
        WHERE follow._dst == "team204" \
        YIELD follow._src AS src, follow._dst AS dst, follow._rank AS rank \
        | DELETE EDGE follow $-.src->$-.dst @ $-.rank;
```

Last update: August 17, 2021

4.12.3 UPDATE EDGE

The `UPDATE EDGE` statement updates properties on an edge.

In Nebula Graph, `UPDATE EDGE` supports compare-and-set (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>`	Yes	Specifies the edge type. The properties to be updated must be on this edge type.	`ON serve`
`<src_vid>`	Yes	Specifies the source vertex ID of the edge.	`"player100"`
`<dst_vid>`	Yes	Specifies the destination vertex ID of the edge.	`"team204"`
`<rank>`	No	Specifies the rank of the edge.	`10`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET start_year = start_year +1`
`WHEN <condition>`	No	Specifies the filter conditions. If `<condition>` evaluates to `false`, the `SET` clause does not take effect.	`WHEN end_year < 2010`
`YIELD <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.

```
nebula> GO FROM "player100" \
        OVER serve \
        YIELD serve.start_year, serve.end_year;
+------------------+----------------+
| serve.start_year | serve.end_year |
+------------------+----------------+
| 1997             | 2016           |
+------------------+----------------+
```

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     |
+------------+----------+
```

Last update: August 17, 2021

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.

> **Danger**
>
> Do not use `UPSERT` for scenarios with highly concurrent writes. You can use `UPDATE` or `INSERT` instead.

Syntax

```
UPSERT EDGE ON <edge_type>
<src_vid> -> <dst_vid> [@rank]
SET <update_prop>
[WHEN <condition>]
[YIELD <properties>]
```

Parameter	Required	Description	Example
`ON <edge_type>`	Yes	Specifies the edge type. The properties to be updated must be on this edge type.	`ON serve`
`<src_vid>`	Yes	Specifies the source vertex ID of the edge.	`"player100"`
`<dst_vid>`	Yes	Specifies the destination vertex ID of the edge.	`"team204"`
`<rank>`	No	Specifies the rank of the edge.	`10`
`SET <update_prop>`	Yes	Specifies the properties to be updated and how they will be updated.	`SET start_year = start_year +1`
`WHEN <condition>`	No	Specifies the filter conditions.	`WHEN end_year < 2010`
`YIELD <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:

```
// This example checks if the following three vertices have any outgoing serve edge. The result "Empty set" indicates that such an edge does not exist.
nebula> GO FROM "player666", "player667", "player668" \
        OVER serve \
        YIELD serve.start_year, serve.end_year;
Empty set

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2021 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player666" -> "team200"@0 \
        SET end_year = 2022 \
        WHEN end_year == 2010 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2021     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player667" -> "team200"@0 \
        SET end_year = 2022 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| __NULL__   | 2022     |
+------------+----------+

nebula> UPSERT EDGE on serve \
        "player668" -> "team200"@0 \
        SET start_year = 2000, end_year = end_year + 1 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2000       | __NULL__ |
+------------+----------+
```

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 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.

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2019, start_year: 2016}] |
+------------------------------------------------------------------------+

nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year == 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

If the edge exists and the `WHEN` conditions are not met, the update does not take effect.

```
nebula> MATCH (v:player{name:"Ben Simmons"})-[e:serve]-(v2) \
        RETURN e;
+------------------------------------------------------------------------+
| e                                                                      |
+------------------------------------------------------------------------+
| [:serve "player149"->"team219" @0 {end_year: 2020, start_year: 2016}] |
+------------------------------------------------------------------------+


nebula> UPSERT EDGE on serve \
        "player149" -> "team219" \
        SET end_year = end_year + 1 \
        WHEN start_year != 2016 \
        YIELD start_year, end_year;
+------------+----------+
| start_year | end_year |
+------------+----------+
| 2016       | 2020     |
+------------+----------+
```

Last update: August 30, 2021

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.

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.
- You can do partial match searches by using composite indexes. The declared fields in the composite index are used from left to right. For more information, see LOOKUP FAQ.
- String operators like `CONTAINS` and `STARTS WITH` are not allowed in LOOKUP for native index searching. Use full-text indexes to do fuzzy searches.

OPERATIONS ON NATIVE INDEXES

- CREATE INDEX
- SHOW CREATE INDEX
- SHOW INDEXES
- DESCRIBE INDEX
- REBUILD INDEX
- SHOW INDEX STATUS
- DROP INDEX
- LOOKUP
- MATCH

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.
- Only strings within a specified length (no longer than 256 bytes) are indexed.
- Full-text indexes do not support logical operations such as `AND`, `OR`, and `NOT`.

> **Note**
>
> To do complete string matches, use native indexes.

OPERATIONS ON FULL-TEXT INDEXES

Before doing any operations on full-text indexes, please make sure that you deploy full-text indexes. Details on full-text indexes deployment, see Deploy Elasticsearch and Deploy Listener.

At this time, full-text indexes are created automatically on the Elasticsearch cluster. And rebuilding or altering full-text indexes are not supported. To drop full-text indexes, you need to drop them on the Elasticsearch cluster manually.

To query full-text indexes, see Search with full-text 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: August 18, 2021

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. You can use it together with `LOOKUP` and `MATCH` statements.

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`, 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` statement, `i_T` cannot replace `i_TA` for querying properties.
- In the `LOOKUP` statement, `i_T` may replace `i_TA` for querying properties.

> **Legacy 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 reduction can be as much as 90% or even more. **DO NOT** use indexes in production environments unless you are fully aware of their influences on your service.
>
> Indexes cannot make queries faster. It can only locate a vertex or an edge according to properties or count the number of vertices or edges.
>
> 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. The longest index length is 255 bytes. Strings longer than 255 bytes will be truncated.

If you must use indexes, we suggest that you:

1. Import the data into Nebula Graph.
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, Nebula Graph 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. Nebula Graph implements the creation in the next heartbeat cycle. To make sure the creation is successful, take one of the following approaches:

- Find the new index in the result of `SHOW TAG/EDGE INDEXES`.
- 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> | <edge_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>`	The name of the index. It must be unique in a graph space. A recommended way of naming is `i_tagName_propName`. The name of the index is case-sensitive and allows letters, numbers, or underlines. Keywords and reserved words are not allowed.
`<tag_name> | <edge_name>`	Specifies the name of the tag or edge associated with the index.
`<prop_name_list>`	To index a **variable-length** string property, you must use `prop_name(length)` to specify the index length. 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 player_index on player();
```

```
nebula> CREATE EDGE INDEX 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 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 var_string(p1 string);
nebula> CREATE TAG INDEX var ON var_string(p1(10));

# To index a fixed-length string property, you do not need to specify the index length.
nebula> CREATE TAG fix_string(p1 FIXED_STRING(10));
nebula> CREATE TAG INDEX fix ON fix_string(p1);
```

```
nebula> CREATE EDGE INDEX 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 player_index_1 on player(name(10), age);
```

Caution

Creating composite property indexes across multiple tags or edge types is not supported.

Nebula Graph follows the left matching principle to select indexes when composite property indexes are used in `LOOKUP` or `MATCH` statements. That is, columns in the `WHERE` conditions must be in the first N columns of the index. For example:

```
# This example creates a composite property index for the first 3 properties of tag t.
nebula> CREATE TAG INDEX example_index ON t(p1, p2, p3);

# Note: The index match is not successful because it does not start from p1.
nebula> LOOKUP ON t WHERE p2 == 1 and p3 == 1;

# The index match is successful.
nebula> LOOKUP ON t WHERE p1 == 1;
# The index match is successful because p1 and p2 are consecutive.
nebula> LOOKUP ON t WHERE p1 == 1 and p2 == 1;
# The index match is successful because p1, p2, and p3 are consecutive.
nebula> LOOKUP ON t WHERE p1 == 1 and p2 == 1 and p3 == 1;
```

Last update: August 30, 2021

4.13.3 SHOW INDEXES

`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"            | "fix_string" | ["p1"]          |
+------------------+--------------+-----------------+
| "player_index_0" | "player"     | ["name"]        |
+------------------+--------------+-----------------+
| "player_index_1" | "player"     | ["name", "age"] |
+------------------+--------------+-----------------+
| "var"            | "var_string" | ["p1"]          |
+------------------+--------------+-----------------+

nebula> SHOW EDGE INDEXES;
+---------------+----------+---------+
| Index Name    | By Edge  | Columns |
+---------------+----------+---------+
| "follow_index" | "follow" | []      |
+---------------+----------+---------+
```

Legacy version compatibility

In Nebula Graph 2.0.1, the `SHOW TAG/EDGE INDEXES` statement only returns `Names`.

Last update: August 18, 2021

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 INDEXES;
+------------------+-------------+-----------------+
| "player_index_0" | "player"    | ["name"]        |
+------------------+-------------+-----------------+
| "player_index_1" | "player"    | ["name", "age"] |
+------------------+-------------+-----------------+

nebula> SHOW CREATE TAG INDEX player_index_1;
+------------------+-------------------------------------------------+
| Tag Index Name   | Create Tag Index                                |
+------------------+-------------------------------------------------+
| "player_index_1" | "CREATE TAG INDEX `player_index_1` ON `player` ( |
|                  |  `name(20)`                                     |
|                  | )"                                              |
+------------------+-------------------------------------------------+
```

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` ( |
|                 | )"                                              |
+-----------------+-------------------------------------------------+
```

> **Legacy version compatibility**
>
> In Nebula Graph 2.0.1, the `SHOW TAG/EDGE INDEXES` statement only returns `Names`.

Last update: August 18, 2021

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> DESCRIBE TAG INDEX player_index_0;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(30)" |
+--------+--------------------+

nebula> DESCRIBE TAG INDEX player_index_1;
+--------+--------------------+
| Field  | Type               |
+--------+--------------------+
| "name" | "fixed_string(10)" |
+--------+--------------------+
| "age"  | "int64"            |
+--------+--------------------+
```

Last update: August 18, 2021

4.13.6 REBUILD INDEX

Danger

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.

During the rebuilding, all queries skip the index and perform sequential scans. This means that the return results can be different because not all the data is indexed during rebuilding.

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.

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 person(name string, age int, gender string, email string);
nebula> CREATE TAG INDEX single_person_index ON person(name(10));

# The following example rebuilds an index and returns the job ID.
nebula> REBUILD TAG INDEX single_person_index;
+------------+
| New Job Id |
+------------+
| 31         |
+------------+

# The following example checks the index status.
nebula> SHOW TAG INDEX STATUS;
+-----------------------+--------------+
| Name                  | Index Status |
+-----------------------+--------------+
| "single_person_index" | "FINISHED"   |
+-----------------------+--------------+

# You can also use "SHOW JOB <job_id>" to check if the rebuilding process is complete.
nebula> SHOW JOB 31;
+---------------+--------------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest)       | Status     | Start Time              | Stop Time               |
+---------------+--------------------+------------+-------------------------+-------------------------+
| 31            | "REBUILD_TAG_INDEX" | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:24.000 |
+---------------+--------------------+------------+-------------------------+-------------------------+
| 0             | "storaged1"         | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:28.000 |
+---------------+--------------------+------------+-------------------------+-------------------------+
| 1             | "storaged2"         | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:28.000 |
+---------------+--------------------+------------+-------------------------+-------------------------+
| 2             | "storaged0"         | "FINISHED" | 2021-07-07T09:04:24.000 | 2021-07-07T09:04:28.000 |
+---------------+--------------------+------------+-------------------------+-------------------------+
```

Nebula Graph 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.

Legacy version compatibility

In Nebula Graph 2.x, the `OFFLINE` option is no longer needed or supported.

Last update: August 18, 2021

4.13.7 SHOW INDEX STATUS

`SHOW INDEX STATUS` returns the name of the created tag or edge type index and its status.

The index status 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 INDEX STATUS;
+----------------------+--------------+
| Name                 | Index Status |
+----------------------+--------------+
| "player_index_0"     | "FINISHED"   |
+----------------------+--------------+
| "player_index_1"     | "FINISHED"   |
+----------------------+--------------+
```

Last update: August 18, 2021

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, Nebula Graph throws an error.

Syntax

```
DROP {TAG | EDGE} INDEX [IF EXISTS] <index_name>;
```

`IF NOT 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;
```

Last update: August 18, 2021

4.14 Full-text index statements

4.14.1 Full-text index restrictions

> **Caution**
>
> This topic introduces the restrictions for full-text indexes. Please read the restrictions very carefully before using the full-text indexes.

For now, full-text search has the following limitations:

1. The maximum indexing string length is 256 bytes. The part of data that exceeds 256 bytes will not be indexed.
2. If there is a full-text index on the tag/edge type, the tag/edge type cannot be deleted or modified.
3. One tag/edge type can only have one full-text index.
4. The type of properties must be `string`.
5. The `WHERE` clause in the full-text search statement `LOOKUP`/`MATCH` does not support logical expressions `AND` and `OR`.
6. Full-text index can not be applied to search multiple tags/edge types.
7. Sorting for the returned results of the full-text search is not supported. Data is returned in the order of data insertion.
8. Full-text index can not search properties with value `NULL`.
9. Altering Elasticsearch indexes is not supported at this time.
10. The pipe operator is not supported in the `LOOKUP` and `MATCH` statements, excluding the examples in our manual.
11. Full-text search only works on single terms.
12. Full-text indexes are not deleted together with the graph space.
13. 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.
14. Do not contain `'` or `\` in the vertex or edge values. If not, an error will be caused in the Elasticsearch cluster storage.
15. It may take a while for Elasticsearch to create indexes. If Nebula Graph warns no index is found, wait for the index to take effect (however, the waiting time is unknown and there is no code to check).

Last update: August 31, 2021

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.

When the Elasticsearch cluster is started, add the template file for the Nebula Graph full-text index. For more information on index templates, see Elasticsearch Document.

Take the following sample template for example:

```
{
 "template": "nebula*",
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "mappings": {
    "properties" : {
            "tag_id" : { "type" : "long" },
            "column_id" : { "type" : "text" },
            "value" :{ "type" : "keyword"}
        }
  }
}
```

Make sure that you specify the following fields in strict accordance with the preceding template format:

```
"template": "nebula*"
"tag_id" : { "type" : "long" },
"column_id" : { "type" : "text" },
"value" :{ "type" : "keyword"}
```

Caution

When creating a full-text index, start the index name with `nebula`.

For example:

```
curl -H "Content-Type: application/json; charset=utf-8" -XPUT http://127.0.0.1:9200/_template/nebula_index_template -d '
{
 "template": "nebula*",
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1
    }
  },
  "mappings": {
    "properties" : {
            "tag_id" : { "type" : "long" },
            "column_id" : { "type" : "text" },
            "value" :{ "type" : "keyword"}
        }
  }
}'
```

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> [,<username>, <password>]), (<elastic_ip:port>), ...];
```

EXAMPLE

```
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);
```

> **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.

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;
+-------------+------+
| Host        | Port |
+-------------+------+
| "127.0.0.1" | 9200 |
+-------------+------+
| "127.0.0.1" | 9200 |
+-------------+------+
| "127.0.0.1" | 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: August 31, 2021

4.14.3 Deploy Raft Listener for Nebula 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 Nebula Graph cluster.
- You have deploy a Elasticsearch cluster.
- You have prepared at least one extra Storage Server. To use the full-text search, you must run one or more Storage Server as the Raft Listener.

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 STORAGE SERVICE

The Listener process and the storaged process use the same binary file. However, their configuration files and using ports are different. You can install Nebula Graph on all servers that need to deploy a Listener, but only the Storage service can be used. For details, see Install Nebula Graph by RPM or DEB Package.

STEP 2: PREPARE THE CONFIGURATION FILE FOR THE LISTENER

You have to prepare a corresponding configuration file on the machine that you want to deploy a Listener. The file must be named as `nebula-storaged-listener.conf` and stored in the `etc` directory. A template is provided for your reference. Note that the file suffix `.production` should be removed.

Most configurations are the same as the configurations of Storage Service. This topic only introduces the differences.

Name	Default value	Description
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids_listener/nebula-storaged.pid`	The file that records the process ID.
`meta_server_addrs`	-	IP addresses and ports of all Meta services. Multiple Meta services are separated by commas.
`local_ip`	-	The local IP address of the Listener service.
`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 `data`.
`part_man_type`	`memory`	The type of the part manager. Optional values are `memory` and `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.

Note

Use real IP addresses in the configuration file instead of domain names or loopback IP addresses such as `127.0.0.1`.

STEP 3: START LISTENERS

Run the following command to start the Listener.

```
./bin/nebula-storaged --flagfile <listener_config_path>/nebula-storaged-listener.conf
```

`${listener_config_path}` is the path where you store the Listener configuration file.

STEP 4: ADD LISTENERS TO NEBULA GRAPH

Connect to Nebula Graph 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 Nebula Graph.

```
ADD LISTENER ELASTICSEARCH <listener_ip:port> [,<listener_ip:port>, ...]
```

Warning

You must use real IPs for a Listener.

Add all Listeners in one statement completely.

```
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:9789,192.168.8.6:9789;
```

Show Listeners

Run the `SHOW LISTENER` statement to list all Listeners.

EXAMPLE

```
nebula> SHOW LISTENER;
+--------+-----------------+-----------------------+----------+
| PartId | Type            | Host                  | Status   |
+--------+-----------------+-----------------------+----------+
| 1      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+-----------------------+----------+
| 2      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+-----------------------+----------+
| 3      | "ELASTICSEARCH" | "[192.168.8.5:46780]" | "ONLINE" |
+--------+-----------------+-----------------------+----------+
```

Remove Listeners

Run the `REMOVE LISTENER ELASTICSEARCH` statement to remove all Listeners in a graph space.

EXAMPLE

```
nebula> REMOVE LISTENER ELASTICSEARCH;
```

Danger

After the Listener is deleted, it cannot be added again. Therefore, the synchronization to the ES cluster cannot be continued and the text index data will be incomplete. If needed, you can only recreate the graph space.

Next

After deploying the Elasticsearch cluster and the Listener, full-text indexes are created automatically on the Elasticsearch cluster. Users can do full-text search now. For more information, see Full-Text search.

Last update: September 2, 2021

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` or `MATCH`.

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.

Natural language full-text search

A natural language search interprets the search string as a phrase in natural human language. The search is case-insensitive. By default, each substring (separated by spaces) will be searched separately. For example, there are three vertices with the tag `player`. The tag `player` contains the property `name`. The `name` of these three vertices are `Kevin Durant`, `Tim Duncan`, and `David Beckham`. Now that the full-text index of `player.name` is established, these three vertices will be queried when using the prefix search statement `LOOKUP ON player WHERE PREFIX(player.name,"d");`.

Syntax

CREATE FULL-TEXT INDEXES

```
CREATE FULLTEXT {TAG | EDGE} INDEX <index_name> ON {<tag_name> | <edge_name>} ([<prop_name_list>]);
```

SHOW FULL-TEXT INDEXES

```
SHOW FULLTEXT INDEXES;
```

REBUILD FULL-TEXT INDEXES

```
REBUILD FULLTEXT INDEX;
```

DROP FULL-TEXT INDEXES

```
DROP FULLTEXT INDEX <index_name>;
```

USE QUERY OPTIONS

```
LOOKUP ON {<tag> | <edge_type>} WHERE <expression> [YIELD <return_list>];

<expression> ::=
    PREFIX | WILDCARD | REGEXP | FUZZY
```

```
<return_list>
    <prop_name> [AS <prop_alias>] [, <prop_name> [AS <prop_alias>] ...]
```

- PREFIX(schema_name.prop_name, prefix_string, row_limit, timeout)
- WILDCARD(schema_name.prop_name, wildcard_string, row_limit, timeout)
- REGEXP(schema_name.prop_name, regexp_string, row_limit, timeout)
- FUZZY(schema_name.prop_name, fuzzy_string, fuzziness, operator, row_limit, timeout)
  - `fuzziness` (optional): Maximum edit distance allowed for matching. The default value is `AUTO`. For other valid values and more information, see Elasticsearch document.
  - `operator` (optional): Boolean logic used to interpret the text. Valid values are `OR` (default) and `AND`.
- `row_limit` (optional): Specifies the number of rows to return. The default value is `100`.
- `timeout` (optional): Specifies the timeout time. The default value is `200ms`.

Examples

```
// This example creates the graph space.
nebula> CREATE SPACE basketballplayer (partition_num=3,replica_factor=1, vid_type=fixed_string(30));

// This example signs in the text service.
nebula> SIGN IN TEXT SERVICE (127.0.0.1:9200);

// This example switches the graph space.
nebula> USE basketballplayer;

// This example adds the listener to the Nebula Graph cluster.
nebula> ADD LISTENER ELASTICSEARCH 192.168.8.5:9789;

// This example creates the tag.
nebula> CREATE TAG player(name string, age int);

// This example creates the native index.
nebula> CREATE TAG INDEX name ON player(name(20));

// This example rebuilds the native index.
nebula> REBUILD TAG INDEX;

// This example creates the full-text index. The index name starts with "nebula".
nebula> CREATE FULLTEXT TAG INDEX nebula_index_1 ON player(name);

// This example rebuilds the full-text index.
nebula> REBUILD FULLTEXT INDEX;

// This example shows the full-text index.
nebula> SHOW FULLTEXT INDEXES;
+------------------+-------------+-------------+--------+
| Name             | Schema Type | Schema Name | Fields |
+------------------+-------------+-------------+--------+
| "nebula_index_1" | "Tag"       | "player"    | "name" |
+------------------+-------------+-------------+--------+

// This example inserts the test data.
nebula> INSERT VERTEX player(name, age) VALUES \
  "Russell Westbrook": ("Russell Westbrook", 30), \
  "Chris Paul": ("Chris Paul", 33),\
  "Boris Diaw": ("Boris Diaw", 36),\
  "David West": ("David West", 38),\
  "Danny Green": ("Danny Green", 31),\
  "Tim Duncan": ("Tim Duncan", 42),\
  "James Harden": ("James Harden", 29),\
  "Tony Parker": ("Tony Parker", 36),\
  "Aron Baynes": ("Aron Baynes", 32),\
  "Ben Simmons": ("Ben Simmons", 22),\
  "Blake Griffin": ("Blake Griffin", 30);

// These examples run test queries.
nebula> LOOKUP ON player WHERE PREFIX(player.name, "B");
+-----------------+
| _vid            |
+-----------------+
| "Boris Diaw"    |
+-----------------+
| "Ben Simmons"   |
+-----------------+
| "Blake Griffin" |
+-----------------+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") YIELD player.name, player.age;
+-----------------+-----------------+-----+
| _vid            | name            | age |
```

```
+-----------------+-----------------+-----+
| "Chris Paul"    | "Chris Paul"    | 33  |
+-----------------+-----------------+-----+
| "Boris Diaw"    | "Boris Diaw"    | 36  |
+-----------------+-----------------+-----+
| "Blake Griffin" | "Blake Griffin" | 30  |
+-----------------+-----------------+-----+

nebula> LOOKUP ON player WHERE WILDCARD(player.name, "*ri*") | YIELD count(*);
+----------+
| count(*) |
+----------+
| 3        |
+----------+

nebula> LOOKUP ON player WHERE REGEXP(player.name, "R.*") YIELD player.name, player.age;
+--------------------+--------------------+-----+
| _vid               | name               | age |
+--------------------+--------------------+-----+
| "Russell Westbrook" | "Russell Westbrook" | 30  |
+--------------------+--------------------+-----+

nebula> LOOKUP ON player WHERE REGEXP(player.name, ".*");
+--------------------+
| _vid               |
+--------------------+
| "Danny Green"      |
+--------------------+
| "David West"       |
+--------------------+
| "Russell Westbrook" |
+--------------------+
...

nebula> LOOKUP ON player WHERE FUZZY(player.name, "Tim Dunncan", AUTO, OR) YIELD player.name;
+-------------+-------------+
| _vid        | name        |
+-------------+-------------+
| "Tim Duncan" | "Tim Duncan" |
+-------------+-------------+

// This example drops the full-text index.
nebula> DROP FULLTEXT INDEX nebula_index_1;
```

Last update: September 17, 2021

4.15 Subgraph and path

4.15.1 GET SUBGRAPH

The `GET SUBGRAPH` statement retrieves information of vertices and edges reachable from the source vertices of the specified edge types and returns information of the subgraph.

Syntax

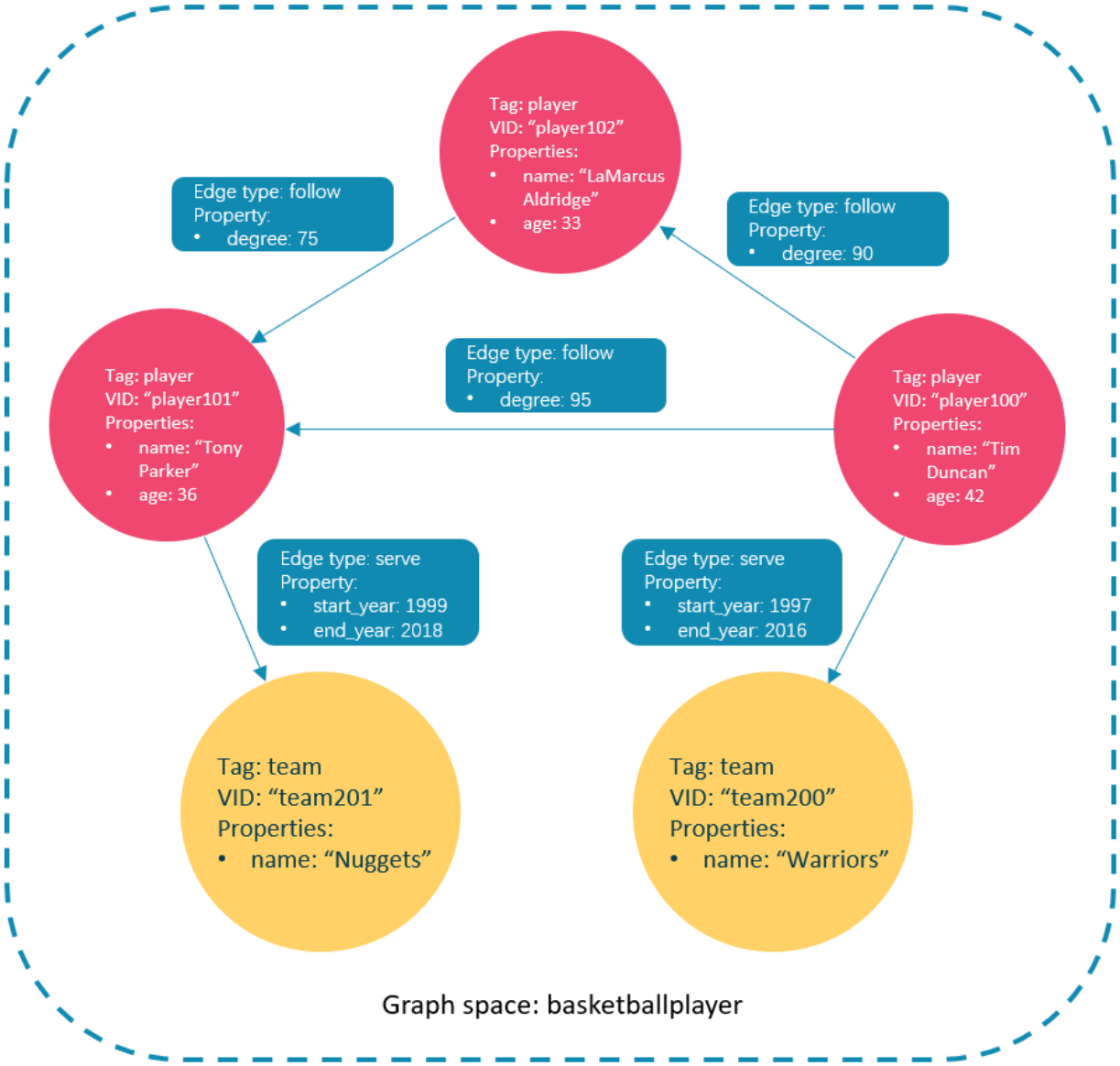
```
GET SUBGRAPH [WITH PROP] [<step_count> STEPS] FROM {<vid>, <vid>...}
[IN <edge_type>, <edge_type>...]
[OUT <edge_type>, <edge_type>...]
[BOTH <edge_type>, <edge_type>...];
```

- `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`.

> **Note**
>
> The path type of `GET SUBGRAPH` is `trail`. Only vertices can be repeatedly visited in graph traversal. For more information, see Path.

Examples

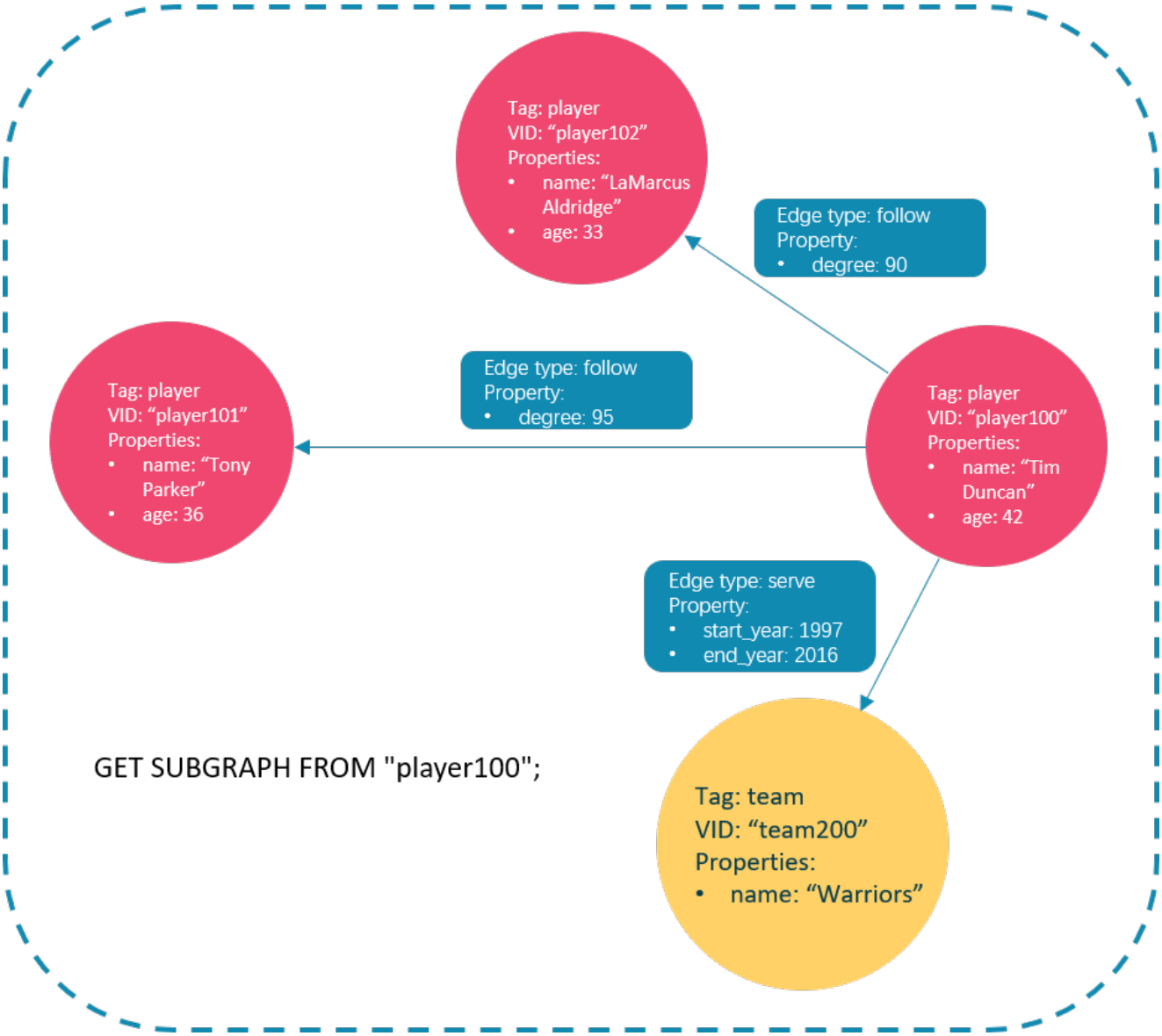
The following graph is used as the sample.



- This example goes one step from the vertex `player100` over all edge types and gets the subgraph.

```
nebula> GET SUBGRAPH 1 STEPS FROM "player100";
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------+
| _vertices                                                             |
_edges                                                                                                                              |
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------+
| [("player100" :player{})]                                             | [[:serve "player100"->"team200" @0 {}], [:follow "player100"->"player101" @0 {}],
[:follow "player100"->"player102" @0 {}]] |
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------+
| [("team200" :team{}), ("player101" :player{}), ("player102" :player{})] | [[:follow "player102"->"player101" @0
{}]]                                                                                  |
+-----------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------------------+
```

The returned subgraph is as follows.



- This example goes one step from the vertex `player100` over incoming `follow` edges and gets the subgraph.

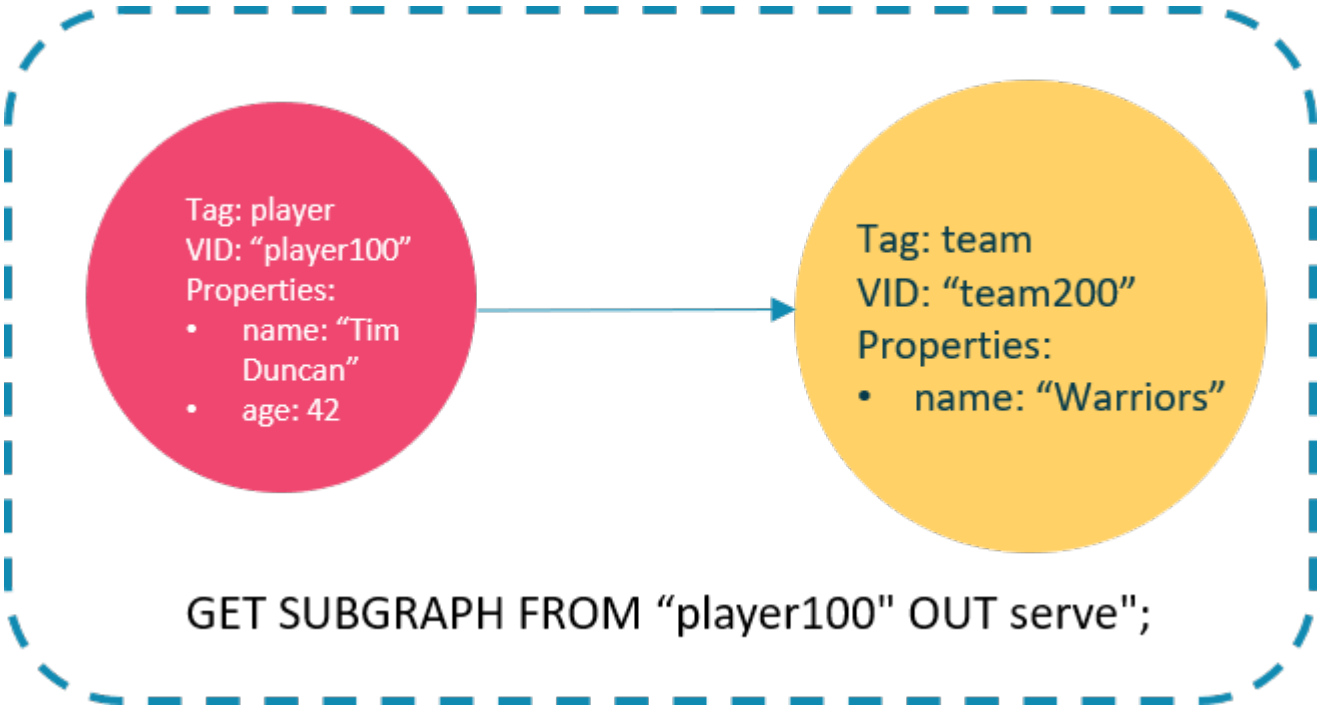
```
nebula> GET SUBGRAPH 1 STEPS FROM "player100" IN follow;
+--------------------------+--------+
| _vertices                | _edges |
+--------------------------+--------+
| [("player100" :player{})] | []     |
+--------------------------+--------+
| []                       | []     |
+--------------------------+--------+
```

There is no incoming `follow` edge to `player100`, so no vertex or edge is returned.

- This example goes one step from the vertex `player100` over outgoing `serve` edges, gets the subgraph, and shows the property of the edge.

```
nebula> GET SUBGRAPH WITH PROP 1 STEPS FROM "player100" OUT serve;
+-------------------------------------------------+-------------------------------------------------------------------------+
| _vertices                                       | _edges                                                                  |
+-------------------------------------------------+-------------------------------------------------------------------------+
| [("player100" :player{age: 42, name: "Tim Duncan"})] | [[:serve "player100"->"team200" @0 {end_year: 2016, start_year: 1997}]] |
+-------------------------------------------------+-------------------------------------------------------------------------+
| [("team200" :team{name: "Warriors"})]           | []                                                                      |
+-------------------------------------------------+-------------------------------------------------------------------------+
```

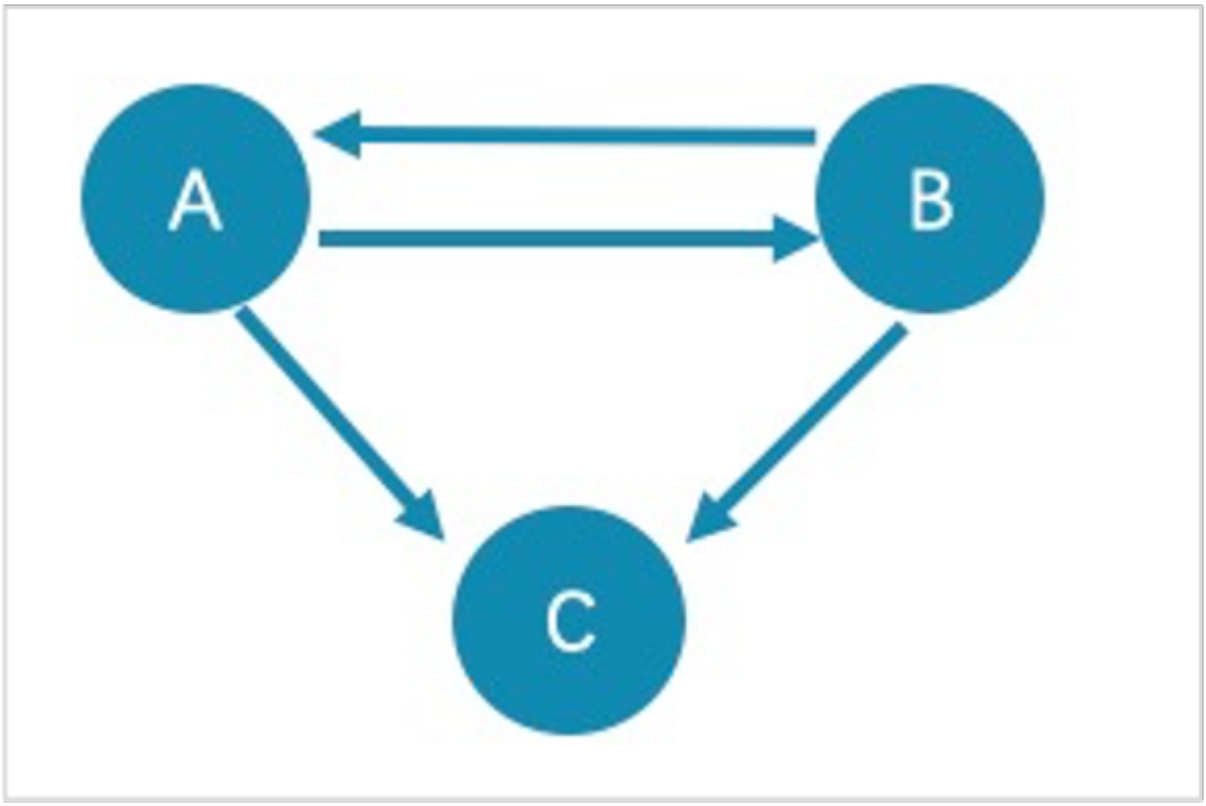
The returned subgraph is as follows.



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->B`, `B->A`, and `A->C`. To show the completeness of the subgraph, an additional hop is made on all vertices that meet the conditions, namely `B->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> go 1 steps from "A" over follow;
```

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 "player141" OUT follow;
+-------------------------------------------------+-----------------------------------------+
| _vertices                                       | _edges                                  |
+-------------------------------------------------+-----------------------------------------+
| [("player141" :player{age: 43, name: "Ray Allen"})]   | [[:follow "player141"->"player124" @0 {}]] |
+-------------------------------------------------+-----------------------------------------+
| [("player124" :player{age: 33, name: "Rajon Rondo"})] | [[:follow "player124"->"player141" @0 {}]] |
+-------------------------------------------------+-----------------------------------------+
```

Last update: September 2, 2021

4.15.2 FIND PATH

The `FIND PATH` statement finds the paths between the selected source vertices and destination vertices.

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 <N> STEPS] [| ORDER BY $-.path] [| LIMIT <M>];

<vertex_id_list> ::=
    [vertex_id [, vertex_id] ...]
```

- `SHORTEST` finds 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.
- `<N>` is the maximum hop number of the path. The default value is `5`.
- `<M>` specifies the maximum number of rows to return.

> **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.
- 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 *;
+------------------------------------------+
| path                                     |
+------------------------------------------+
| <("player102")-[:serve@0 {}]->("team204")> |
+------------------------------------------+
```

```
nebula> FIND SHORTEST PATH WITH PROP FROM "team204" TO "player100" OVER * REVERSELY;
+-------------------------------------------------------------------------------------------------------------------------------------------+
| path                                                                                                                                      |
+-------------------------------------------------------------------------------------------------------------------------------------------+
```

```
| <("team204" :team{name: "Spurs"})<-[:serve@0 {end_year: 2016, start_year: 1997}]-("player100" :player{age: 42, name: "Tim Duncan"})> |
+-------------------------------------------------------------------------------------------------------------------------------------+
```

```
nebula> FIND ALL PATH FROM "player100" TO "team204" OVER * WHERE follow.degree is EMPTY or follow.degree >=0;
+-------------------------------------------------------------------------+
| path                                                                    |
+-------------------------------------------------------------------------+
| <("player100")-[:serve@0 {}]->("team204")>                              |
+-------------------------------------------------------------------------+
| <("player100")-[:follow@0 {}]->("player125")-[:serve@0 {}]->("team204")> |
+-------------------------------------------------------------------------+
| <("player100")-[:follow@0 {}]->("player101")-[:serve@0 {}]->("team204")> |
+-------------------------------------------------------------------------+
...
```

```
nebula> FIND NOLOOP PATH FROM "player100" TO "team204" OVER *;
+-------------------------------------------------------------------------------------------------------+
| path                                                                                                  |
+-------------------------------------------------------------------------------------------------------+
| <("player100")-[:serve@0 {}]->("team204")>                                                            |
+-------------------------------------------------------------------------------------------------------+
| <("player100")-[:follow@0 {}]->("player125")-[:serve@0 {}]->("team204")>                               |
+-------------------------------------------------------------------------------------------------------+
| <("player100")-[:follow@0 {}]->("player101")-[:serve@0 {}]->("team204")>                               |
+-------------------------------------------------------------------------------------------------------+
| <("player100")-[:follow@0 {}]->("player101")-[:follow@0 {}]->("player125")-[:serve@0 {}]->("team204")> |
+-------------------------------------------------------------------------------------------------------+
| <("player100")-[:follow@0 {}]->("player101")-[:follow@0 {}]->("player102")-[:serve@0 {}]->("team204")> |
+-------------------------------------------------------------------------------------------------------+
```

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 `FIND ALL PATH FROM "player100" TO "team204" OVER * WHERE follow.degree is EMPTY or follow.degree >=0;`.

Filtering properties of vertices is not supported for now.

Last update: August 19, 2021

4.16 Query tuning statements

4.16.1 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 Nebula Graph 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"] <your_nGQL_statement>;
```

- `PROFILE`

```
PROFILE [format="row" | "dot"] <your_nGQL_statement>;
```

Output formats

The output of an `EXPLAIN` or a `PROFILE` statement has two formats, the default `row` format and the `dot` 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 format="row" SHOW TAGS;
Execution succeeded (time spent 327/892 us)

Execution Plan

-----+----------+--------------+----------------+-------------------------------------------------------------------------
| id | name     | dependencies | profiling data | operator info                                                           |
-----+----------+--------------+----------------+-------------------------------------------------------------------------
|  1 | ShowTags | 0            |                | outputVar: [{"colNames":[],"name":"__ShowTags_1","type":"DATASET"}] |
|    |          |              |                | inputVar:                                                               |
-----+----------+--------------+----------------+-------------------------------------------------------------------------
|  0 | Start    |              |                | outputVar: [{"colNames":[],"name":"__Start_0","type":"DATASET"}]    |
-----+----------+--------------+----------------+-------------------------------------------------------------------------
```

- `PROFILE`

```
nebula> PROFILE format="row" SHOW TAGS;
+--------+
| Name   |
+--------+
| player |
+--------+
| team   |
+--------+
Got 2 rows (time spent 2038/2728 us)

Execution Plan

-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------
| id | name     | dependencies | profiling data                                  | operator info                                                           |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------
|  1 | ShowTags | 0            | ver: 0, rows: 1, execTime: 42us, totalTime: 1177us | outputVar: [{"colNames":[],"name":"__ShowTags_1","type":"DATASET"}] |
|    |          |              |                                                 | inputVar:                                                               |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------
|  0 | Start    |              | ver: 0, rows: 0, execTime: 1us, totalTime: 57us | outputVar: [{"colNames":[],"name":"__Start_0","type":"DATASET"}]    |
-----+----------+--------------+-------------------------------------------------+-------------------------------------------------------------------------
```

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

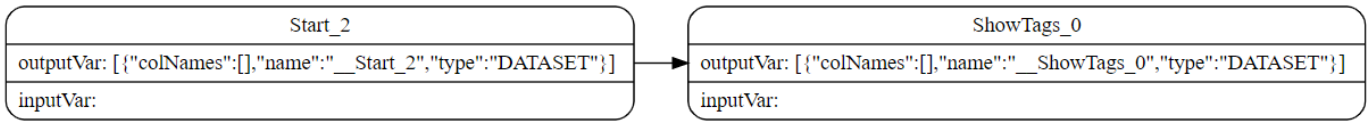
You can use the `format="dot"` option to output the return message in the `dot` language, and then use Graphviz to generate a graph of the plan.

> **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.



Last update: August 24, 2021

4.17 Operation and maintenance statements

4.17.1 BALANCE syntax

The `BALANCE` statements support the load balancing operations of the Nebula Graph Storage services. For more information about storage load balancing and examples for using the `BALANCE` statements, see Storage load balance.

The `BALANCE` statements are listed as follows.

Syntax	Description
`BALANCE DATA`	Starts a task to balance the distribution of storage partitions in a Nebula Graph cluster. It returns the task ID (`balance_id`).
`BALANCE DATA <balance_id>`	Shows the status of the `BALANCE DATA` task.
`BALANCE DATA STOP`	Stops the `BALANCE DATA` task.
`BALANCE DATA REMOVE <host_list>`	Scales in the Nebula Graph cluster and detaches specific storage hosts.
`BALANCE LEADER`	Balances the distribution of storage raft leaders in a Nebula Graph cluster.

Last update: August 23, 2021

4.17.2 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 time-consuming if the data amount in the graph space is large. The job manager helps you run, show, stop, and recover jobs.

SUBMIT JOB COMPACT

The `SUBMIT JOB COMPACT` statement triggers the long-term RocksDB `compact` operation.

For more information about `compact` configuration, see Storage Service configuration.

EXAMPLE

```
nebula> SUBMIT JOB COMPACT;
+------------+
| New Job Id |
+------------+
| 40         |
+------------+
```

SUBMIT JOB FLUSH

The `SUBMIT JOB FLUSH` statement writes the RocksDB memfile in the memory to the hard disk.

EXAMPLE

```
nebula> SUBMIT JOB FLUSH;
+------------+
| New Job Id |
+------------+
| 96         |
+------------+
```

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.

EXAMPLE

```
nebula> SUBMIT JOB STATS;
+------------+
| New Job Id |
+------------+
| 97         |
+------------+
```

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.

`job_id` is returned when you run the `SUBMIT JOB` statement.

EXAMPLE

```
nebula> SHOW JOB 96;
+----------------+---------------+------------+-------------------------+-------------------------+
| Job Id(TaskId) | Command(Dest) | Status     | Start Time              | Stop Time               |
+----------------+---------------+------------+-------------------------+-------------------------+
| 96             | "FLUSH"       | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+----------------+---------------+------------+-------------------------+-------------------------+
```

```
| 0             | "storaged2"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
| 1             | "storaged0"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
| 2             | "storaged1"   | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+---------------+---------------+------------+-------------------------+-------------------------+
```

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.
`Command(Dest)`	The first row shows the command executed and the other rows show on which storaged processes the task is running.
`Status`	Shows the status of the job or task. For more information, see Job status.
`Start Time`	Shows a timestamp indicating the time when the job or task enters the `RUNNING` phase.
`Stop Time`	Shows a timestamp indicating the time when the job or task gets `FINISHED`, `FAILED`, or `STOPPED`.

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.

```
Queue -- running -- finished -- removed
     \          \              /
      \          \ -- failed -- /
       \          \            /
        \ ---------- stopped -/
```

SHOW JOBS

The `SHOW JOBS` statement lists all the unexpired jobs.

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.

EXAMPLE

```
nebula> SHOW JOBS;
+--------+---------------------+------------+-------------------------+-------------------------+
| Job Id | Command             | Status     | Start Time              | Stop Time               |
+--------+---------------------+------------+-------------------------+-------------------------+
| 97     | "STATS"             | "FINISHED" | 2020-11-28T14:48:52.000 | 2020-11-28T14:48:52.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
| 96     | "FLUSH"             | "FINISHED" | 2020-11-28T14:14:29.000 | 2020-11-28T14:14:29.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
| 95     | "STATS"             | "FINISHED" | 2020-11-28T13:02:11.000 | 2020-11-28T13:02:11.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
| 86     | "REBUILD_EDGE_INDEX" | "FINISHED" | 2020-11-26T13:38:24.000 | 2020-11-26T13:38:24.000 |
+--------+---------------------+------------+-------------------------+-------------------------+
```

STOP JOB

The `STOP JOB` statement stops jobs that are not finished.

EXAMPLE

```
nebula> STOP JOB 22;
+--------------+
| Result       |
+--------------+
| "Job stopped" |
+--------------+
```

RECOVER JOB

The `RECOVER JOB` statement re-executes the failed jobs and returns the number of recovered jobs.

EXAMPLE

```
nebula> RECOVER JOB;
+------------------+
| Recovered job num |
+------------------+
| 5 job recovered   |
+------------------+
```

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"
```

Last update: September 2, 2021

4.17.3 Kill queries

`KILL QUERY` can terminate the query being executed, and is often used to terminate slow 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)]: Execution had been killed
```

Last update: August 23, 2021

5. Deployment and installation

5.1 Prepare resources for compiling, installing, and running Nebula Graph

This topic describes the requirements and suggestions for compiling and installing Nebula Graph, as well as how to estimate the resource you need to reserve for running a Nebula Graph cluster.

5.1.1 Reading guide

If you are reading this topic with the questions listed below, click them to jump to their answers.

- What do I need to compile Nebula Graph?
- What do I need to run Nebula Graph in a test environment?
- What do I need to run Nebula Graph in a production environment?
- How much memory and disk space do I need to reserve for my Nebula Graph cluster?

5.1.2 Requirements for compiling the Nebula Graph source code

Hardware requirements for compiling Nebula Graph

Item	Requirement
CPU architecture	x86_64
Memory	4 GB
Disk	10 GB, SSD

Supported operating systems for compiling Nebula Graph

For now, we can only compile Nebula Graph in the Linux system. We recommend that you use any Linux system with kernel version `2.6.32` or above.

Software requirements for compiling Nebula Graph

You must have the correct version of the software listed below to compile Nebula Graph. If they are not as required or you are not sure, follow the steps in Prepare software for compiling Nebula Graph to get them ready.

Software	Version	Note
glibc	2.17 or above	You can run `ldd --version` 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	-
gcc	7.5.0 or above	You can run `gcc -v` to check the gcc version.
gcc-c++	Any stable version	-
cmake	3.9.0 or above	You can run `cmake --version` to check the cmake version.
gettext	Any stable 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 Nebula Graph

This section guides you through the downloading and installation of software required for compiling Nebula Graph.

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 \
                    gettext \
                    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 \
                        m4 \
                        git \
                        wget \
                        unzip \
                        xz-utils \
                        curl \
                        lsb-core \
                        build-essential \
                        libreadline-dev \
                        ncurses-dev \
                        cmake \
                        gettext
   ```

2. Check if the GCC and cmake on your host are in the right version. See Software requirements for compiling Nebula Graph for the required versions.

   ```
   $ g++ --version
   $ cmake --version
   ```

   If your GCC and CMake are in the right version, then you are all set. If they are not, follow the sub-steps as follows.

   1. Clone the `nebula-common` repository to your host.

   ```
   ```bash
   $ git clone -b 2.5.0 https://github.com/vesoft-inc/nebula-common.git
   ```

   Users can use the `--branch` or `-b` option to specify the branch to be cloned. For example, for 2.5.0, run the following command.

   ```bash
   $ git clone --branch v2.5.0 https://github.com/vesoft-inc/nebula-common.git
   ```
   ```

   2. Make `nebula-common` the current working directory.

   ```
   ```bash
   $ cd nebula-common
   ```
   ```

   3. Run the following commands to install and enable CMake and GCC.

```bash
// Install CMake.
$ ./third-party/install-cmake.sh cmake-install

// Enable CMake.
$ source cmake-install/bin/enable-cmake.sh

// Authorize the write privilege to the opt directory.
$ sudo mkdir /opt/vesoft && sudo chmod -R a+w /opt/vesoft

// Install GCC. Installing GCC to the opt directory requires the write privilege. And users can change it to other locations.
$ ./third-party/install-gcc.sh --prefix=/opt

// Enable GCC.
$ source /opt/vesoft/toolset/gcc/7.5.0/enable
```

3. Execute the script `install-third-party.sh`.

```
$ ./third-party/install-third-party.sh
```

5.1.3 Requirements and suggestions for installing Nebula Graph 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 Nebula Graph in the Linux system. To install Nebula Graph 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 Nebula Graph as follows:

Machine name	Number of metad	Number of storaged	Number of graphd
A	1	1	1
B	None	1	1
C	None	1	1

5.1.4 Requirements and suggestions for installing Nebula Graph in production environments

Hardware requirements for production environments

Item	Requirement
CPU architecture	x86_64
Number of CPU core	48
Memory	96 GB
Disk	2 * 900 GB, NVMe SSD

Supported operating systems for production environments

For now, we can only install Nebula Graph in the Linux system. To install Nebula Graph 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 Nebula Graph. For more information, see kernel configuration.

Suggested service architecture for production environments

Danger

DO NOT deploy a cluster 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 Nebula Graph as follows:

Machine name	Number of metad	Number of storaged	Number of graphd
A	1	1	1
B	1	1	1
C	1	1	1
D	None	1	1
E	None	1	1

5.1.5 Capacity requirements for running a Nebula Graph cluster

Users can estimate the memory, disk space, and partition number needed for a Nebula Graph 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` * 6 * 120%	-
Memory for a cluster	Bytes	[`the_sum_of_edge_number_and_vertex_number` * 15 + `the_number_of_RocksDB_instances` * (`write_buffer_size` * `max_write_buffer_number` + `rocksdb_block_cache`)] * 120%	`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	-	`the_number_of_disks_in_the_cluster` * `disk_partition_num_multiplier`	`disk_partition_num_multiplier` is an integer between 2 and 10 (both including). Its value depends on the disk performance. Use 2 for HDD.

- Question 1: Why do we multiply the disk space and memory by 120%?

  Answer: The extra 20% is for buffer.

- Question 2: How to get the number of RocksDB instances?

  Answer: Each directory in the `--data_path` item in the `etc/nebula-storaged.conf` file corresponds to a RocksDB instance. Count the number of directories to get the RocksDB instance number.

> **Note**
>
> Users can decrease the memory size occupied by the bloom filter by adding `--enable_partitioned_index_filter=true` in `etc/nebula-storaged.conf`. But it may decrease the read performance in some random-seek cases.

5.1.6 FAQ

About storage devices

Nebula Graph 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.
- Do not use RAID.
- Use local SSD devices.

About CPU architecture

> **Enterpriseonly**
>
> Nebula Graph 2.5.0 does not support running or compiling directly on the ARM architecture (including Apple Mac M1 or Huawei Kunpeng).

Last update: September 2, 2021

5.2 Compile and install Nebula Graph

5.2.1 Install Nebula Graph by compiling the source code

Installing Nebula Graph 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 Nebula Graph.
- The host to be installed with Nebula Graph has access to the Internet.

Installation steps

1. Use Git to clone the source code of Nebula Graph to the host.
   - [Recommended] To install Nebula Graph v2.5.0, run the following command.

     ```
     $ git clone --branch v2.5.0 https://github.com/vesoft-inc/nebula-graph.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-graph.git
     ```

2. Make the `nebula-graph` directory the current working directory.

   ```
   $ cd nebula-graph
   ```

3. Create a `build` directory and make it the current working directory.

   ```
   $ mkdir build && cd build
   ```

4. Generate Makefile with CMake.

   > **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.

   - If the source code of Nebula Graph v2.5.0 is installed and cloned in step 1, run the following command. The `-DNEBULA_COMMON_REPO_TAG` and `-DNEBULA_STORAGE_REPO_TAG` options are used to specify the correct branches of nebula-common and nebula-storage repositories to keep the releases of the Nebula Graph components consistent.

     ```
     $ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release \
     -DNEBULA_COMMON_REPO_TAG=v2.5.0 -DNEBULA_STORAGE_REPO_TAG=v2.5.0 ..
     ```

   - If the source code of the `master` branch is installed in step 1, run the following command.

     ```
     $ cmake -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DENABLE_MODULE_UPDATE=ON -DCMAKE_BUILD_TYPE=Release ..
     ```

5. Compile Nebula Graph.

   > **Note**
   >
   > Check Prepare resources for compiling, installing, and running Nebula Graph.

To speed up the compiling, use the `-j` option to set a concurrent number `N`. It should be \(\min(\text{CPU}core number, \frac{the_memory_size(GB)}{2})\).

```
$ make -j{N} # E.g., make -j2
```

6. Install Nebula Graph.

```
$ sudo make install-all
```

7. 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 Nebula Graph release is installed, update it in the following steps.

```
1. In the `nebula-graph/` directory, run `git pull upstream master` to update the source code.

2. In the `nebula-graph/modules/common/` and `nebula-graph/modules/storage/` directories, run `git pull upstream master` separately.

3. In the `nebula-graph/build/` directory, run `make -j{N}` and `make install-all` again.
```

Next

- Manage Nebula Graph services
- Connect to Nebula Graph
- Nebula Graph CRUD

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.

ENABLE_BUILD_STORAGE

Starting from 2.5.0, Nebula Graph uses two separated github repositories of graph and storage for separated compiling. The `ENABLE_BUILD_STORAGE` variable is set to `OFF` by default so that the storage service is not installed together with the graph service.

If you are deploying Nebula Graph on a single host for testing, you can set `ENABLE_BUILD_STORAGE` to `ON` to download and install the storage service automatically.

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 Nebula Graph 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 Nebula Graph developers.

CMAKE_BUILD_TYPE

Nebula Graph supports the following building types of `MAKE_BUILD_TYPE`:

- `Debug`

  The default value of `CMAKE_BUILD_TYPE`. It indicates building Nebula Graph with the debug info but not the optimization options.

- `Release`

  It indicates building Nebula Graph with the optimization options but not the debug info.

- `RelWithDebInfo`

  It indicates building Nebula Graph with the optimization options and the debug info.

- `MinSizeRel`

  It indicates building Nebula Graph with the optimization options for controlling the code size but not the debug info.

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 Nebula Graph.

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.
4. Update the code `git pull`, compile again, and add the CMake option `-DENABLE_MODULE_UPDATE=ON` in step 4.

Last update: August 25, 2021

5.2.2 Install Nebula Graph with RPM or DEB package

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

Prerequisites

Prepare the right resources.

> **Note**
>
> The console is not complied or packaged with Nebula Graph server binaries. You can install nebula-console by yourself.

Download the package from cloud service

- Download the released version.

  URL:

```
//Centos 6
https://oss-cdn.nebula-graph.io/package/<release_version>/nebula-graph-<release_version>.el6.x86_64.rpm

//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 release package 2.5.0 for `Centos 7.5`:

```
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.el7.x86_64.rpm.sha256sum.txt
```

  download release package `2.5.0` for `Ubuntu 1804`:

```
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.io/package/2.5.0/nebula-graph-2.5.0.ubuntu1804.amd64.deb.sha256sum.txt
```

- Download the nightly version.

> **Danger**
>
> - Nightly versions are usually used to test new features. Don't use it for production.
> - Nightly versions may not be build successfully every night. And the names may change from day to day.

URL:

```
//Centos 6
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el6.x86_64.rpm

//Centos 7
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el7.x86_64.rpm

//Centos 8
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.el8.x86_64.rpm

//Ubuntu 1604
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1604.amd64.deb

//Ubuntu 1804
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu1804.amd64.deb

//Ubuntu 2004
https://oss-cdn.nebula-graph.io/package/v2-nightly/<yyyy.mm.dd>/nebula-graph-<yyyy.mm.dd>-nightly.ubuntu2004.amd64.deb
```

For example, download the `Centos 7.5` package developed and built in `2021.03.28`:

```
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.el7.x86_64.rpm.sha256sum.txt
```

For example, download the `Ubuntu 1804` package developed and built in `2021.03.28`:

```
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb
wget https://oss-cdn.nebula-graph.io/package/v2-nightly/2021.03.28/nebula-graph-2021.03.28-nightly.ubuntu1804.amd64.deb.sha256sum.txt
```

Install Nebula Graph

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

```
sudo rpm -ivh --prefix=<installation_path> <package_name>
```

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

```
sudo dpkg -i --instdir==<installation_path> <package_name>
```

> **Note**
>
> The default installation path is `/usr/local/nebula/`.

What's next

- start Nebula Graph
- connect to Nebula Graph

Last update: August 30, 2021

5.2.3 Deploy Nebula Graph with Docker Compose

Using Docker Compose can quickly deploy Nebula Graph services based on the prepared configuration file. It is only recommended to use this method when testing functions of Nebula Graph.

Prerequisites

- You have installed the following applications on your host.

Application	Recommended version	Official installation reference
Docker	Latest	Install Docker Engine
Docker Compose	Latest	Install Docker Compose
Git	Latest	Download Git

- If you are deploying Nebula Graph 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 Nebula Graph with Docker Compose on your host, to avoid compatibility issues, you need to delete the `nebula-docker-compose/data` directory.

How to deploy and connect to Nebula Graph

1. Clone the `master` branch of the `nebula-docker-compose` repository to your host with Git.

> **Danger**
>
> The `master` branch contains the untested code for the latest Nebula Graph development release. **DO NOT** use this release in a production environment.

```
$ git clone -b v2.5.0 https://github.com/vesoft-inc/nebula-docker-compose.git
```

2. Go to the `nebula-docker-compose` directory.

```
$ cd nebula-docker-compose/
```

3. Run the following command to start all the Nebula Graph services.

> **Note**
>
> Update the Nebula Graph images and Nebula Console images first if they are out of date.

```
[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_metad1_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
Creating nebula-docker-compose_graphd1_1   ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_storaged1_1 ... done
```

> **Note**
>
> For more information of the preceding services, see Nebula Graph architecture.

4. Connect to Nebula Graph.

   a. Run the following command to start a new docker container with the Nebula Console image, and connect the container to the network where Nebula Graph is deployed (nebula-docker-compose_nebula-net).

```
$ docker run --rm -ti --network nebula-docker-compose_nebula-net --entrypoint=/bin/sh vesoft/nebula-console:v2.5.0
```

> **Note**
>
> The local network may be different from the `nebula-docker-compose_nebula-net` in the above example. Use the following command.

```
$ docker network  ls
NETWORK ID          NAME                                DRIVER              SCOPE
a74c312b1d16        bridge                              bridge              local
dbfa82505f0e        host                                host                local
ed55ccf356ae        nebula-docker-compose_nebula-net    bridge              local
93ba48b4b288        none                                null                local
```

   b. Connect to Nebula Graph with Nebula Console.

```
docker> nebula-console -u <user_name> -p <password> --address=graphd --port=9669
```

> **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.

   c. Run the `SHOW HOSTS` statement to check the status of the `nebula-storaged` processes.

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-------------+------+----------+--------------+----------------------+------------------------+
| "Total"     |      |          | 0            |                      |                        |
+-------------+------+----------+--------------+----------------------+------------------------+
```

5. Run `exit` twice to switch back to your terminal (shell). You can run Step 4 to log in to Nebula Graph again.

Check the Nebula Graph service status and ports

Run `docker-compose ps` to list all the services of Nebula Graph and their status and ports.

```
$ docker-compose ps
Name                     Command                       State                                                Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     ./bin/nebula-graphd --flag ...   Up (health: starting)   13000/tcp, 13002/tcp, 0.0.0.0:33295->19669/tcp, 0.0.0.0:33291->19670/tcp,
                                                                                             3699/tcp, 0.0.0.0:33298->9669/tcp
nebula-docker-compose_graphd2_1     ./bin/nebula-graphd --flag ...   Up (health: starting)   13000/tcp, 13002/tcp, 0.0.0.0:33285->19669/tcp, 0.0.0.0:33284->19670/tcp,
                                                                                             3699/tcp, 0.0.0.0:33286->9669/tcp
nebula-docker-compose_graphd_1      ./bin/nebula-graphd --flag ...   Up (health: starting)   13000/tcp, 13002/tcp, 0.0.0.0:33288->19669/tcp, 0.0.0.0:33287->19670/tcp,
                                                                                             3699/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (health: starting)   11000/tcp, 11002/tcp, 0.0.0.0:33276->19559/tcp, 0.0.0.0:33275->19560/tcp,
                                                                                             45500/tcp, 45501/tcp, 0.0.0.0:33278->9559/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (health: starting)   11000/tcp, 11002/tcp, 0.0.0.0:33279->19559/tcp, 0.0.0.0:33277->19560/tcp,
                                                                                             45500/tcp, 45501/tcp, 0.0.0.0:33281->9559/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (health: starting)   11000/tcp, 11002/tcp, 0.0.0.0:33282->19559/tcp, 0.0.0.0:33280->19560/tcp,
                                                                                             45500/tcp, 45501/tcp, 0.0.0.0:33283->9559/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (health: starting)   12000/tcp, 12002/tcp, 0.0.0.0:33290->19779/tcp, 0.0.0.0:33289->19780/tcp,
                                                                                             44500/tcp, 44501/tcp, 0.0.0.0:33294->9779/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (health: starting)   12000/tcp, 12002/tcp, 0.0.0.0:33296->19779/tcp, 0.0.0.0:33292->19780/tcp,
                                                                                             44500/tcp, 44501/tcp, 0.0.0.0:33299->9779/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (health: starting)   12000/tcp, 12002/tcp, 0.0.0.0:33297->19779/tcp, 0.0.0.0:33293->19780/tcp,
                                                                                             44500/tcp, 44501/tcp, 0.0.0.0:33300->9779/tcp
```

Nebula Graph 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 Nebula Graph services.

Check the service data and logs

All the data and logs of Nebula Graph are stored persistently in the `nebula-docker-compose/data` and `nebula-docker-compose/logs` directories.

The structure of the directories is as follows:

```
nebula-docker-compose/
  |-- docker-compose.yaml
  ├── data
  │   ├── meta0
  │   ├── meta1
  │   ├── meta2
  │   ├── storage0
  │   ├── storage1
  │   └── storage2
  └── logs
      ├── graph
      ├── graph1
      ├── graph2
      ├── meta0
      ├── meta1
      ├── meta2
      ├── storage0
      ├── storage1
      └── storage2
```

Stop the Nebula Graph services

You can run the following command to stop the Nebula Graph services:

```
$ docker-compose down
```

The following information indicates you have successfully stopped the Nebula Graph services:

```
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad0_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing network nebula-docker-compose_nebula-net
```

Danger

The parameter `-v` in the command `docker-compose down -v` will **delete** all your local Nebula Graph storage data. Try this command if you are using the nightly release and having some compatibility issues.

Modify configurations

The configuration file of Nebula Graph deployed by 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.

For more instructions, see Configurations.

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:v2.5.0
    ...
    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 update the docker images of Nebula Graph services

To update the images of the Graph Service, Storage Service, and Meta Service, run `docker-compose pull` in the `nebula-docker-compose` directory.

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/increase-rate-limit`.

You have met the rate limit of Docker Hub. Learn more on Understanding Docker Hub Rate Limiting.

How to update the Nebula console client

To update the Nebula Console client, run the following command.

```
docker pull vesoft/nebula-console:v2.5.0
```

How to upgrade Nebula Graph services

To upgrade Nebula Graph, update the Nebula Graph docker images and restart the services.

1. In the `nebula-docker-compose` directory, run `docker-compose pull` to update the Nebula Graph docker images.
2. Run `docker-compose down` to stop the Nebula Graph services.
3. Run `docker-compose up -d` to start the Nebula Graph services again.

Why can't I connect to Nebula Graph via port 3699 after updating the nebula-docker-compose repository (Nebula Graph 2.0.0-RC)?

In Nebula Graph 2.0.0-RC release, the default port is changed from `3699` to `9669`. Please use port `9669` to connect to Nebula Graph, or modify the port in `docker-compose.yaml`.

Why can't I access the data after updating the nebula-docker-compose repository? (Jan 4, 2021)

If you have updated the nebula-docker-compose repository after Jan 4, 2021, and there are pre-existing data, modify the `docker-compose.yaml` file and change the port numbers to the previous ones before connecting to Nebula Graph.

Why can't I access the data after updating the nebula-docker-compose repository? (Jan 27, 2021)

The data format has been modified on Jan 27, 2021, and is incompatible with the previous data. Run `docker-compose down -v` to delete all your local data.

Related documents

- Install and deploy Nebula Graph with the source code
- Install Nebula Graph by RPM or DEB
- Connect to Nebula Graph

Last update: September 3, 2021

5.2.4 Deploy Nebula Graph cluster with RPM/DEB package

For now, Nebula Graph does not provide an official deployment tool. Users can deploy a Nebula Graph cluster with RPM or DEB package manually. This topic provides some examples of deploying Nebula Graph clusters.

Deployment

Machine name	IP address	Number of graphd	Number of storaged	Number of metad
A	192.168.10.111	1	1	1
B	192.168.10.112	1	1	1
C	192.168.10.113	1	1	1
D	192.168.10.114	1	1	None
E	192.168.10.115	1	1	None

Prerequisites

Prepare 5 machines for deploying the cluster.

Manual deployment process

STEP 1: INSTALL NEBULA GRAPH

Install Nebula Graph on each machine in the cluster. Available approaches of installation are as follows.

- Install Nebula Graph with RPM or DEB package
- Install Nebula Graph by compiling the source code

STEP 2: MODIFY THE CONFIGURATIONS

To deploy Nebula Graph according to your requirements, you have to modify the configuration files.

All the configuration files for Nebula Graph, 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
A	`nebula-graphd.conf`, `nebula-storaged.conf`, `nebula-metad.conf`
B	`nebula-graphd.conf`, `nebula-storaged.conf`, `nebula-metad.conf`
C	`nebula-graphd.conf`, `nebula-storaged.conf`, `nebula-metad.conf`
D	`nebula-graphd.conf`, `nebula-storaged.conf`
E	`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 Nebula Graph cluster.

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=192.168.10.111
# 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=192.168.10.111
# 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=192.168.10.111
# Meta daemon listening port
--port=9559
```

- Deploy machine B
  - `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=192.168.10.112
# 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=192.168.10.112
# 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`

```
########## 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=192.168.10.113
# 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`

```
########## 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=192.168.10.114
# Storage daemon listening port
--port=9779
```

- Deploy machine E
  - `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=192.168.10.115
# 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=192.168.10.115
# Storage daemon listening port
--port=9779
```

STEP 3: START THE CLUSTER

Start the corresponding service on each machine. Descriptions are as follows.

Machine name	The process to be started
A	graphd, storaged, metad
B	graphd, storaged, metad
C	graphd, storaged, metad
D	graphd, storaged
E	graphd, storaged

The command to start the Nebula Graph services is as follows.

```
sudo /usr/local/nebula/scripts/nebula.service start <metad|graphd|storaged|all>
```

Note

- When the graphd process, the storaged process, and the metad process are all started, you can use `all` instead.
- `/usr/local/nebula` is the default installation path for Nebula Graph. Use the actual path if you have customized the path. For more information about how to start and stop the services, see Manage Nebula Graph services.

STEP 4: CHECK THE CLUSTER STATUS

Install the native CLI client Nebula Console, then connect to any machine that has started the graphd process, 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 Nebula Graph!

> SHOW HOSTS;
+-----------------+------+----------+--------------+----------------------+------------------------+
| Host            | Port | Status   | Leader count | Leader distribution  | Partition distribution |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.111" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.112" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.113" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.114" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "192.168.10.115" | 9779 | "ONLINE" | 0            | "No valid partition" | "No valid partition"   |
+-----------------+------+----------+--------------+----------------------+------------------------+
| "Total"          |      |          | 0            |                      |                        |
+-----------------+------+----------+--------------+----------------------+------------------------+
```

Last update: September 2, 2021

5.3 Manage Nebula Graph Service

You can use the `nebula.service` script to start, stop, restart, terminate, and check the Nebula Graph services. This topic takes starting, stopping and checking the Nebula Graph services for examples.

`nebula.service` is stored in the `/usr/local/nebula/` directory by default, which is also the default installation path of Nebula Graph. If you have customized the path, use the actual path in your environment.

5.3.1 Syntax

```
$ sudo /usr/local/nebula/scripts/nebula.service
[-v] [-c <config_file_path>]
<start|stop|restart|status|kill>
<metad|graphd|storaged|all>
```

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 Nebula Graph services as the target services.

5.3.2 Start Nebula Graph

In non-container environment

Run the following command to start Nebula Graph.

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

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to start Nebula Graph.

```
nebula-docker-compose]$ docker-compose up -d
Building with native build. Learn about native build in Compose here: https://docs.docker.com/go/compose-native-build/
Creating network "nebula-docker-compose_nebula-net" with the default driver
Creating nebula-docker-compose_metad0_1 ... done
Creating nebula-docker-compose_metad2_1 ... done
Creating nebula-docker-compose_metad1_1 ... done
Creating nebula-docker-compose_storaged2_1 ... done
Creating nebula-docker-compose_graphd1_1   ... done
Creating nebula-docker-compose_storaged1_1 ... done
Creating nebula-docker-compose_storaged0_1 ... done
Creating nebula-docker-compose_graphd2_1   ... done
Creating nebula-docker-compose_graphd_1    ... done
```

5.3.3 Stop Nebula Graph

Danger

Don't run `kill -9` to forcibly terminate the processes, otherwise, there is a low probability of data loss.

In non-container environment

Run the following command to stop Nebula Graph.

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

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to stop Nebula Graph.

```
nebula-docker-compose]$ docker-compose down
Stopping nebula-docker-compose_graphd_1    ... done
Stopping nebula-docker-compose_graphd2_1   ... done
Stopping nebula-docker-compose_storaged0_1 ... done
Stopping nebula-docker-compose_storaged1_1 ... done
Stopping nebula-docker-compose_graphd1_1   ... done
Stopping nebula-docker-compose_storaged2_1 ... done
Stopping nebula-docker-compose_metad1_1    ... done
Stopping nebula-docker-compose_metad2_1    ... done
Stopping nebula-docker-compose_metad0_1    ... done
Removing nebula-docker-compose_graphd_1    ... done
Removing nebula-docker-compose_graphd2_1   ... done
Removing nebula-docker-compose_storaged0_1 ... done
Removing nebula-docker-compose_storaged1_1 ... done
Removing nebula-docker-compose_graphd1_1   ... done
Removing nebula-docker-compose_storaged2_1 ... done
Removing nebula-docker-compose_metad1_1    ... done
Removing nebula-docker-compose_metad2_1    ... done
Removing nebula-docker-compose_metad0_1    ... done
Removing network nebula-docker-compose_nebula-net
```

If you are using a development or nightly version for testing and have compatibility issues, try to run `docker-compose down -v` to **DELETE** all data stored in Nebula Graph and import data again.

5.3.4 Check the service status

In non-container environment

Run the following command to check the service status of Nebula Graph.

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

- Nebula Graph is running normally if the following information is returned.

```
[INFO] nebula-metad: Running as 26601, Listening on 9559
[INFO] nebula-graphd: Running as 26644, Listening on 9669
[INFO] nebula-storaged: Running as 26709, Listening on 9779
```

- If the return information is similar to the following one, there is a problem.

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

The Nebula Graph 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 return information to troubleshoot problems.

You may also go to the Nebula Graph community for help.

In docker container (deployed with docker-compose)

Run the following command in the `nebula-docker-compose/` directory to check the service status of Nebula Graph.

```
nebula-docker-compose]$ docker-compose ps
            Name                           Command                  State                                        Ports
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
nebula-docker-compose_graphd1_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49223->19669/tcp, 0.0.0.0:49222->19670/tcp, 0.0.0.0:49224->9669/tcp
nebula-docker-compose_graphd2_1     /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49229->19669/tcp, 0.0.0.0:49228->19670/tcp, 0.0.0.0:49230->9669/tcp
nebula-docker-compose_graphd_1      /usr/local/nebula/bin/nebu ...   Up (healthy)   0.0.0.0:49221->19669/tcp, 0.0.0.0:49220->19670/tcp, 0.0.0.0:9669->9669/tcp
nebula-docker-compose_metad0_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49212->19559/tcp, 0.0.0.0:49211->19560/tcp, 0.0.0.0:49213->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad1_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49209->19559/tcp, 0.0.0.0:49208->19560/tcp, 0.0.0.0:49210->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_metad2_1      ./bin/nebula-metad --flagf ...   Up (healthy)   0.0.0.0:49206->19559/tcp, 0.0.0.0:49205->19560/tcp, 0.0.0.0:49207->9559/tcp,
                                                                                    9560/tcp
nebula-docker-compose_storaged0_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49218->19779/tcp, 0.0.0.0:49217->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49219->9779/tcp, 9780/tcp
nebula-docker-compose_storaged1_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49215->19779/tcp, 0.0.0.0:49214->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                     0.0.0.0:49216->9779/tcp, 9780/tcp
nebula-docker-compose_storaged2_1   ./bin/nebula-storaged --fl ...   Up (healthy)   0.0.0.0:49226->19779/tcp, 0.0.0.0:49225->19780/tcp, 9777/tcp, 9778/tcp,
                                                                                    0.0.0.0:49227->9779/tcp, 9780/tcp
```

Use the `CONTAINER ID` to log in the container and troubleshoot.

```
nebula-docker-compose]$ docker exec -it 2a6c56c405f5 bash
[root@2a6c56c405f5 nebula]#
```

5.3.5 What's next

Connect to Nebula Graph

Last update: August 30, 2021

5.4 Connect to Nebula Graph

Nebula Graph supports multiple types of clients, including a CLI client, a GUI client, and clients developed in popular programming languages. This topic provides an overview of Nebula Graph clients and basic instructions on how to use the native CLI client, Nebula Console.

5.4.1 Nebula Graph clients

You can use supported clients or console to connect to Nebula Graph.

5.4.2 Use Nebula Console to connect to Nebula Graph

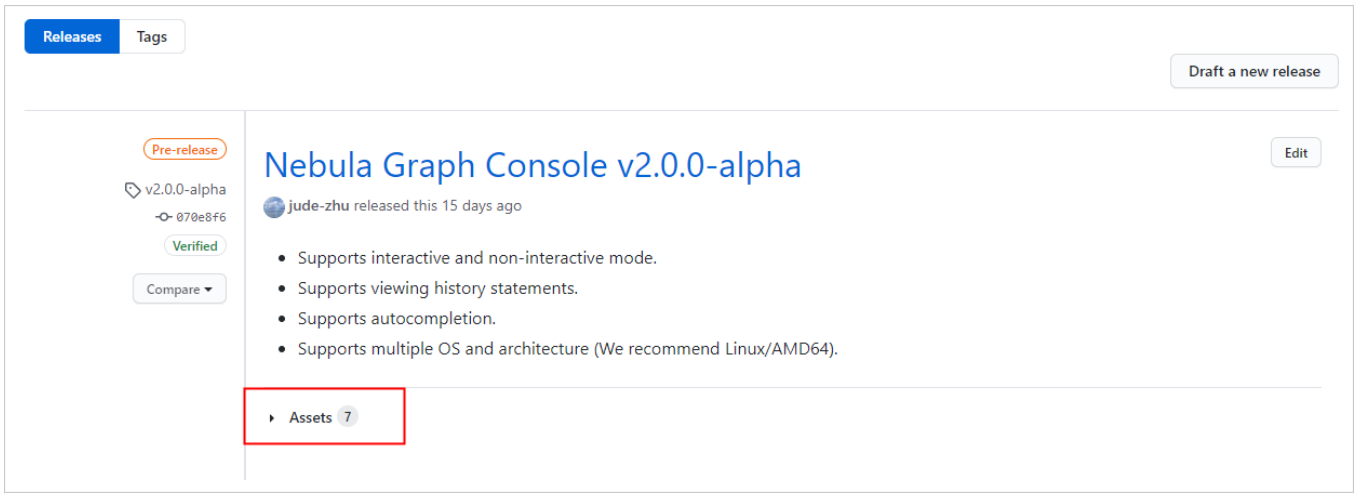
Prerequisites

- You have started the Nebula Graph services. For how to start the services, see Start and Stop Nebula Graph.
- The machine you plan to run Nebula Console on has network access to the Nebula Graph services.

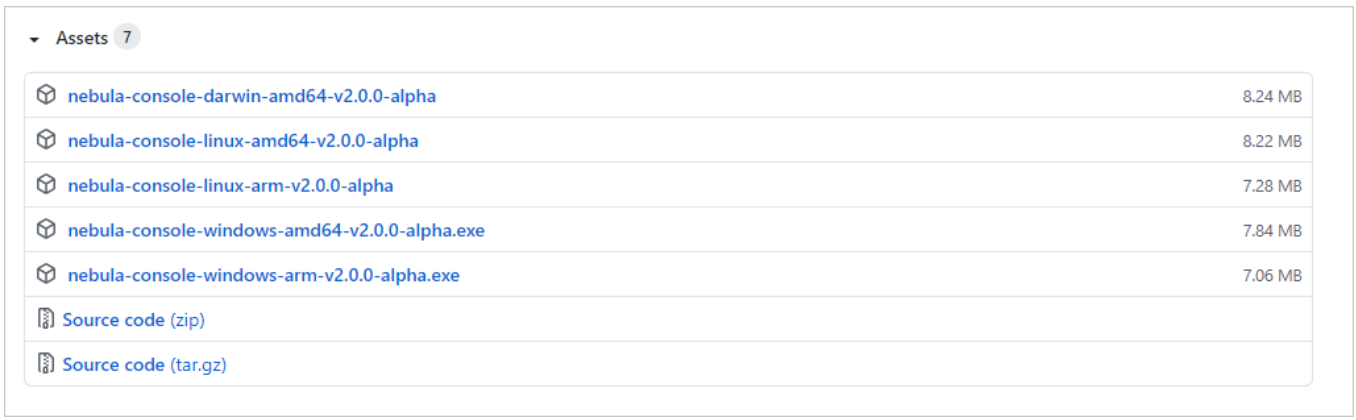
Steps

1. On the nebula-console page, select a Nebula Console version and click **Assets**.

> **Note**
>
> We recommend that you select the **latest** release.



2. In the **Assets** area, find the correct binary file for the machine where you want to run Nebula 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 Nebula 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 Nebula Graph.

   • 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]
```

The description of the parameters is as follows.

Option	Description
`-h`	Shows the help menu.
`-addr`	Sets the IP address of the graphd service. The default address is 127.0.0.1.
`-port`	Sets the port number of the graphd service. The default port number is 9669.
`-u/-user`	Sets the username of your Nebula Graph account. Before enabling authentication, you can use any existing username. The default username is `root`.
`-p/-password`	Sets the password of your Nebula Graph 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 second. 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. You'll get the return messages and the connection stops then.

You can find more details in the Nebula Console Repository.

5.4.3 Nebula Console commands

Nebula Console can export CSV file, DOT file, and import too.

> **Note**
>
> The commands are case insensitive.

Export a CSV file

> **Note**
>
> - A CSV file will be saved in the working directory, i.e., what linux command `pwd` show;
> - This command only works for the next query statement.

The command to export a csv file.

```
nebula> :CSV <file_name.csv>
```

Export a DOT file

> **Note**
>
> - A DOT file will be saved in the working directory, i.e., what linux command `pwd` show;
> - You can copy the contents of DOT file, and paste in GraphvizOnline, to visualize the excution plan;
> - This command only works for the next query statement.

The command to export a DOT file.

```
nebula> :dot <file_name.dot>
```

For example,

```
nebula> :dot a.dot
nebula> PROFILE FORMAT="dot" GO FROM "player100" OVER follow;
```

Importing a testing dataset

The testing dataset is named `nba`. Details about schema and data can be seen by commands `SHOW`.

Using the following command to import the testing dataset,

```
nebula> :play nba
```

Run a command multiple times

Sometimes, you want to run a command multiple times. Run the following command.

```
nebula> :repeat N
```

For example,

```
nebula> :repeat 3
nebula> GO FROM "player100" OVER follow;
+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 2602/3214 us)

Fri, 20 Aug 2021 06:36:05 UTC

+-------------+
| follow._dst |
+-------------+
| "player101" |
+-------------+
| "player125" |
+-------------+
Got 2 rows (time spent 583/849 us)

Fri, 20 Aug 2021 06:36:05 UTC

+-------------+
| follow._dst |
+-------------+
| "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 to wait

Sleep N seconds.

It is usually used when altering schema. Since schema is altered in async way, and take effects in the next heartbeat cycle.

```
nebula> :sleep N
```

5.4.4 Disconnect Nebula Console from Nebula Graph

You can use `:EXIT` or `:QUIT` to disconnect from Nebula Graph. For convenience, Nebula Console supports using these commands in lower case without the colon (":"), such as `quit`.

```
nebula> :QUIT

Bye root!
```

5.4.5 FAQ

How can I install Nebula Console from the source code

To download and compile the latest source code of Nebula Console, follow the instructions on the nebula console GitHub page.

Last update: August 31, 2021

5.5 Upgrade

5.5.1 Upgrade Nebula Graph to v2.5.0

The legacy versions of Nebula Graph refer to the versions lower than Nebula Graph v2.0.0-GA. This topic describes how to upgrade Nebula Graph to v2.5.0.

> **Note**
>
> To upgrade Nebula Graph v2.0.0-GA or later versions to v2.5.0, see Nebula Graph v2.0.x to v2.5.0.

Limitations

- Rolling Upgrade is not supported. You must stop the Nebula Graph 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 Nebula Graph is deployed with Docker, including Docker Swarm, Docker Compose, and K8s.
- You must upgrade the old Nebula Graph 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.
- The hard disk space of each machine should be **three times** as much as the space taken by the original data directories.
- Known issues that could cause data loss are listed on GitHub known issues. The issues are all related to altering schema or default values.
- To connect to Nebula Graph 2.0.0, you must upgrade all the Nebula Graph clients. The communication protocols of the old versions and the latest versions are not compatible.
- The upgrade takes about 30 minutes in this test environment.
- **DO NOT** use soft links to switch the data directories.
- You must have the sudo privileges to complete the steps in this topic.

Installation paths

OLD INSTALLATION PATH

By default, old versions of Nebula Graph are installed in `/usr/local/nebula/`, hereinafter referred to as `${nebula-old}`. The default configuration file path is `${nebula-old}/etc/`.

- Storaged data path is defined by the `--data_path` option in the `${nebula-old}/etc/nebula-storaged.conf` file. The default path is `data/storage`.
- Metad data path is defined by the `--data_path` option in the `${nebula-old}/etc/nebula-metad.conf` file. The default path is `data/meta`.

> **Note**
>
> The actual paths in your environment may be different from those described in this topic. You can run the Linux command `ps -ef | grep nebula` to locate them.

NEW INSTALLATION PATH

`${nebula-new}` represents the installation path of the new Nebula Graph version, such as `/usr/local/nebula-new/`.

```
# mkdir -p ${nebula-new}
```

Upgrade steps

1. **Stop all client connections**. You can run the following commands on each Graph server to turn off the Graph Service and avoid dirty write.

```
# ${nebula-old}/scripts/nebula.service stop graphd
[INFO] Stopping nebula-graphd...
[INFO] Done
```

2. Run the following commands to stop all services of the old version Nebula Graph.

```
# ${nebula-old}/scripts/nebula.service stop all
[INFO] Stopping nebula-metad...
[INFO] Done
[INFO] Stopping nebula-graphd...
[INFO] Done
[INFO] Stopping nebula-storaged...
[INFO] Done
```

   The `storaged` process needs about 1 minute to flush data. Wait 1 minute and then run `ps -ef | grep nebula` to check and make sure that all the Nebula Graph services are stopped.

> **Note**
>
> If the services are not fully stopped in 20 minutes, **stop upgrading** and go to the Nebula Graph community for help.

3. Install the new version of Nebula Graph on each machine.

   a. Install the new binary file.

   - To install with RPM/DEB packages, download the installation package of the corresponding operating system from release page.

```
# sudo rpm --force -i --prefix=${nebula-new}  ${nebula-package-name.rpm} # for centos/redhat
# sudo dpkg -i --instdir==${nebula-new} ${nebula-package-name.deb} # for ubuntu
```

     For detailed steps, see Install Nebula Graph with RPM or DEB package.

   - To install with the source code, follow the substeps. For detailed steps, see Install Nebula Graph by compiling the source code. Some key commands are as follows.

     - Clone the source code.

```
# git clone --branch v2.5.0 https://github.com/vesoft-inc/nebula-graph.git
```

     - Configure CMake.

```
# cmake -DCMAKE_INSTALL_PREFIX=${nebula-new} -DENABLE_BUILD_STORAGE=on -DENABLE_TESTING=OFF -DCMAKE_BUILD_TYPE=Release -
DNEBULA_COMMON_REPO_TAG=v2.5.0  -DNEBULA_STORAGE_REPO_TAG=v2.5.0 ..
```

   b. Copy the configuration files from the old path to the new path.

```
# cp -rf ${nebula-old}/etc ${nebula-new}/
```

4. Follow the substeps to prepare the Meta servers (usually 3 of them in a cluster).
   - Locate the old Meta data path and copy the data files to the new path.

     Find the `--data_path` option in `${nebula-old}/etc/nebula-metad.conf`. The default value is `data/meta`.

     - If the legacy versions **has not changed** the `--data_path` item, run the following command to copy the meta data to the new directory.

       ```
       # mkdir -p ${nebula-new}/data/meta/
       # cp -r ${nebula-old}/data/meta/* ${nebula-new}/data/meta/
       ```

     - If the legacy versions change the default metad directory, copy it according to the actual directory.

   - Modify the new Meta configuration files.
     - Edit the new metad configuration file.

       ```
       # vim ${nebula-new}/nebula-metad.conf
       ```

     - [Optional]Add the following parameters in the Meta configuration files if you need them.

       `--null_type=false`: Disables the support for using `NULL`.**The default value is** `true`. When set to `false`, you must specify a default value when altering tags or edge types, otherwise, data reading fails.

       `--string_index_limit=32`: Specifies the index length for string values as 32. The default length is 64.

   > **Note**
   >
   > You must make sure that this step is applied on every Meta server.

5. Prepare the Storage configuration files on each Storage server.
   - [Optional]If the old Storage data path is not the default setting `--data_path=data/storage`, modify it.

     ```
     # vim ${nebula-new}/nebula-storaged.conf
     ```

     Change the value of `--data_path` as the new data path.

   - Create the new Storage data directories.

     ```
     # mkdir -p ${nebula-new}/data/storage/
     ```

   If the `--data_path` default value has been modified, create the Storage data directories according to the modification.

6. Start the new Meta Service.
   - Run the following command on each Meta server.

     ```
     # ${nebula-new}/scripts/nebula.service start metad
     [INFO] Starting nebula-metad...
     [INFO] Done
     ```

   - Check if every nebula-metad process is started normally.

     ```
     # ps -ef |grep nebula-metad
     ```

   - Check if there is any error information in the Meta logs in `${nebula-new}/logs`.

   > **Note**
   >
   > If any nebula-metad process cannot start normally, **stop upgrading, start the Nebula Graph services from the old directories**, and take the error logs to the Nebula Graph community for help.

7. Run the following commands to upgrade the Storage data format.

   ```
   # ${nebula-new}/bin/db_upgrader  \
   --src_db_path=<old_storage_directory_path> \
   --dst_db_path=<new_storage_directory_path>  \
   ```

```
--upgrade_meta_server=<meta_server_ip1>:<port1>[,<meta_server_ip2>:<port2>,...] \
--upgrade_version=<old_nebula_version> \
```

The parameters are described as follows.

- `--src_db_path`: Specifies the absolute path of the **OLD** Storage data directories. Separate multiple paths with commas, without spaces.
- `--dst_db_path`: Specifies the absolute path of the **NEW** Storage data directories. Separate multiple paths with commas, without spaces. The paths must correspond to the paths set in `--src_db_path` one by one.
- `--upgrade_meta_server`: Specifies the addresses of the new Meta servers that you started in step 6.
- `--upgrade_version`: If the old Nebula Graph version is v1.2.0, set the parameter value to `1`. If the old version is v2.0.0-RC1, set the value to `2`. Do not set the value to other numbers.

Danger

Do not mix up the order of `--src_db_path` and `--dst_db_path`. Otherwise, the old data will be damaged during the upgrade.

For example, upgrade from v1.2.x:

```
# /usr/local/nebula_new/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage/data1/,/usr/local/nebula/data/storage/data2/ \
--dst_db_path=/usr/local/nebula_new/data/storage/data1/,/usr/local/nebula_new/data/storage/data2/\
--upgrade_meta_server=192.168.*.14:45500,192.168.*.15:45500,192.168.*.16:45500 \
--upgrade_version=1
```

For example, upgrade from v2.0.0-RC1:

```
# /usr/local/nebula_new/bin/db_upgrader \
--src_db_path=/usr/local/nebula/data/storage/ \
--dst_db_path=/usr/local/nebula_new/data/storage/ \
--upgrade_meta_server=192.168.*.14:9559,192.168.*.15:9559,192.168.*.16:9559 \
--upgrade_version=2
```

Note

- If anything goes wrong, **Stop upgrading, stop all the Meta servers, and start the Nebula Graph services from the old directories.**
- Make sure that all the Storage servers have finished the upgrade.

8. Start the new Storage Service on each Storage server.

```
# ${nebula-new}/scripts/nebula.service start storaged
# ${nebula-new}/scripts/nebula.service status storaged
```

Note

If this step goes wrong on any server, Take the logs in `${nebula-new}/logs/` to the Nebula Graph community for help. **Stop upgrading. Stop all the Meta servers and Storage servers. Start the Nebula Graph services from the old directories.**

9. Start the new Graph Service on each Graph server.

```
# ${nebula-new}/scripts/nebula.service start graphd
# ${nebula-new}/scripts/nebula.service status graphd
```

Note

If this step goes wrong on any server, take the logs in `${nebula-new}/logs/` to the Nebula Graph community for help. **Stop upgrading. Stop all the Meta servers, Storage servers, and Graph servers. Start the Nebula Graph services from the old directories.**

10. Connect to Nebula Graph with the new versions of Nebula Console. Verify if the Nebula Graph services are available and if the data can be accessed normally. Make sure that the command parameters, including the IP address and port of the Graph Service, are the same as the old one.

```
nebula> SHOW HOSTS;
nebula> SHOW SPACES;
nebula> USE <space_name>
nebula> SHOW PARTS;
nebula> SUBMIT JOB STATS;
nebula> SHOW STATS;
```

> **Note**
>
> The old releases of Nebula Console may have compatibility issues.

11. Upgrade other Nebula Graph clients.

    You must upgrade all other clients to corresponding Nebula Graph v2.5.0. The clients include but are not limited to Python, Java, go, C++, Flink-connector, Spark-util, and Nebula Bench. Find the v2.5.0 branch for each client.

> **Note**
>
> Communication protocols of v2.5.0 are not compatible with that of the old releases. To upgrade the clients, compile the v2.5.0 source code of the clients or download corresponding binaries.
>
> Tip for maintenance: The data path after the upgrade is `${nebula-new}/`. Modify relative paths for hard disk monitor systems, log, or ELK, etc.

Upgrade failure and rollback

If the upgrade fails, stop all Nebula Graph services of the new version, and start the services of the old version.

All Nebula Graph clients in use must be switched to the **old version**.

Appendix 1: Test Environment

The test environment for this topic is as follows:

- Machine specifications: 32 CPU cores, 62 GB memory, and SSD.
- Data size: 100 GB of Nebula Graph 1.2.0 LDBC test data, with 1 graph space, 24 partitions, and 92 GB of data directory size.
- Concurrent configuration: `--max_concurrent=5`, `--max_concurrent_parts=24`, and `--write_batch_num=100`.

The upgrade cost **21 minutes** in all, including 13 minutes of compaction. The descriptions are as follows.

Parameter	Default value
`--max_concurrent`	5
`--max_concurrent_parts`	10
`--write_batch_num`	100

Appendix 2: Nebula Graph V2.0.0 code address and commit ID

Code address	Commit ID
graphd	91639db
storaged and metad	761f22b
common	b2512aa

FAQ

CAN I WRITE THROUGH THE CLIENT DURING THE UPGRADE?

A: No. The state of the data written during this process is undefined.

CAN I UPGRADE OTHER OLD VERSIONS EXCEPT FOR V1.2.X AND V2.0.0-RC TO V2.5.0?

A: Upgrading from other old versions is not tested. Theoretically, versions between v1.0.0 and v1.2.0 could adopt the upgrade approach for v1.2.x. v2.0.0-RC nightly versions cannot apply the solutions in this topic.

HOW TO UPGRADE IF A MACHINE HAS ONLY THE GRAPH SERVICE, BUT NOT THE STORAGE SERVICE?

A: Upgrade the Graph Service with the corresponding binary or rpm package.

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 github issues.

WHAT ARE THE DIFFERENCES BETWEEN DELETING DATA THEN INSTALLING THE NEW VERSION AND UPGRADING ACCORDING TO THIS TOPIC?

A: The default configurations for v2.x and v1.x are different, including the ports used. The upgrade solution keeps the old configurations, and the delete-and-install solution uses the new configurations.

IS THERE A TOOL OR SOLUTION FOR VERIFYING DATA CONSISTENCY AFTER THE UPGRADE?

A: No.

Last update: September 3, 2021

5.5.2 Upgrade Nebula Graph v2.0.x to v2.5.0

To upgrade Nebula Graph v2.0.x to v2.5.0, you only need to use the RPM/DEB package of v2.5.0 for the upgrade, or compile it and then reinstall.

> **Note**
>
> Nebula Graph v2.0.x refers to v2.0.0-GA and v2.0.1 releases. If your Nebula Graph version is too low (v2.0.0-RC, v2.0.0-beta, v1.x), see Upgrade Nebula Graph to v2.5.0.

Upgrade steps with RPM/DEB packages

1. Download the RPM/DEB package.
2. Stop all Nebula Graph services. For details, see Manage Nebula Graph Service. It is recommended to back up the configuration file before updating.
3. Execute the following command to upgrade:
   - RPM package

     ```
     $ sudo rpm -Uvh <package_name>
     ```

     If you specify the path during installation, you also need to specify the path during upgrade.

     ```
     $ sudo rpm -Uvh --prefix=<installation_path> <package_name>
     ```

   - DEB package

     ```
     $ sudo dpkg -i <package_name>
     ```

4. Start the required services on each server. For details, see Manage Nebula Graph Service.

Upgrade steps by compiling the new source code

1. Back up the old version of the configuration file. The configuration file is saved in the `etc` directory of the Nebula Graph installation path.
2. Update the repository and compile the source code. For details, see Install Nebula Graph by compiling the source code.

> **Note**
>
> When compiling, set the installation path, which is the same as the installation path of the old version.

Upgrade steps by deploying Docker Compose

See How to update the Docker image of Nebula Graph services.

Last update: September 2, 2021

5.6 Uninstall Nebula Graph

This topic describes how to uninstall Nebula Graph.

> **Caution**
>
> Before re-installing Nebula Graph on a machine, follow this topic to completely uninstall the old Nebula Graph, in case the remaining data interferes with the new services, including inconsistencies between Meta services.

5.6.1 Prerequisite

The Nebula Graph services should be stopped before the uninstallation. For more information, see Manage Nebula Graph services.

5.6.2 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 Nebula Graph stores data may not be in the installation path of Nebula Graph. Check the configuration files to confirm the data paths, and then manually delete the directories to clear all data.

> **Note**
>
> For a Nebula Graph cluster, delete the data files of all Storage and Meta servers.

1. Check the Storage Service disk settings. For example:

```
########## Disk ##########
# Root data path. Split by comma. e.g. --data_path=/disk1/path1/,/disk2/path2/
# One path per Rocksdb instance.
--data_path=/nebula/data/storage
```

2. Check the Metad Service configurations and find the corresponding metadata directories.
3. Delete the data and the directories found in step 2.

5.6.3 Step 2: Delete the installation directories

> **Note**
>
> Delete all installation directories, including the `cluster.id` file in them.

The default installation path is `/usr/local/nebula`, which is specified by `--prefix` while installing Nebula Graph.

Uninstall Nebula Graph deployed with source code

Find the installation directories of Nebula Graph, and delete them all.

Uninstall Nebula Graph deployed with RPM packages

1. Run the following command to get the Nebula Graph version.

```
$ rpm -qa | grep "nebula"
```

The return message is as follows.

```
nebula-graph-2.5.0-1.x86_64
```

2. Run the following command to uninstall Nebula Graph.

```
sudo rpm -e <nebula_version>
```

For example:

```
sudo rpm -e nebula-graph-2.5.0-1.x86_64
```

3. Delete the installation directories.

Uninstall Nebula Graph deployed with DEB packages

1. Run the following command to get the Nebula Graph version.

```
$ dpkg -l | grep "nebula"
```

The return message is as follows.

```
ii  nebula-graph  2.5.0  amd64     Nebula Package built using CMake
```

2. Run the following command to uninstall Nebula Graph.

```
sudo dpkg -r <nebula_version>
```

For example:

```
sudo dpkg -r nebula-graph
```

3. Delete the installation directories.

Uninstall Nebula Graph deployed with Docker Compose

1. In the `nebula-docker-compose` directory, run the following command to stop the Nebula Graph services.

```
docker-compose down -v
```

2. Delete the `nebula-docker-compose` directory.

Last update: August 31, 2021

6. Configurations and logs

6.1 Configurations

6.1.1 Configurations

Nebula Graph builds the configurations based on the gflags repository. Most configurations are flags. When the Nebula Graph service starts, it will get the configuration information from Configuration files by default. Configurations that are not in the file apply the default values.

> **Note**
>
> - Because there are many configurations and they may change as Nebula Graph 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.

> **Legacy 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 in the topic of 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:

```
<binary> --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 Nebula Graph, use the actual path to query the configurations.

Get configurations

Use the `curl` command to get the value of the running configurations.

> **Legacy version compatibility**
>
> The `curl` commands and parameters in Nebula Graph v2.x. are different from Nebula Graph v1.x.

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
```

> **Note**
>
> In an actual environment, use the real host IP address instead of `127.0.0.1` in the above example.

Configuration files

Nebula Graph provides two initial configuration files for each service, `<service_name>.conf.default` and `<service_name>.conf.production`. Users can use them in different scenarios conveniently. The default path is `/usr/local/nebula/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.

> **Caution**
>
> To ensure the availability of services, the configurations of the same service must be consistent, except for the local IP address `local_ip`. For example, three Storage servers are deployed in one Nebula Graph cluster. The configurations of the three Storage servers need to be the same, except for the IP address.

The initial configuration files corresponding to each service are as follows.

Nebula Graph service	Initial configuration file	Description
Meta	`nebula-metad.conf.default` and `nebula-metad.conf.production`	Meta service configuration
Graph	`nebula-graphd.conf.default` and `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 Nebula Graph 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 Nebula Graph service will first read the cached configurations, which may cause configuration inconsistencies between clusters and cause unknown risks.

Modify configurations

By default, each Nebula Graph service gets configurations from its configuration files. Users can modify configurations and make them valid according to the following steps:

1. Use a text editor to modify the configuration files of the target service and save the modification.
2. Choose an appropriate time to restart **all** Nebula Graph services to make the modifications valid.

Last update: September 2, 2021

6.1.2 Meta Service configuration

Nebula Graph 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 Nebula Graph 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, Nebula Graph 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 `nebula-metad.conf.default`.

For all parameters and their current values, see Configurations.

Basics configurations

Name	Predefined value	Description
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids/nebula-metad.pid`	The file that records the process ID.
`timezone_name`	-	Specifies the Nebula Graph 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=CST-8` represents the GMT+8 time zone.
`local_config`	`true`	When set to `true`, the process gets configurations from the configuration files.
`minimum_reserved_bytes`	-	Specifies the minimum remaining space of each data storage path. When the value is lower than this standard, the cluster metadata operation may fail. This configuration is measured in bytes. The default value is `1073741824`, namely, 1GB.

Note

- While inserting property values of time types, Nebula Graph transforms them to a UTC time 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 Nebula Graph. Other time-related data of the Nebula Graph processes still uses the default time zone of the host, such as the log printing time.

Logging configurations

Name	Predefined value	Description
`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.
`minloglevel`	`0`	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`, Nebula Graph will not print any logs.
`v`	`0`	Specifies the detailed level of the log. The larger the value, the more detailed the log is. Optional values are `0`, `1`, `2`, `3`.
`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.
`redirect_stdout`	`true`	When set to `true`, the process redirects the `stdout` and `stderr` to separate output files.
`stdout_log_file`	`metad-stdout.log`	Specifies the filename for the `stdout` log.
`stderr_log_file`	`metad-stderr.log`	Specifies the filename for the `stderr` log.
`stderrthreshold`	`2`	Specifies the `minloglevel` to be copied to the `stderr` log.

Networking configurations

Name	Predefined value	Description
`meta_server_addrs`	`127.0.0.1:9559`	Specifies the IP addresses and ports of all Meta Services. Multiple addresses are separated with commas.
`local_ip`	`127.0.0.1`	Specifies the local IP 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.
`port`	`9559`	Specifies RPC daemon listening port of the Meta service. The external port for the Meta Service is predefined to `9559`. The internal port is predefined to `port + 1`, i.e., `9560`. Nebula Graph uses the internal port for multi-replica interactions.
`ws_ip`	`0.0.0.0`	Specifies the IP address for the HTTP service.
`ws_http_port`	`19559`	Specifies the port for the HTTP service.
`ws_h2_port`	`19560`	Specifies the port for the HTTP2 service.
`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.
`heartbeat_interval_secs`	`10`	Specifies the default heartbeat interval. Make sure the `heartbeat_interval_secs` values for all services are the same, otherwise Nebula Graph **CANNOT** work normally. This configuration is measured in seconds.

> **Caution**
>
> The real IP address must be used in the configuration file. Otherwise, `127.0.0.1/0.0.0.0` cannot be parsed correctly in some cases.

Storage configurations

Name	Predefined Value	Description
`data_path`	`data/meta`	The storage path for Meta data.

Misc configurations

Name	Predefined Value	Description
`default_parts_num`	`100`	Specifies the default partition number when creating a new graph space.
`default_replica_factor`	`1`	Specifies the default replica number when creating a new graph space.

RocksDB options configurations

Name	Predefined Value	Description
`rocksdb_wal_sync`	`true`	Enables or disables RocksDB WAL synchronization. Available values are `true` (enable) and `false` (disable).

Last update: September 2, 2021

6.1.3 Graph Service configuration

Nebula Graph 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 Nebula Graph 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, Nebula Graph 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 `nebula-metad.conf.default`.

For all parameters and their current values, see Configurations.

Basics configurations

Name	Predefined value	Description
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids/nebula-graphd.pid`	The file that records the process ID.
`enable_optimizer`	`true`	When set to `true`, the optimizer is enabled.
`system_memory_high_watermark_ratio`	-	Specifies the trigger threshold of the high-level memory alarm mechanism. The default value is `0.8`. If the system memory usage is higher than this value, an alarm mechanism will be triggered, and Nebula Graph will stop querying. This parameter is not predefined in the initial configuration files.
`timezone_name`	-	Specifies the Nebula Graph 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.
`local_config`	`true`	When set to `true`, the process gets configurations from the configuration files.

Note

- While inserting property values of time types, Nebula Graph transforms them to a UTC time 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 Nebula Graph. Other time-related data of the Nebula Graph processes still uses the default time zone of the host, such as the log printing time.

Logging configurations

Name	Predefined value	Description
`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.
`minloglevel`	`0`	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`, Nebula Graph will not print any logs.
`v`	`0`	Specifies the detailed level of the log. The larger the value, the more detailed the log is. Optional values are `0`, `1`, `2`, `3`.
`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.
`redirect_stdout`	`true`	When set to `true`, the process redirects the `stdout` and `stderr` to separate output files.
`stdout_log_file`	`graphd-stdout.log`	Specifies the filename for the `stdout` log.
`stderr_log_file`	`graphd-stderr.log`	Specifies the filename for the `stderr` log.
`stderrthreshold`	`2`	Specifies the `minloglevel` to be copied to the `stderr` log.

Query configurations

Name	Predefined value	Description
`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.
`session_reclaim_interval_secs`	`10`	Specifies the interval that the Session information is sent to the Meta service. This configuration is measured in seconds.

Networking configurations

Name	Predefined value	Description
`meta_server_addrs`	`127.0.0.1:9559`	Specifies the IP addresses and ports of all Meta Services. Multiple addresses are separated with commas.
`local_ip`	`127.0.0.1`	Specifies the local IP 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.
`listen_netdev`	`any`	Specifies the listening network device.
`port`	`9669`	Specifies RPC daemon listening port of the Graph service.
`reuse_port`	`false`	When set to `false`, the `SO_REUSEPORT` is closed.
`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`.
`client_idle_timeout_secs`	`0`	Specifies the time to expire an idle connection. `0` means that the connection will never expire. This configuration is measured in seconds.
`session_idle_timeout_secs`	`0`	Specifies the time to expire an idle session. `0` means that the session will never expire. This configuration is measured in seconds.
`num_accept_threads`	`1`	Specifies the number of threads that accept incoming connections.
`num_netio_threads`	`0`	Specifies the number of networking IO threads. `0` is the number of CPU cores.
`num_worker_threads`	`0`	Specifies the number of threads that execute queries. `0` is the number of CPU cores.
`ws_ip`	`0.0.0.0`	Specifies the IP address for the HTTP service.
`ws_http_port`	`19669`	Specifies the port for the HTTP service.
`ws_h2_port`	`19670`	Specifies the port for the HTTP2 service.
`heartbeat_interval_secs`	`10`	Specifies the default heartbeat interval. Make sure the `heartbeat_interval_secs` values for all services are the same, otherwise Nebula Graph **CANNOT** work normally. This configuration is measured in seconds.
`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.
`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.

Caution

The real IP address must be used in the configuration file. Otherwise, `127.0.0.1/0.0.0.0` cannot be parsed correctly in some cases.

Charset and collate configurations

Name	Predefined value	Description
`default_charset`	`utf8`	Specifies the default charset when creating a new graph space.
`default_collate`	`utf8_bin`	Specifies the default collate when creating a new graph space.

Authorization configurations

Name	Predefined value	Description
`enable_authorize`	`false`	When set to `false`, the system authentication is not enabled. For more information, see Authentication.
`auth_type`	`password`	Specifies the login method. Available values are `password`, `ldap`, and `cloud`.

Last update: September 2, 2021

6.1.4 Storage Service configurations

Nebula Graph provides two initial configuration files for the Storage Service, `nebula-storaged.conf.default` and `nebula-storaged.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 Nebula Graph 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, Nebula Graph 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 `nebula-metad.conf.default`.

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
`daemonize`	`true`	When set to `true`, the process is a daemon process.
`pid_file`	`pids/nebula-storaged.pid`	The file that records the process ID.
`timezone_name`	-	Specifies the Nebula Graph 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=CST-8` represents the GMT+8 time zone.
`local_config`	`true`	When set to `true`, the process gets configurations from the configuration files.

Note

- While inserting property values of time types, Nebula Graph transforms them to a UTC time 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 Nebula Graph. Other time-related data of the Nebula Graph processes still uses the default time zone of the host, such as the log printing time.

Logging configurations

Name	Predefined value	Description
`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.
`minloglevel`	`0`	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`, Nebula Graph will not print any logs.
`v`	`0`	Specifies the detailed level of the log. The larger the value, the more detailed the log is. Optional values are `0`, `1`, `2`, `3`.
`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.
`redirect_stdout`	`true`	When set to `true`, the process redirects the `stdout` and `stderr` to separate output files.
`stdout_log_file`	`graphd-stdout.log`	Specifies the filename for the `stdout` log.
`stderr_log_file`	`graphd-stderr.log`	Specifies the filename for the `stderr` log.
`stderrthreshold`	`2`	Specifies the `minloglevel` to be copied to the `stderr` log.

Networking configurations

Name	Predefined value	Description
`meta_server_addrs`	`127.0.0.1:9559`	Specifies the IP addresses and ports of all Meta Services. Multiple addresses are separated with commas.
`local_ip`	`127.0.0.1`	Specifies the local IP 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.
`port`	`9779`	Specifies RPC daemon listening port of the Storage service. The external port for the Meta Service is predefined to `9779`. The internal port is predefined to `9777`, `9778`, and `9780`. Nebula Graph uses the internal port for multi-replica interactions.
`ws_ip`	`0.0.0.0`	Specifies the IP address for the HTTP service.
`ws_http_port`	`19779`	Specifies the port for the HTTP service.
`ws_h2_port`	`19780`	Specifies the port for the HTTP2 service.
`heartbeat_interval_secs`	`10`	Specifies the default heartbeat interval. Make sure the `heartbeat_interval_secs` values for all services are the same, otherwise Nebula Graph **CANNOT** work normally. This configuration is measured in seconds.

Caution

The real IP address must be used in the configuration file. Otherwise, `127.0.0.1/0.0.0.0` cannot be parsed correctly in some cases.

Raft configurations

Name	Predefined value	Description
raft_heartbeat_interval_secs	30	Specifies the time to expire the Raft election. The configuration is measured in seconds.
raft_rpc_timeout_ms	500	Specifies the time to expire the Raft RPC. The configuration is measured in milliseconds.
wal_ttl	14400	Specifies the lifetime of the RAFT WAL. The configuration is measured in seconds.

Disk configurations

Name	Predefined value	Description
data_path	data/storage	Specifies the data storage path. Multiple paths are separated with comm RocksDB example corresponds to one path.
minimum_reserved_bytes	268435456	Specifies the minimum remaining space of each data storage path. Whe lower than this standard, the cluster data writing may fail. This configur measured in bytes. The default value is 1073741824, namely, 1GB.
rocksdb_batch_size	4096	Specifies the block cache for a batch operation. The configuration is mea
rocksdb_block_cache	4	Specifies the block cache for BlockBasedTable. The configuration is mea megabytes.
engine_type	rocksdb	Specifies the engine type.
rocksdb_compression	lz4	Specifies the compression algorithm for RocksDB. Optional values are lz4, lz4hc, zlib, bzip2, and zstd.
rocksdb_compression_per_level	\	Specifies the compression algorithm for each level.
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 kExceptHistog kExceptTimers, kExceptDetailedTimers, kExceptTimeForMutex, and kAll.
enable_rocksdb_prefix_filtering	false	When set to true, the prefix bloom filter for RocksDB is enabled. Enabli filter makes the graph traversal faster.
rocksdb_filtering_prefix_length	12	Specifies the prefix length for each key. Optional values are 12 and 16. configuration is measured in bytes.
enable_partitioned_index_filter	-	When set to true, it reduces the amount of memory used by the bloom f some random-seek situations, it may reduce the read performance.

RocksDB options

Name	Predefined value	Description
rocksdb_db_options	{}	Specifies the RocksDB database options.
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.
rocksdb_block_based_table_options	{"block_size":"8192"}	Specifies the RocksDB block based table options.

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_whole_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
level0_stop_writes_trigger
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.

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 super-large 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
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 configuration files.

Compatibility

The reservoir sampling algorithm in Nebula Graph 1.x is no longer supported in Nebula Graph 2.5.0.

Storage configurations for large dataset

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: September 2, 2021

6.1.5 Kernel configurations

This topic introduces the Kernel configurations in Nebula Graph.

Resource control

ULIMIT PRECAUTIONS

The `ulimit` command specifies the resource threshold for the current shell session. The precautions are as follows:

- The changes made by `ulimit` only take effect for the current session or child process.
- The resource threshold (soft threshold) cannot exceed the hard threshold.
- Common users cannot use commands to adjust the hard threshold, even with `sudo`.
- To modify the system level or adjust the hard threshold, edit the file `/etc/security/limits.conf`. This method requires re-login to take effect.

ULIMIT -C

`ulimit -c` limits the size of the core dumps. We recommend that you set it to `unlimited`. The command is:

```
ulimit -c unlimited
```

ULIMIT -N

`ulimit -n` limits the number of open files. We recommend that you set it to more than 100,000. For example:

```
ulimit -n 130000
```

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 PAGE

For better delay performance, you must disable the transparent huge pages (THP). The command is:

```
root# echo never > /sys/kernel/mm/transparent_hugepage/enabled
root# echo never > /sys/kernel/mm/transparent_hugepage/defrag
root# swapoff -a && swapon -a
```

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: September 2, 2021

6.2 Log management

6.2.1 Logs

Nebula Graph uses glog to print 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 log directory is `/usr/local/nebula/logs/`.

If the log directory is deleted while Nebula Graph 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

- `minloglevel`: 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`, Nebula Graph 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
```

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 Nebula Graph 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 logs

RocksDB 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.

Last update: September 2, 2021

7. Monitor and metrics

7.1 Query Nebula Graph metrics

Nebula Graph supports querying the monitoring metrics through HTTP ports.

7.1.1 Metrics

Each metric of Nebula Graph consists of three fields: name, type, and time range. The fields are separated by periods, for example, `num_queries.sum.600`. Different Nebula Graph 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, COUNT, AVG, RATE, and the P-th sample quantiles such as P75, P95, P99, and P99.9.
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://<ip>:<port>/stats?stats=<metric_name_list> [&format=json]"
```

Parameter	Description
`ip`	The IP address 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 Nebula Graph 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.

Examples

- Query a single metric

  Query the query number in the last 10 minutes in the Graph Service.

```
$ curl -G "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 -G "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 -G "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, Nebula Graph 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
  ```

Last update: September 2, 2021

8. Data security

8.1 Authentication and authorization

8.1.1 Authentication

Nebula Graph replies on local authentication or LDAP authentication to implement access control.

Nebula Graph creates a session when a client connects to it. The session stores information about the connection, including the user information.

By default, authentication is disabled and Nebula Graph allows connections with the username `root` and any password. If the authentication system is enabled, Nebula Graph checks a session according to the authentication configuration, and decides whether the session should be allowed or denied.

Local authentication

Local authentication indicates that usernames and passwords are stored locally on the server, with the passwords encrypted.

ENABLE LOCAL AUTHENTICATION

1. In the `/usr/local/nebula/etc/nebula-graphd.conf` file, set `--enable_authorize=true` and save the modification.

> **Note**
>
> `/usr/local/nebula/` is the default installation path for Nebula Graph. If you have changed it, use the actual path.

2. Restart the Nebula Graph services. For how to restart, see Manage Nebula Graph services.

> **Note**
>
> You can use the username `root` and password `nebula` to log into Nebula Graph after enabling local authentication. This account has the build-in God role. For more information about roles, see Roles and privileges.

LDAP authentication

> **Enterpriseonly**
>
> LDAP authentication is an Enterprise-only feature. For how to enable LDAP, see Authenticate with an LDAP server (TODO: doc).

Lightweight Directory Access Protocol (LDAP), is a lightweight client-server protocol for accessing directories and building a centralized account management system.

LDAP authentication and local authentication can be enabled at the same time, but LDAP authentication has a higher priority. If the local authentication server and the LDAP server both have the information of user `Amber`, Nebula Graph reads from the LDAP server first.

Last update: September 1, 2021

8.1.2 User management

This topic describes how to manage users and roles.

By default, Nebula Graph allows connections with the username `root` and any password. After enabling authentication, only valid users can connect to Nebula Graph and access the resources according to the user roles.

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 GOD role can run `CREATE USER` to create a new user.

- Syntax

```
CREATE USER [IF NOT EXISTS] <user_name> [WITH PASSWORD '<password>'];
```

- Example

```
nebula> CREATE USER user1 WITH PASSWORD 'nebula';
```

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 Nebula Graph built-in roles, see Roles and privileges

> **Note**
>
> If the target user is connected to Nebula Graph when running `GRANT ROLE`, the new role takes effect when the user logs out and logs in again.

- Syntax

```
GRANT ROLE <role_type> ON <space_name> TO <user_name>;
```

- Example

```
nebula> GRANT ROLE USER ON basketballplayer TO user1;
```

SHOW ROLES

Run `SHOW ROLES` to list the roles in a graph space.

- Syntax

```
SHOW ROLES IN <space_name>;
```

- Example

```
nebula> SHOW ROLES IN basketballplayer;
+---------+-----------+
| Account | Role Type |
+---------+-----------+
| "user1" | "ADMIN"   |
+---------+-----------+
```

REVOKE ROLE

Users with the GOD role or the ADMIN role can run `REVOKE ROLE` to revoke a user's role in a graph space.

> **Note**
>
> If the target user is connected to Nebula Graph when running `REVOKE ROLE`, the old role still takes effect until the user logs out.

- Syntax

```
REVOKE ROLE <role_type> ON <space_name> FROM <user_name>;
```

- Example

```
nebula> REVOKE ROLE USER ON basketballplayer FROM user1;
```

CHANGE PASSWORD

With the correct username and password, users can run `CHANGE PASSWORD` to set a new password for a user.

- Syntax

```
CHANGE PASSWORD <user_name> FROM '<old_password>' TO '<new_password>';
```

- Example

```
nebula> CHANGE PASSWORD user1 FROM 'nebula' TO 'nebula123';
```

ALTER USER

The `root` user with the GOD role can run `ALTER USER` to set a new password for a user.

- Syntax

```
ALTER USER <user_name> WITH PASSWORD '<password>';
```

- Example

```
nebula> ALTER USER user1 WITH PASSWORD 'nebula';
```

DROP USER

The `root` user with the GOD role can run `DROP USER` to remove a user.

> **Note**
>
> Removing a user does not close the user's current session, 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 GOD role can run `SHOW USERS` to list all the users.

- Syntax

```
SHOW USERS;
```

- Example

```
nebula> SHOW USERS;
+-----------+
| Account   |
+-----------+
| "test1"   |
+-----------+
| "test2"   |
+-----------+
| "test3"   |
+-----------+
```

Last update: September 1, 2021

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

Nebula Graph 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`.

> **Note**
>
> Modify the password for `root` as soon as possible for security.

  - The default username `root` is immutable.
  - If authentication is disabled, you can use any username and password to connect to Nebula Graph. This user is regarded as the GOD role.
- 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.
- 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.
  - The Schema information is 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**
>
> A user can have only one role in a graph space.

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`
Write space	Y					`CREATE SPACE`, `DROP SPACE`, `CREATE SNAPSHOT`, `DROP SNAPSHOT`, `BALANCE DATA`, `BALANCE DATA STOP`, `BALANCE DATA REMOVE`, `BALANCE LEADER`, `ADMIN`, `CONFIG`, `INGEST`, `DOWNLOAD`, `BUILD TAG INDEX`, `BUILD EDGE INDEX`
Read schema	Y	Y	Y	Y	Y	`DESCRIBE TAG`, `DESCRIBE EDGE`, `DESCRIBE TAG INDEX`, `DESCRIBE EDGE INDEX`
Write schema	Y	Y	Y			`CREATE TAG`, `ALTER TAG`, `CREATE EDGE`, `ALTER EDGE`, `DROP 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`

Note

- 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: February 18, 2022

8.1.4 OpenLDAP authentication

Last update: September 16, 2021

8.2 Backup and restore data with snapshots

Nebula Graph supports using snapshots to backup and restore data.

8.2.1 Authentication and snapshots

Nebula Graph 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 function. For more information about roles, see Roles and privileges.

8.2.2 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.
- Nebula Graph 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 the snapshots is not supported for now.

8.2.3 Snapshot form and path

Nebula Graph snapshots are in the form of directories with names like `SNAPSHOT_2021_03_09_08_43_12`. The suffix `2021_03_09_08_43_12` is generated automatically based on the creation time.

When a snapshot is created, snapshot directories will be automatically created in the `checkpoints` directory on the leader Meta server and each Storage server.

To fast locate the path where the snapshots are stored, you can use the Linux command `find`. For example:

```
$ find |grep 'SNAPSHOT_2021_03_11_07_30_36'
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_11_07_30_36
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_11_07_30_36/data
./data/meta2/nebula/0/checkpoints/SNAPSHOT_2021_03_11_07_30_36/data/000081.sst
...
```

> **Note**
>
> For how to get the snapshot name, see View snapshots.

8.2.4 Create a snapshot

Run `CREATE SNAPSHOT` to create a snapshot for all the graph spaces based on the current time for Nebula Graph.

> **Note**
>
> Creating a snapshot for a specific graph space is not supported yet.

If the creation fails, delete the snapshot and try again. If it still fails, go to the Nebula Graph community for help.

8.2.5 View snapshots

To view all existing snapshots, run `SHOW SNAPSHOTS`.

```
nebula> SHOW SNAPSHOTS;
+---------------------------------+---------+------------------+
| Name                            | Status  | Hosts            |
```

```
+--------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_08_43_12" | "VALID" | "127.0.0.1:9779" |
+--------------------------------+---------+------------------+
| "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`	Name of the snapshot directory.
`Status`	Status of the snapshot. `VALID` indicates that the creation succeeded and `INVALID` indicates that it failed.
`Hosts`	IP addresses and ports of all Storage servers at the time the snapshot was created.

8.2.6 Delete a snapshot

To delete a snapshot, use the following syntax:

```
DROP SNAPSHOT <snapshot_name>;
```

Example:

```
nebula> DROP SNAPSHOT SNAPSHOT_2021_03_09_08_43_12;
nebula> SHOW SNAPSHOTS;
+--------------------------------+---------+------------------+
| Name                           | Status  | Hosts            |
+--------------------------------+---------+------------------+
| "SNAPSHOT_2021_03_09_09_10_52" | "VALID" | "127.0.0.1:9779" |
+--------------------------------+---------+------------------+
```

8.2.7 Restore data with a snapshot

1. Find the snapshot directories you want to use for data restoration.
2. Choose an approach to restore the data files:
   - Change the `data_path` in the Meta configuration and Storage configuration to the snapshot path.
   - Copy the snapshot directories to other locations, and change the `data_path` to these locations.
   - Copy all the content in the snapshot directories into the directories where the `checkpoints` directories are located, and cover the existing files that have duplicate names with them. For example, cover `/usr/local/nebula/data/meta/nebula/0/data` with `/usr/local/nebula/data/meta/nebula/0/checkpoints/SNAPSHOT_2021_03_09_09_10_52/data`.
3. Restart Nebula Graph.

Last update: September 1, 2021

9. Service Tuning

9.1 Compaction

This document gives some information about compaction.

9.1.1 Introduction to compaction

In Nebula Graph, compaction is the most important background process. Compaction 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 to make the data easier to read. The read performance can increase by times after compaction. Thus, to get high read performance, trigger compaction manually when writing a large amount of data into Nebula Graph. Note that compaction leads to long time hard disk IO, we suggest that you do compaction during off-peak hours (for example, early morning).

Nebula Graph has two types of compaction: automatic compaction and full compaction.

9.1.2 Automatic compaction

Automatic compaction is done when the system reads data, writes data, or the system restarts. Automatic compaction is enabled by default.

9.1.3 Full compaction

Full compaction enables large scale background operations for a graph space such as merging files, deleting the data expired by TTL. Use these statements to enable full compaction:

```
nebula> USE <your_graph_space>;
nebula> SUBMIT JOB COMPACT;
```

The preceding statement returns a job_id. To show the compaction progress, use this statement:

```
nebula> SHOW JOB <job_id>;
```

> **Note**
>
> Do the full compaction during off-peak hours because full compaction has a lot of IO operations.

9.1.4 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, for example, early morning.
- To control the read and write traffic limitation for compactions, set the following parameter in the `nebula-storaged.conf` configuration file.

```
# Limit the read/write rate to 20MB/s.
--rate_limit=20 (in MB/s)
```

9.1.5 FAQ

Q: Can I do full compactions for multiple graph spaces at the same time?

A: Yes, you can. But the IO is much larger at this time.

Q: How much time does it take for full compactions?

A: When `rate_limit` is set to `20`, you can estimate the full compaction time by dividing the hard disk usage by the `rate_limit`. If you do not set the `rate_limit` value, the empirical value is around 50 MB/s.

Q: Can I modify `--rate_limit` dynamically?

A: No, you cannot.

Q: Can I stop a full compaction after it starts?

A: 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: September 2, 2021

9.2 Storage load balance

You can use the `BALANCE` statements to balance the distribution of partitions and Raft leaders, or remove redundant Storage servers.

9.2.1 Prerequisites

The graph spaces stored in Nebula Graph must have more than one replicas for the system to balance the distribution of partitions and Raft leaders.

9.2.2 Balance partition distribution

`BALANCE DATA` starts a task to equally distribute the storage partitions in a Nebula Graph cluster. A group of subtasks will be created and implemented to migrate data and balance the partition distribution.

> **Danger**
>
> DON'T stop any machine in the cluster or change its IP address until all the subtasks finish. Otherwise, the follow-up subtasks fail.

Take scaling out Nebula Graph for an example.

After you add new storage hosts into the cluster, no partition is deployed on the new hosts. You can run `SHOW HOSTS` to check the partition distribution.

```
nebual> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "basketballplayer:4"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "basketballplayer:8"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:15"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition"             | "No valid partition"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition"             | "No valid partition"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer:45"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
```

Run `BALANCE DATA` to start balancing the storage partitions. If the partitions are already balanced, `BALANCE DATA` fails.

```
nebula> BALANCE DATA;
+------------+
| ID         |
+------------+
| 1614237867 |
+------------+
```

A BALANCE task ID is returned after running `BALANCE DATA`. Run `BALANCE DATA <balance_id>` to check the status of the `BALANCE` task.

```
nebula> BALANCE DATA 1614237867;
+------------------------------------------------------------+-------------------+
| balanceId, spaceId:partId, src->dst                        | status            |
+------------------------------------------------------------+-------------------+
| "[1614237867, 11:1, storaged1:9779->storaged3:9779]"       | "SUCCEEDED"       |
+------------------------------------------------------------+-------------------+
| "[1614237867, 11:1, storaged2:9779->storaged4:9779]"       | "SUCCEEDED"       |
+------------------------------------------------------------+-------------------+
| "[1614237867, 11:2, storaged1:9779->storaged3:9779]"       | "SUCCEEDED"       |
+------------------------------------------------------------+-------------------+
...
+------------------------------------------------------------+-------------------+
| "Total:22, Succeeded:22, Failed:0, In Progress:0, Invalid:0" | 100               |
+------------------------------------------------------------+-------------------+
```

When all the subtasks succeed, the load balancing process finishes. Run `SHOW HOSTS` again to make sure the partition distribution is balanced.

> **Note**
>
> `BALANCE DATA` does not balance the leader distribution.

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 4            | "basketballplayer:4"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 8            | "basketballplayer:8"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 0            | "No valid partition"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 0            | "No valid partition"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer:45"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
```

If any subtask fails, run `BALANCE DATA` again to restart the balancing. If redoing load balancing does not solve the problem, ask for help in the Nebula Graph community.

9.2.3 Stop data balancing

To stop a balance task, run `BALANCE DATA STOP`.

- If no balance task is running, an error is returned.
- If a balance task is running, the task ID is returned.

`BALANCE DATA STOP` does not stop the running subtasks but cancels all follow-up subtasks. The running subtasks continue.

To check the status of the stopped balance task, run `BALANCE DATA <balance_id>`.

Once all the subtasks are finished or stopped, you can run `BALANCE DATA` again to balance the partitions again.

- If any subtask of the preceding balance task failed, Nebula Graph restarts the preceding balance task.
- If no subtask of the preceding balance task failed, Nebula Graph starts a new balance task.

9.2.4 RESET a balance task

If a balance task fails to be restarted after being stopped, run `BALANCE DATA RESET PLAN` to reset the task. After that, run `BALANCE DATA` again to start a new balance task.

9.2.5 Remove storage servers

To remove specific storage servers and scale in the Storage Service, use the `BALANCE DATA REMOVE <host_list>` syntax.

For example, to remove the following storage servers:

Server name	IP	Port
storage3	192.168.0.8	19779
storage4	192.168.0.9	19779

Run the following statement:

```
BALANCE DATA REMOVE 192.168.0.8:19779,192.168.0.9:19779;
```

Nebula Graph will start a balance task, migrate the storage partitions in storage3 and storage4, and then remove them from the cluster.

> **Note**
>
> The removed server's state will change to `OFFLINE`.

9.2.6 Balance leader distribution

`BALANCE DATA` only balances the partition distribution. If the raft leader distribution is not balanced, some of the leaders may overload. To load balance the raft leaders, run `BALANCE LEADER`.

```
nebula> BALANCE LEADER;
```

Run `SHOW HOSTS` to check the balance result.

```
nebula> SHOW HOSTS;
+-------------+------+----------+--------------+----------------------------------+------------------------+
| Host        | Port | Status   | Leader count | Leader distribution              | Partition distribution |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged0" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged1" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged2" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged3" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "storaged4" | 9779 | "ONLINE" | 3            | "basketballplayer:3"             | "basketballplayer:9"   |
+-------------+------+----------+--------------+----------------------------------+------------------------+
| "Total"     |      |          | 15           | "basketballplayer:15"            | "basketballplayer:45"  |
+-------------+------+----------+--------------+----------------------------------+------------------------+
```

Last update: September 2, 2021

9.3 Graph data modeling suggestions

This section provides general suggestions for modeling data in Nebula Graph.

> **Note**
>
> The following suggestions may not apply to some special scenarios. In these cases, find help in the Nebula Graph community.

9.3.1 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 re-design 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.

No predefined bonds between Tags and Edge types

Define the bonds between Tags and Edge types in the application, not Nebula Graph. 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 Nebula Graph Schema.

Control changes in the business model and the data model

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.

Nebula Graph 2.5.0 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.

Breadth-first traversal over depth-first traversal

Nebula Graph 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 Nebula Graph 2.5.0, 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 syntax `(dst)<-[edge]-(src)` or `GO FROM dst REVERSELY`.

If you don't care about the directions or want to query against both directions, use the 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 performance decrease by 90% or even more. Only use an index when you need to find vertices or edges through their properties.

Design VIDs appropriately

See VID.

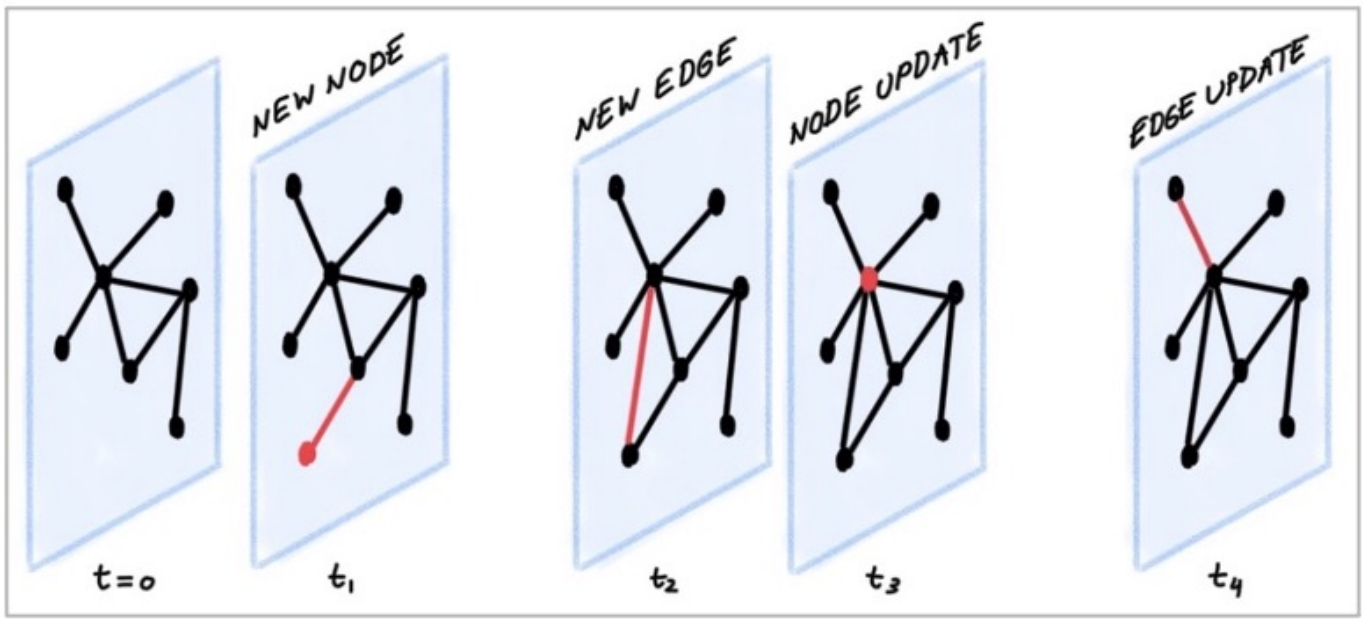
Long texts

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 Nehula Graph.

9.3.2 Dynamic graphs (sequence graphs) are not supported

In some scenarios, graphs need to have time information to describe how the structure of the entire graph changes over time.[1]

The Rank field on Edges in Nebula Graph 2.5.0 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 Nebula Graph does not support sequence graphs.



1. https://blog.twitter.com/engineering/en_us/topics/insights/2021/temporal-graph-networks ↵

Last update: September 2, 2021

9.4 System design suggestions

9.4.1 QPS or low-latency first

Nebula Graph is good at handling small requests with high concurrency. In such scenarios, the whole graph is huge, containing maybe trillions of vertices, 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 send them to multiple Graph servers. This can decrease the memory used by each Graph server as well. Besides, you can use Nebula Algorithm for such scenarios.

9.4.2 Horizontal or vertical scaling

Nebula Graph 2.5.0 supports horizontal scaling.

- The horizontal scaling of the Storage Service:

  Increasing the number of Storage machines increases the overall capability of the cluster linearly, including increasing overall QPS and reducing latency.

  However, the number of partitions is fixed when creating a graph space. The service capability of a single partition is determined by a single server. The operations depending on a single partition include fetching properties of a single vertex (`FETCH`), a breadth-first traversal from a single vertex (`GO`), etc.

- The horizontal scaling of the Graph Service:

  Each request from the client is handled by one and only one Graph server, with no other Graph servers participating in the processing of the request. Therefore, increasing the number of Graph machines can increase the overall QPS of the cluster, but cannot lower the latency of a single request.

- Metad does not support horizontal scaling.

Vertical scaling usually has higher hardware costs, but relatively simple operations. Nebula Graph 2.5.0 can also be scaled vertically.

9.4.3 Data transmission and optimization

- Read/write balance. Nebula Graph 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. Write large batches of data with SST files, and small batches of data with `INSERT` statements.
- Run `COMPACTION` and `BALANCE` jobs to optimize data format and storage distribution at the right time.
- Nebula Graph 2.5.0 N Does not support transactions and isolation in the relational database sense and is closer to NoSQL.

9.4.4 Query preheating and data preheating

Preheat on the application side:

- The Graph Service does not support pre-compiling queries and generating corresponding query plans, nor can it cache previous query results.
- The Storage Service does not support preheating data, and 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.

Last update: September 2, 2021

9.5 Execution plan

Nebula Graph 2.5.0 applies rule-based execution plans. Users cannot change execution plans, precompile queries and corresponding plan cache, or accelerate queries by specifying indexes.

To view the execution plan and executive summary, see EXPLAIN and PROFILE.

Last update: September 2, 2021

9.6 Processing super vertices

9.6.1 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), hot stocks in the stock market, four major banks in the banking system, hubs in transportation networks, high-traffic websites on the Internet, and on-fire products in E-commerce networks.

In Nebula Graph, a vertex and its properties form a key-value pair, with the ID of the vertex and other meta information as the key. Its outgoing edges are stored with it in the same partition in the form of different key-value pairs, and they are stored in 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

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 Nebula Graph 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 Nebula Graph.

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, A transfer scenario is `(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.

  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, for the loan network of `(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_TO]->(BankA), (BankA2)-[BELONGS_TO]->(BankA), (BankA3)-[BELONGS_TO]->(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: September 2, 2021

9.7 Add or delete tag

openCypher`SET label`REMOVE label` Nebula GraphTag TagTagTagTag`DELETE TAG` !!! caution Tag ## basketballplayerTag`shareholder` `DELETE TAG`Tag

```
//Tag
nebula> CREATE TAG shareholder();
nebula> CREATE TAG INDEX shareholder_tag on shareholder();
//Tag
nebula> INSERT VERTEX shareholder() VALUES "player100":();
nebula> INSERT VERTEX shareholder() VALUES "player101":();
//
nebula> MATCH (v:shareholder) RETURN v;
+------------------------------------------------------------------------+
| v                                                                      |
+------------------------------------------------------------------------+
| ("player100" :player{age: 42, name: "Tim Duncan"} :shareholder{})  |
+------------------------------------------------------------------------+
| ("player101" :player{age: 36, name: "Tony Parker"} :shareholder{}) |
+------------------------------------------------------------------------+
nebula> LOOKUP ON shareholder;
+-------------+
| VertexID    |
+-------------+
| "player100" |
+-------------+
| "player101" |
+-------------+
//player100
nebula> DELETE TAG shareholder FROM "player100";
nebula> LOOKUP ON shareholder;
+-------------+
| VertexID    |
+-------------+
| "player101" |
+-------------+
```

!!! note `REBUILD TAG INDEX ;` -->

Last update: September 16, 2021

10. Client

10.1 Clients overview

Nebula Graph supports multiple types of clients for users to connect to and manage the Nebula Graph database.

- Nebula Console: the native CLI client
- Nebula CPP: the Nebula Graph client for C++
- Nebula Java: the Nebula Graph client for Java
- Nebula Python: the Nebula Graph client for Python
- Nebula Go: the Nebula Graph client for Golang

> **Note**
>
> For now, only Nebula Java is thread-safe.

Last update: September 1, 2021

10.2 Nebula CPP

Nebula CPP is a C++ client for connecting to and managing the Nebula Graph database.

10.2.1 Prerequisites

- You have installed C++ and GCC 4.8 or later versions.
- You have prepared the correct resources.

10.2.2 Compatibility with Nebula Graph

Nebula Graph version	Nebula CPP version
2.5.0	2.5.0
2.0.1	2.0.0
2.0.0	2.0.0

10.2.3 Install Nebula CPP

1. Clone the Nebula CPP source code to the host.
   - (Recommended) To install a specific version of Nebula CPP, use the Git option `--branch` to specify the branch. For example, to install v2.5.0, run the following command:

     ```
     $ git clone --branch v2.5.0 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.

   > **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 ..
   ```

   > **Note**
   >
   > If G++ does not support C++ 11, add the option `-DDISABLE_CXX11_ABI=ON`.

5. Compile Nebula CPP.

   To speed up the compiling, use the `-j` option to set a concurrent number `N`. It should be \(\min(\text{CPU}\text{core number}, \frac{the_memory_size(GB)}{2})\).

```
$ make -j{N}
```

6. Install Nebula CPP.

```
$ sudo make install
```

7. Update the dynamic link library.

```
$ sudo ldconfig
```

10.2.4 Use Nebula 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 Nebula Graph dynamic libraries. The default path is `/usr/local/nebula/lib64`.
- `include_folder_path`: The storage of the Nebula Graph 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
```

Core of the example code

This sub-section shows the core of the example code. For all the code, see SessionExample.

```
nebula::init(&argc, &argv);
auto address = "192.168.xx.1:9669";
nebula::ConnectionPool pool;
pool.init({address}, nebula::Config{});
auto session = pool.getSession("root", "nebula");

auto result = session.execute("SHOW HOSTS");
std::cout << *result.data;

std::atomic_bool complete{false};
session.asyncExecute("SHOW HOSTS", [&complete](nebula::ExecutionResponse&& cbResult) {
    std::cout << *cbResult.data;
    complete.store(true);
});
session.release();
```

Last update: September 2, 2021

10.3 Nebula Java

Nebula Java is a Java client for connecting to and managing the Nebula Graph database.

10.3.1 Prerequisites

You have installed Java 8.0 or later versions.

10.3.2 Compatibility with Nebula Graph

Nebula Graph version	Nebula Java version
2.5.0	2.5.0
2.0.1	2.0.0
2.0.0	2.0.0
2.0.0-rc1	2.0.0-rc1

10.3.3 Download Nebula Java

- (Recommended) To install a specific version of Nebula Java, use the Git option `--branch` to specify the branch. For example, to install v2.5.0, run the following command:

```
$ git clone --branch v2.5.0 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
```

10.3.4 Use Nebula Java

> **Note**
>
> We recommend that each thread uses 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`.

> **Note**
>
> `2.0.0-SNAPSHOT` indicates the daily development version that may have unknown issues. We recommend that you replace `2.0.0-SNAPSHOT` with a released version number to use a table version.

```
<dependency>
  <groupId>com.vesoft</groupId>
  <artifactId>client</artifactId>
  <version>2.0.0-SNAPSHOT</version>
</dependency>
```

If you can't download the dependency for the development version, set the following content in `pom.xml`. Released versions have no such issue.

```
<repositories>
  <repository>
    <id>snapshots</id>
    <url>https://oss.sonatype.org/content/repositories/snapshots/</url>
  </repository>
</repositories>
```

If there is no Maven to manage the project, manually download the JAR file to install Nebula Java.

Core of the example code

This sub-section shows the core of the example code. For all the code, see GraphClientExample.

```
NebulaPool pool = new NebulaPool();
Session session = null;
try {
    NebulaPoolConfig nebulaPoolConfig = new NebulaPoolConfig();
    nebulaPoolConfig.setMaxConnSize(100);
    List<HostAddress> addresses = Arrays.asList(new HostAddress("192.168.xx.1", 9669),
            new HostAddress("192.168.xx.2", 9670));
    pool.init(addresses, nebulaPoolConfig);
    session = pool.getSession("root", "nebula", false);

    //create space
    String space = "test";
    String createSpace = "CREATE SPACE IF NOT EXISTS " + space + " (partition_num=15, replica_factor=1, vid_type=fixed_string(30)); ";
    ResultSet resp = session.execute(createSpace);


    //create schema
    String createSchema = "USE " + space + "; CREATE TAG IF NOT EXISTS person(name string, age int);"
            + "CREATE EDGE IF NOT EXISTS like(likeness double)";
    ResultSet resp = session.execute(createSchema);

    //insert vertex
    String insertVertexes = "INSERT VERTEX person(name, age) VALUES " + "'Bob':('Bob', 10), "
            + "'Lily':('Lily', 9), " + "'Tom':('Tom', 10), " + "'Jerry':('Jerry', 13), "
            + "'John':('John', 11);";
    ResultSet resp = session.execute(insertVertexes);

    // inert edge
    String insertEdges = "INSERT EDGE like(likeness) VALUES " + "'Bob'->'Lily':(80.0), "
            + "'Bob'->'Tom':(70.0), " + "'Lily'->'Jerry':(84.0), " + "'Tom'->'Jerry':(68.3), "
            + "'Bob'->'John':(97.2);";
    ResultSet resp = session.execute(insertEdges);

    // query
    String query = "GO FROM \"Bob\" OVER like " + "YIELD $$.person.name, $$.person.age, like.likeness";
    ResultSet resp = session.execute(query);
    printResult(resp);
}finally {
  if (session != null) {
      session.release();
  }
  pool.close();
}
```

Last update: September 3, 2021

10.4 Nebula Python

Nebula Python is a Python client for connecting to and managing the Nebula Graph database.

10.4.1 Prerequisites

You have installed Python 3.5 or later versions.

10.4.2 Compatibility with Nebula Graph

Nebula Graph version	Nebula Python version
2.5.0	2.5.0
2.0.1	2.0.0
2.0.0	2.0.0
2.0.0-rc1	2.0.0rc1

10.4.3 Install Nebula Python

Install Nebula Python with pip

```
$ pip install nebula2-python==<version>
```

Install Nebula Python from the source code

1. Clone the Nebula Python source code to the host.
   - (Recommended) To install a specific version of Nebula Python, use the Git option `--branch` to specify the branch. For example, to install v2.5.0, run the following command:

     ```
     $ git clone --branch v2.5.0 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 dependencies.

   ```
   $ pip install -r requirements.txt
   ```

   > **Note**
   >
   > To run unit tests in the development mode, install dependencies of `requirements-dev.txt`.

4. Run the following command to install Nebula Python.

   ```
   $ sudo python3 setup.py install
   ```

10.4.4 Core of the example code

This section shows the core of the example code. For all the code, see Example.

Connect to the Graph Service

```
# Customize configurations.
config = Config()
config.max_connection_pool_size = 10
# Initialize the connection pool.
connection_pool = ConnectionPool()
# Returns true if the server is healthy, false otherwise.
ok = connection_pool.init([('192.168.xx.1', 9669)], config)

# Method 1: Manually specify when to release the session.
# Get the session from the connection pool.
session = connection_pool.get_session('root', 'nebula')

# Select a graph space.
session.execute('USE basketballplayer')

# Run the SHOW TAGS statement.
result = session.execute('SHOW TAGS')
print(result)

# Release the session.
session.release()

# Method 2: Use session_context to automatically release the session.
with connection_pool.session_context('root', 'nebula') as session:
    session.execute('USE basketballplayer;')
    result = session.execute('SHOW TAGS;')
    print(result)

# Close the connection pool.
connection_pool.close()
```

Connect to the Storage Server

```
# Set the IP addresses of all Meta servers.
meta_cache = MetaCache([('192.168.xx.1', 9559),
                        ('192.168.xx.2', 9559),
                        ('192.168.xx.3', 9559)],
                       50000)
graph_storage_client = GraphStorageClient(meta_cache)

resp = graph_storage_client.scan_vertex(
        space_name='ScanSpace',
        tag_name='person')
while resp.has_next():
    result = resp.next()
    for vertex_data in result:
        print(vertex_data)

resp = graph_storage_client.scan_edge(
    space_name='ScanSpace',
    edge_name='friend')
while resp.has_next():
    result = resp.next()
    for edge_data in result:
        print(edge_data)
```

Last update: September 3, 2021

10.5 Nebula Go

Nebula Go is a Golang client for connecting to and managing the Nebula Graph database.

10.5.1 Prerequisites

You have installed Golang 1.13 or later versions.

10.5.2 Compatibility with Nebula Graph

Nebula Graph version	Nebula Go version
2.5.0	2.5.0
2.0.1	2.0.0-GA
2.0.0	2.0.0-GA

10.5.3 Download Nebula Go

- (Recommended) To install a specific version of Nebula Go, use the Git option `--branch` to specify the branch. For example, to install v2.5.0, run the following command:

```
$ git clone --branch v2.5.0 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
```

10.5.4 Install or update

Run the following command to install or update Nebula Go:

```
$ go get -u -v github.com/vesoft-inc/nebula-go@<tag>
```

`tag`: Specify the branch, such as `master` or `v2.5.0`.

10.5.5 Core of the example code

This section shows the core of the example code. For all the code, see graph_client_basic_example and graph_client_goroutines_example.

```
const (
    address = "192.168.xx.1"
    port     = 9669
    username = "root"
    password = "nebula"
)

func main() {
    hostAddress := nebula.HostAddress{Host: address, Port: port}
    hostList := []nebula.HostAddress{hostAddress}
    testPoolConfig := nebula.GetDefaultConf()
    pool, err := nebula.NewConnectionPool(hostList, testPoolConfig, log)
    defer pool.Close()
    session, err := pool.GetSession(username, password)
    defer session.Release()

    checkResultSet := func(prefix string, res *nebula.ResultSet) {
        if !res.IsSucceed() {
            log.Fatal(fmt.Sprintf("%s, ErrorCode: %v, ErrorMsg: %s", prefix, res.GetErrorCode(), res.GetErrorMsg()))
        }
    }
    {
        createSchema := "CREATE SPACE IF NOT EXISTS basic_example_space(vid_type=FIXED_STRING(20)); " +
            "USE basic_example_space;" +
            "CREATE TAG IF NOT EXISTS person(name string, age int);" +
```

```
            "CREATE EDGE IF NOT EXISTS like(likeness double)"
        resultSet, err := session.Execute(createSchema)
        checkResultSet(createSchema, resultSet)
    }
    fmt.Print("\n")
    log.Info("Nebula Go Client Basic Example Finished")
}
```

Last update: September 1, 2021

11. Nebula Graph Studio

11.1 Change Log

11.1.1 v3.0.0 (2021.08.13)

- Feature Enhancements:
  - Compatible with Nebula Graph v2.5.0.
  - Supported adding `COMMENT` in Space, Tag, Edge Type, Index while configuration Schema.

Last update: September 2, 2021

11.2 About Nebula Graph Studio

11.2.1 What is Nebula Graph Studio

Nebula Graph Studio (Studio in short) is a browser-based visualization tool to manage Nebula Graph. It provides you with a graphical user interface to manipulate graph schemas, import data, explore graph data, and run nGQL statements to retrieve data. With Studio, you can quickly become a graph exploration expert from scratch. Users can view the latest source code in the Nebula Graph GitHub repository, see nebula-studio for details.

Released versions

For now, Studio has three release versions:

- Docker-based. You can deploy Studio with Docker and connect it to Nebula Graph. For more information, see Docker-based Studio.
- RPM-based. You can deploy Studio with RPM and connect it to Nebula Graph. For more information, see RPM-based Studio.
- tar-based. You can deploy Studio with tar and connect it to Nebula Graph. For more information, see tar-based Studio.

The functions of the three released versions are the same and may be restricted when using Studio. For more information, see Limitations.

Features

Studio provides these features:

- Graphical user interface (GUI) makes Nebula Graph management more user-friendly:
  - On the **Schema** page, you can manage schemas with a graphical user interface. It helps you quickly get started with Nebula Graph.
  - On the **Console** page, you can run nGQL statements and read the results in a human-friendly way.
  - 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 **Explore** page, you can explore the graph data. It helps you dig the relationships among data and improves the efficiency of data analysis.

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 Nebula Graph and then use Studio to explore or analyze data in a visualized way.
- You have deployed Nebula Graph and imported a dataset. You want to use a GUI to run nGQL statements or explore and analyze graph data in a visualized way.
- You are a beginner of nGQL (Nebula Graph 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 Nebula Graph by default. Users can log into Studio with the `root` account and any password.

When Nebula Graph enables authentication, users can only sign into Studio with the specified account. For more information, see Authentication.

Last update: September 3, 2021

11.2.2 Explanations of terms

This topic provides explanations of terms you may need to know when using Studio.

- Nebula Graph Studio: Referred to as Studio in this manual. Studio is a browser-based visualization tool to manage Nebula Graph. It provides you with a graphical user interface to manipulate graph schemas, import data, explore graph data, and run nGQL statements to retrieve data.
- Nebula Graph: Nebula Graph 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. For details, refer to Nebula Graph User Manual.

Last update: August 27, 2021

11.2.3 Limitations

This topic introduces the limitations of Studio.

Nebula Graph versions

> **Note**
>
> The Studio version is released independently of the Nebula Graph core. The correspondence between the versions of Studio and the Nebula Graph core, as shown in the table below.

Nebula Graph version	Studio version
1.x	1.x
2.0 & 2.0.1	2.x
2.5.0	3.0.0

Architecture

For now, Docker-based and RPM-based 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, you can export the queried results in the CSV format on the **Console** page and export data in the CSV format on the **Explore** page. No other backup methods are available.

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

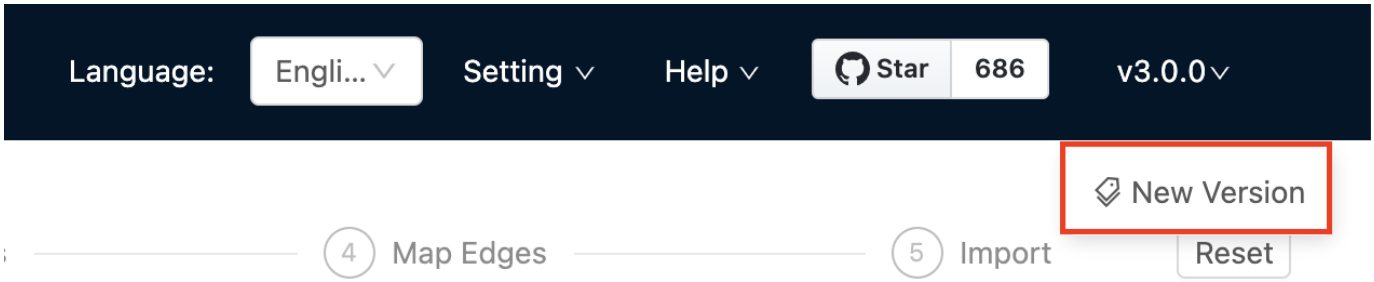
We recommend that you use Chrome to get access to Studio.

Last update: September 3, 2021

11.2.4 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: August 27, 2021

11.2.5 Shortcuts

This topic lists the shortcuts supported in Studio.

Description	Operation
Run nGQL statements in **Console**	Shift + Enter
Select multiple vertices in **Schema**	Shift + Left-click
Zoom out graph in **Schema**	Shift + '-'
Zoom in graph in **Schema**	Shift + '+'
Show graph in **Schema**	Shift + 'l'
Rollback in **Schema**	Shift + 'z'
Delete map in **Schema**	Selected + Shift + 'del'
Expand a vertex in **Schema**	Double Left-click or Shift + Enter

Last update: September 3, 2021

11.3 Deploy and connect

11.3.1 Deploy Studio

This topic describes how to deploy Studio locally by Docker, RPM, and tar package.

> **Note**
>
> You can also try some functions online in Studio.

Docker-based Studio

PREREQUISITES

Before you deploy Studio, you must do a check of these:

- The Nebula Graph services are deployed and started. For more information, see Nebula Graph 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
8080	Nebula-http-gateway, Client's HTTP service
5699	Nebula importer file import tool, provide data import service

PROCEDURE

To deploy and start Docker-based Studio, run the following commands. Here we use Nebula Graph v2.5.0 for demonstration:

1. Download the configuration files for the deployment.

Installation package	Nebula Graph version
nebula-graph-studio-v3.tar.gz	v2.5.0

2. Create the `nebula-graph-studio-v3` directory and decompress the installation package to the directory.

```
mkdir nebula-graph-studio-v3 && tar -zxvf nebula-graph-studio-v3.gz -C nebula-graph-studio-v3
```

3. Change to the `nebula-graph-studio-v3` directory.

```
cd nebula-graph-studio-v3
```

4. Pull the Docker image of Studio.

```
docker-compose pull
```

5. Build and start Docker-based Studio. In this command, `-d` is to run the containers in the background.

```
docker-compose up -d
```

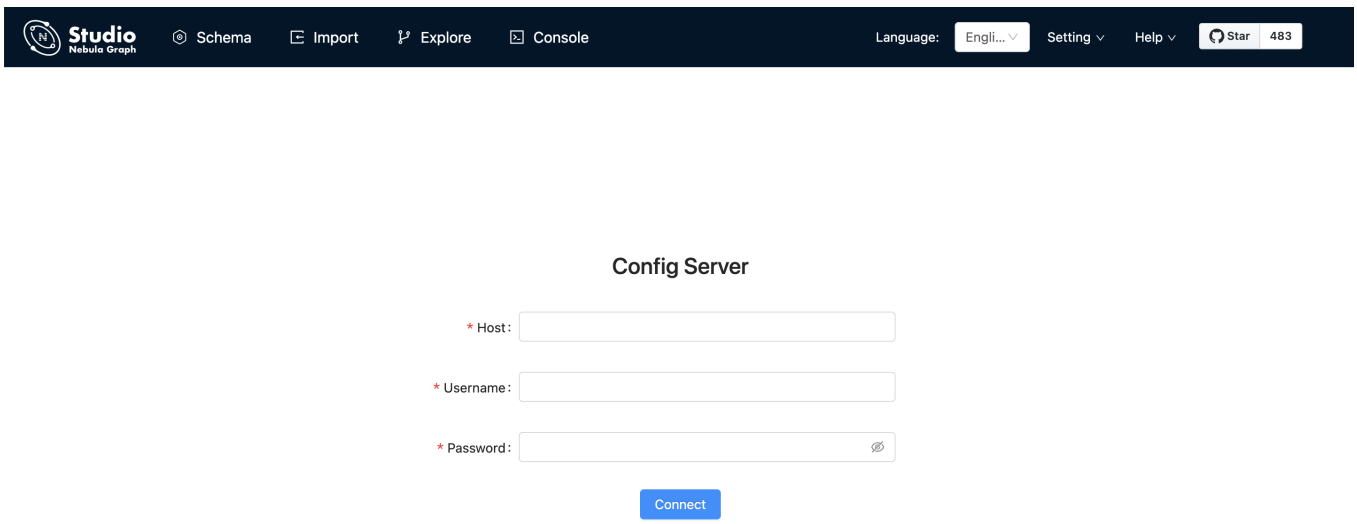
If these lines are returned, Docker-based Studio v3.x is deployed and started.

```
Creating docker_importer_1 ... done
Creating docker_client_1   ... done
Creating docker_web_1      ... done
Creating docker_nginx_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 `http://localhost:7001` to get access to Studio.

If you can see the **Config Server** page on the browser, Docker-based Studio is started successfully.



RPM-based Studio

PREREQUISITES

Before you deploy Docker-based Studio, you must confirm that:

- The Nebula Graph services are deployed and started. For more information, see Nebula Graph Database Manual.
- If your Linux distribution is CentOS, install `lsof` and Node.js of versions above v10.16.0+.

> **Note**
>
> `node` and `npm` should be installed in `/usr/bin/` directory. Avoid the situation that the node command cannot be found during RPM installation. For example, the default directory of nodejs12 is in `/opt/rh/rh-nodejs12`, you can use following commands to build soft link:
>
> ```
> $ sudo ln -s /opt/rh/rh-nodejs12/root/usr/bin/node /usr/bin/node
> $ sudo ln -s /opt/rh/rh-nodejs12/root/usr/bin/npm /usr/bin/npm
> ```

- Before the installation starts, the following ports are not occupied.

Port	Description
7001	Web service provided by Studio.
8080	HTTP service provided by Nebula HTTP Gateway.
5699	Data import service provided by Nebula Importer.

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	Nebula version
nebula-graph-studio-3.0.0-1.x86_64.rpm	nebula-graph-studio-3.0.0-1.x86_64.rpm.sha256	v2.5.0

2. Use `sudo rpm -i <rpm>` to install RPM package.

   For example, install Studio 3.0.0, use the following command:

   ```
   sudo rpm -i nebula-graph-studio-3.0.0-1.x86_64.rpm
   ```

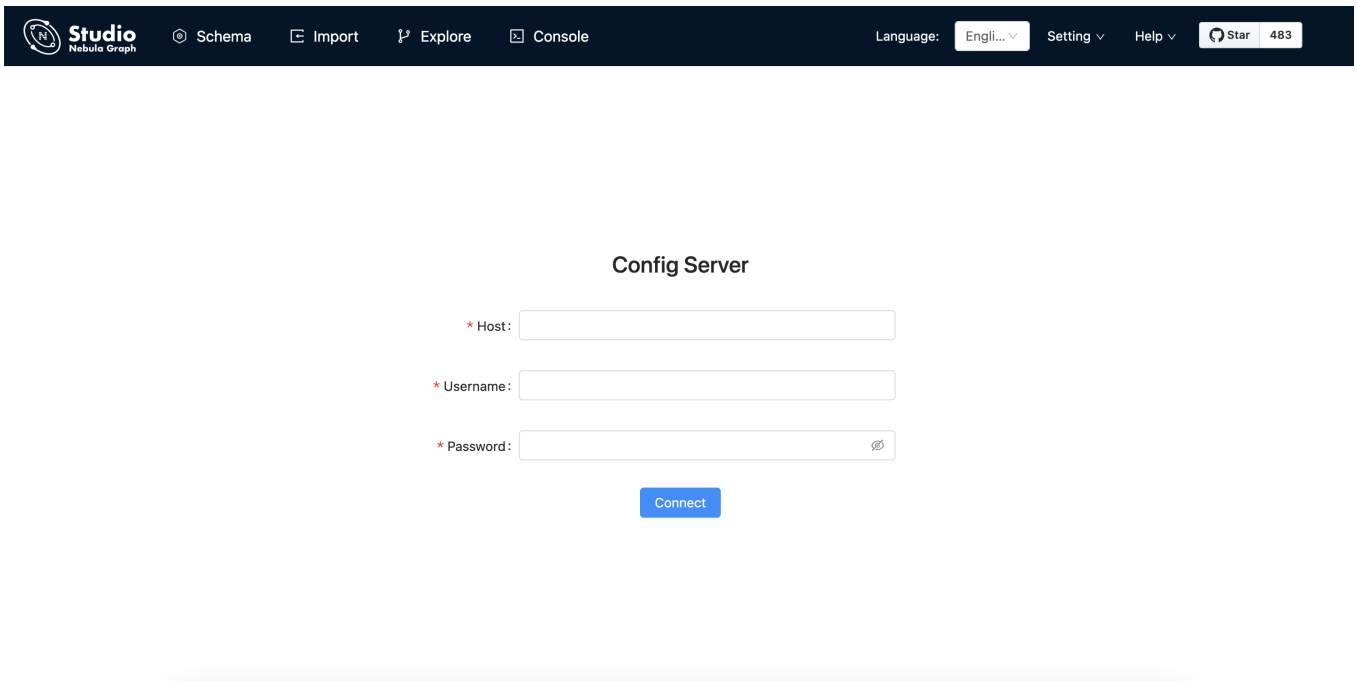
   When the screen returns the following message, it means that the PRM-based Studio has been successfully started.

   ```
   egg started on http://0.0.0.0:7001
   nohup: Add the output to "nohup.out"
   --- START OF NEBULA IMPORTER ---
   [INFO] httpserver.go:80: Starting http server on 5699
   ```

3. 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 `http://localhost:7001` to get access to Studio.

   If you can see the **Config Server** page on the browser, Docker-based Studio is started successfully.



UNINSTALL

Users can uninstall Studio using the following command:

```
sudo rpm -e nebula-graph-studio-3.0.0-1.x86_64
```

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/start.sh
  ```

- Stop the service manually

  ```
  bash /usr/local/nebula-graph-studio/scripts/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 use the following command to change Studio service port and restart the service.

```
//Open the configuration file
$ vi config/config.default.js

//Change the port number
...
    config.cluster = {
        listen: {
            port: 7001, // Modify this port number and change it to any one currently available
            hostname: '0.0.0.0',
        },
    };
...

//Restart npm
$ npm run start
```

tar-based Studio

PREREQUISITES

Before you deploy Docker-based Studio , you must do a check of these:

- The Nebula Graph services are deployed and started. For more information, see Nebula Graph Database Manual.
- The Linux distribution is CentOS, installed `lsof` and Node.js of version above v10.16.0+.

> **Note**
>
> `node` and `npm` should be installed in `/usr/bin/` directory. Avoid the situation that the node command cannot be found during RPM installation. For example, the default directory of nodejs12 is in `/opt/rh/rh-nodejs12`, you can use following commands to build soft link:

```
$ sudo ln -s /opt/rh/rh-nodejs12/root/usr/bin/node /usr/bin/node
$ sudo ln -s /opt/rh/rh-nodejs12/root/usr/bin/npm /usr/bin/npm
```

- Before the installation starts, the following ports are not occupied.

Port	Description
7001	Web service provided by Studio
8080	Nebula-http-gateway, Client's HTTP service
5699	Nebula importer, provide data import service

INSTALL

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-3.0.0-1.x86_64.tar.gz	3.0.0

2. Use `tar -xvf` to decompress the tar package.

```
tar -xvf nebula-graph-studio-3.0.0-1.x86_64.tar.gz
```

PROCEDURE

> **Note**
>
> The root directory `nebula-graph-studio` has three installation packages: nebula-graph-studio, nebula-importer and nebula-http-gateway. You need to deploy and start the services separately on the same machine to complete the deployment of Studio.

1. Deploy and start nebula-importer.

```
$ cd nebula-importer
$ ./nebula-importer --port 5699 --callback "http://0.0.0.0:7001/api/import/finish" &
```

2. Deploy and start nebula-http-gateway.

```
$ cd nebula-http-gateway
$ nohup ./nebula-httpd &
```

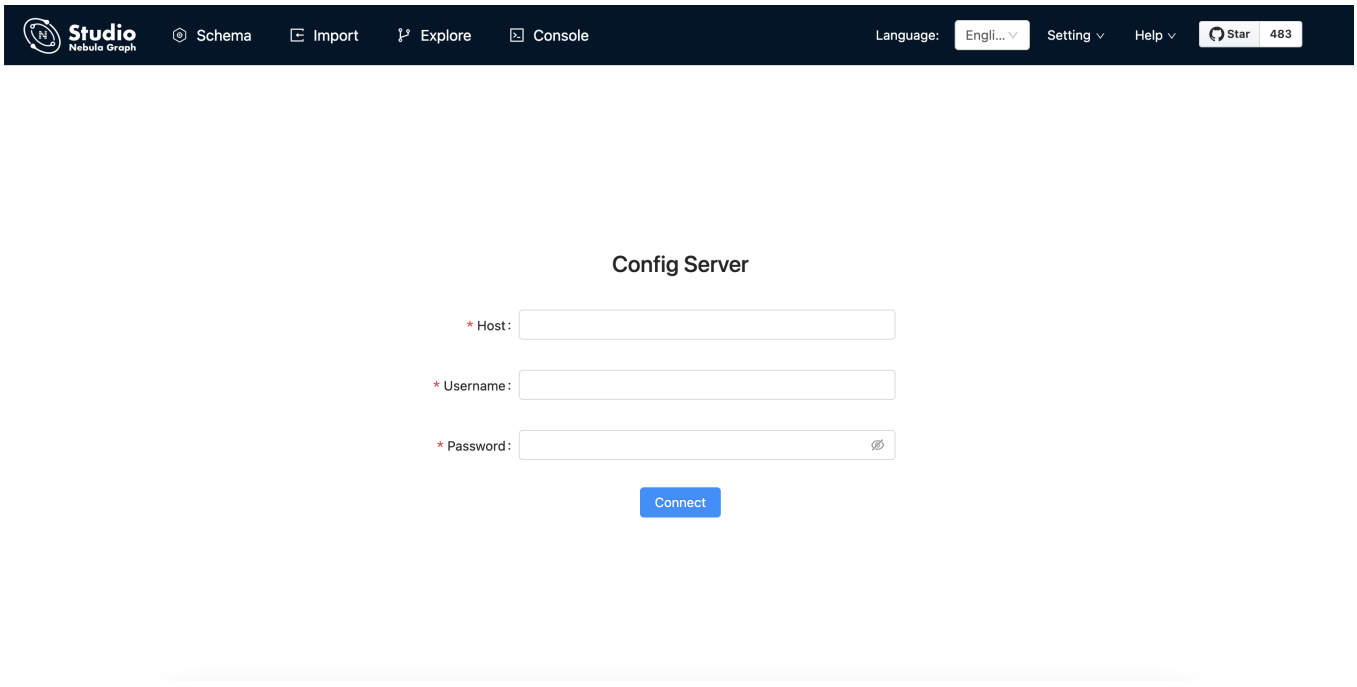
3. Deploy and start nebula-graph-studio.

```
$ cd nebula-graph-studio
$ npm run start
```

4. When tar-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 `http://localhost:7001` to get access to Studio.

If you can see the **Config Server** page on the browser, Docker-based Studio is started successfully.



STOP SERVICE

You can use `kill pid` to stop the service:

```
$ kill $(lsof -t -i :5699) # stop nebula-importer
$ kill $(lsof -t -i :8080) # stop nebula-http-gateway
$ cd nebula-graph-studio
$ npm run stop # stop nebula-graph-studio
```

Next to do

On the **Config Server** page, connect Docker-based Studio to Nebula Graph. For more information, see Connect to Nebula Graph.

Last update: September 3, 2021

11.3.2 Connect to Nebula Graph

After successfully launching Studio, you need to configure to connect to Nebula Graph. This topic describes how Studio connects to the Nebula Graph database.

Prerequisites

Before connecting to the Nebula Graph database, you need to confirm the following information:

- The Nebula Graph 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 Nebula Graph. The default port is `9669`.

> **Note**
>
> Run `ifconfig` or `ipconfig` on the machine to get the IP address.

- You have a Nebula Graph account and its password.

> **Note**
>
> If authentication is enabled in Nebula Graph and different role-based accounts are created, you must use the assigned account to connect to Nebula Graph. If authentication is disabled, you can use the `root` and any password to connect to Nebula Graph. For more information, see Nebula Graph Database Manual.

Procedure

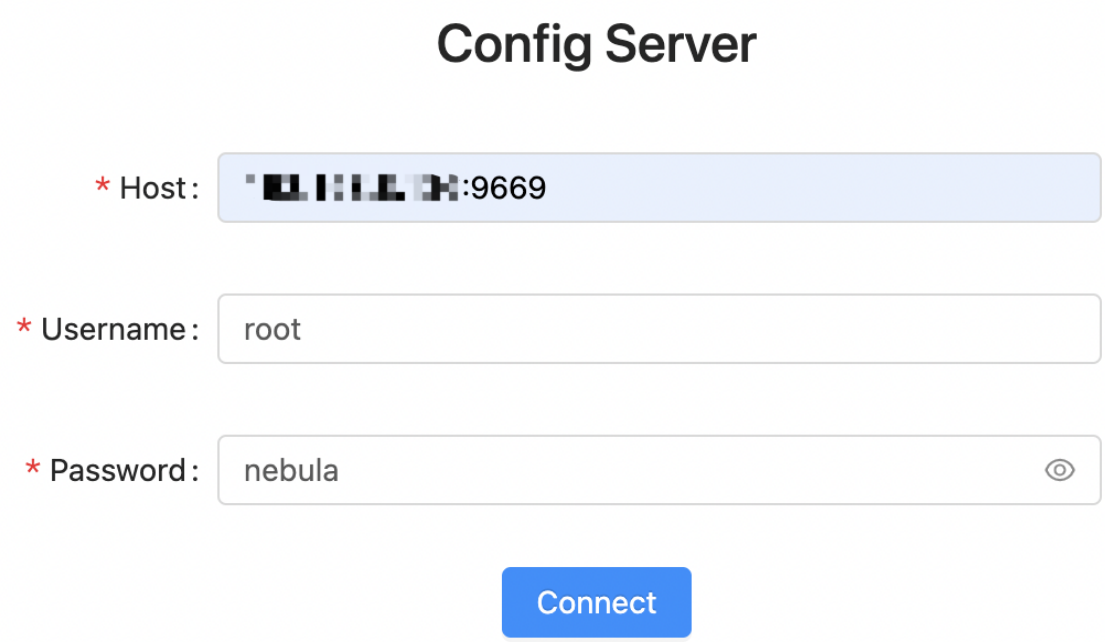
To connect Studio to Nebula Graph, follow these steps:

1. On the **Config Server** page of Studio, configure these fields:

   - **Host**: Enter the IP address and the port of the Graph service of Nebula Graph. The valid format is `IP:port`. The default port is `9669`.

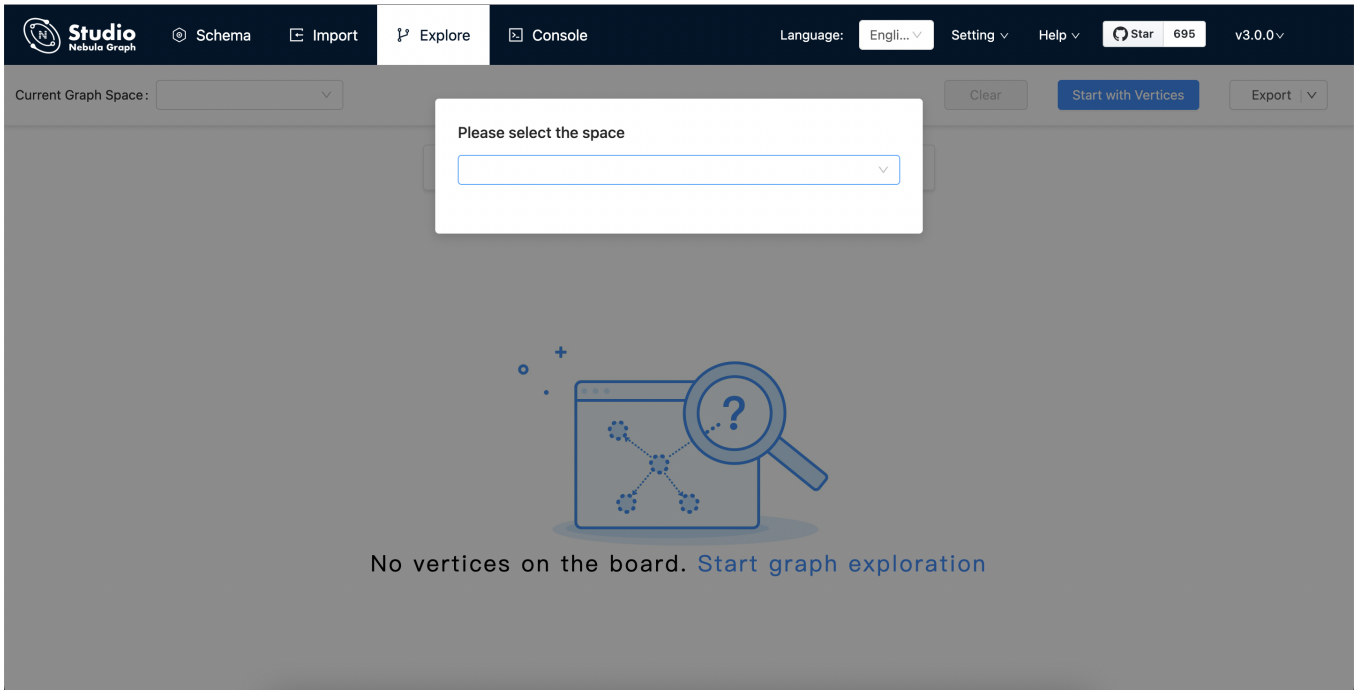
     > **Note**
     >
     > When Nebula Graph and Studio are deployed on the same machine, you must enter the IP address of the machine, but not `127.0.0.1` or `localhost`, in the **Host** field.

   - **Username** and **Password**: Fill in the log in account according to the authentication settings of Nebula Graph.
     - 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.



2. After the configuration, click the **Connect** button.

   If you can see the **Explore** page, Studio is successfully connected to Nebula Graph.



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 Nebula Graph again.

Next to do

When Studio is successfully connected to Nebula Graph, you can do these operations:

- If your account has GOD or ADMIN privilege, you can create a schema on the **Console** page or on the **Schema** page.
- If your account has GOD, ADMIN, DBA, or USER privilege, you can batch import data on the **Import** page or insert data with nGQL statements on the **Console** page.
- If your account has GOD, ADMIN, DBA, USER, or GUEST privilege, you can retrieve data with nGQL statements on the **Console** page or explore and analyze data on the **Explore** page.

Last update: September 2, 2021

11.3.3 Clear connection

Clear connection

If you want to reset Nebula Graph, you can clear the connection and reconfigure the database.

When the Studio is still connected to a Nebula Graph database, you can choose **setting > clear connect** at the toolbar. If the **Config Server** page is displayed on the browser, it means that Studio has successfully disconnected from the Nebula Graph database.

Last update: September 3, 2021

11.4 Quick start

11.4.1 Design a schema

To manipulate graph data in Nebula Graph with Studio, you must have a graph schema. This article introduces how to design a graph schema for Nebula Graph.

A graph schema for Nebula Graph must have these essential elements:

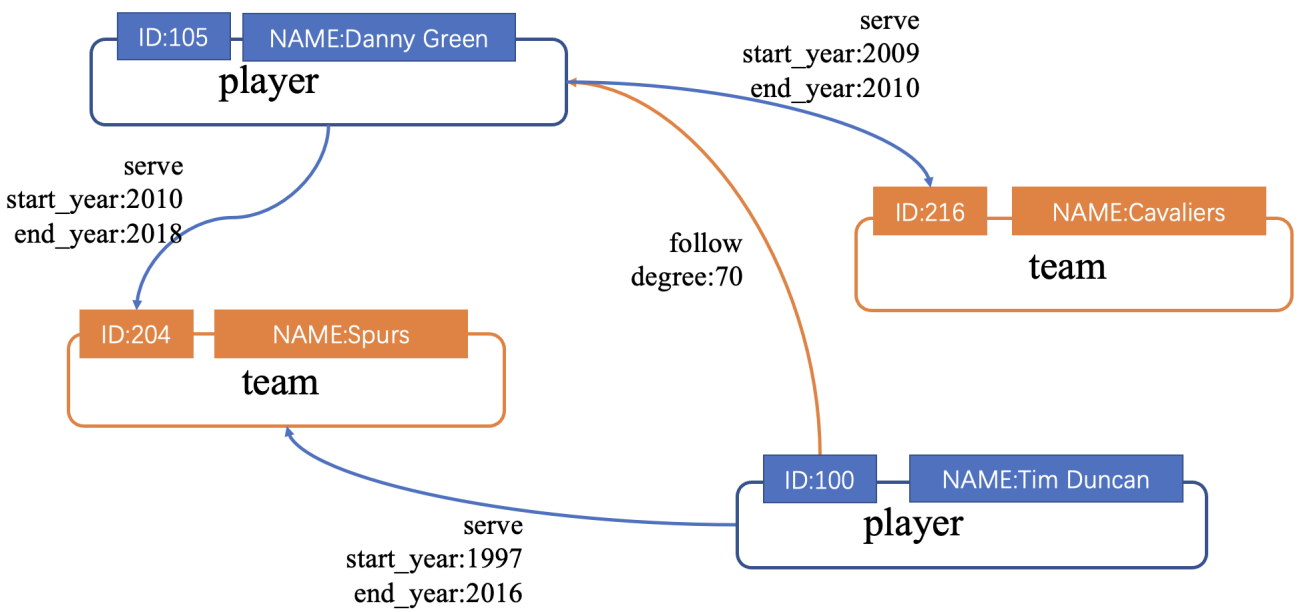
- Tags (namely vertex types) and their properties.
- Edge types and their properties.

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**	- `name` (`string`) - `age` (`int`)	Represents the player.
Tag	**team**	- `name` (`string`)	Represents the team.
Edge type	**serve**	- `start_year` (`int`) - `end_year` (`int`)	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 (**serve**/**follow**) between a **player** and a **team**.



Last update: September 3, 2021

11.4.2 Create a schema

To batch import data into Nebula Graph, you must have a graph schema. You can create a schema on the **Console** page or on the **Schema** page of Studio.

> **Note**
>
> You can use nebula-console to create a schema. For more information, see Nebula Graph Manual and Get started with Nebula Graph.

Prerequisites

To create a graph schema on Studio, you must do a check of these:

- Studio is connected to Nebula Graph.
- Your account has the privilege of GOD, ADMIN, or DBA.
- The schema is designed.
- A graph space is created.

> **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

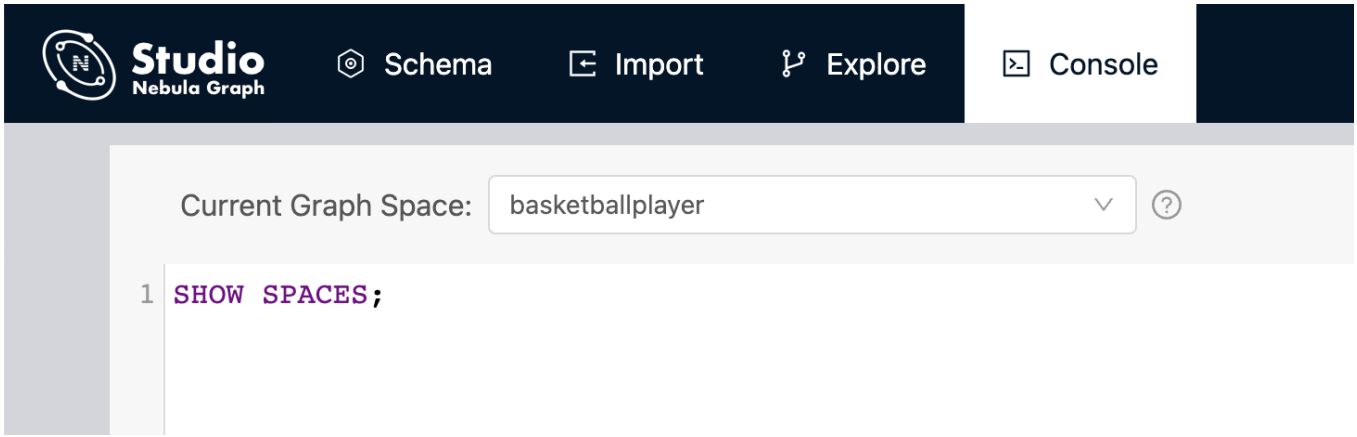
To create a schema on the **Schema** page, follow these steps:

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

To create a schema on the **Console** page, follow these steps:

1. In the toolbar, click the **Console** tab.
2. In the **Current Graph Space** field, choose a graph space name. In this example, **basketballplayer** is used.



3. In the input box, enter these statements one by one and click the button .

```
// To create a tag named "player", with two property
nebula> CREATE TAG player(name string, age int);

// To create a tag named "team", with one property
nebula> CREATE TAG team(name string);

// To create an edge type named "follow", with one properties
nebula> CREATE EDGE follow(degree int);

// To create an edge type named "serve", with two properties
nebula> CREATE EDGE serve(start_year int, end_year int);
```

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 serve;
```

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: September 3, 2021

11.4.3 Import data

After CSV files of data and a schema are created, you can use the **Import** page to batch import vertex and edge data into Nebula Graph for graph exploration and data analysis.

Prerequisites

To batch import data, do a check of these:

- Studio is connected to Nebula Graph.
- A schema is created.
- CSV files meet the demands of the Schema.
- Your account has privilege of GOD, ADMIN, DBA, or USER.

Procedure

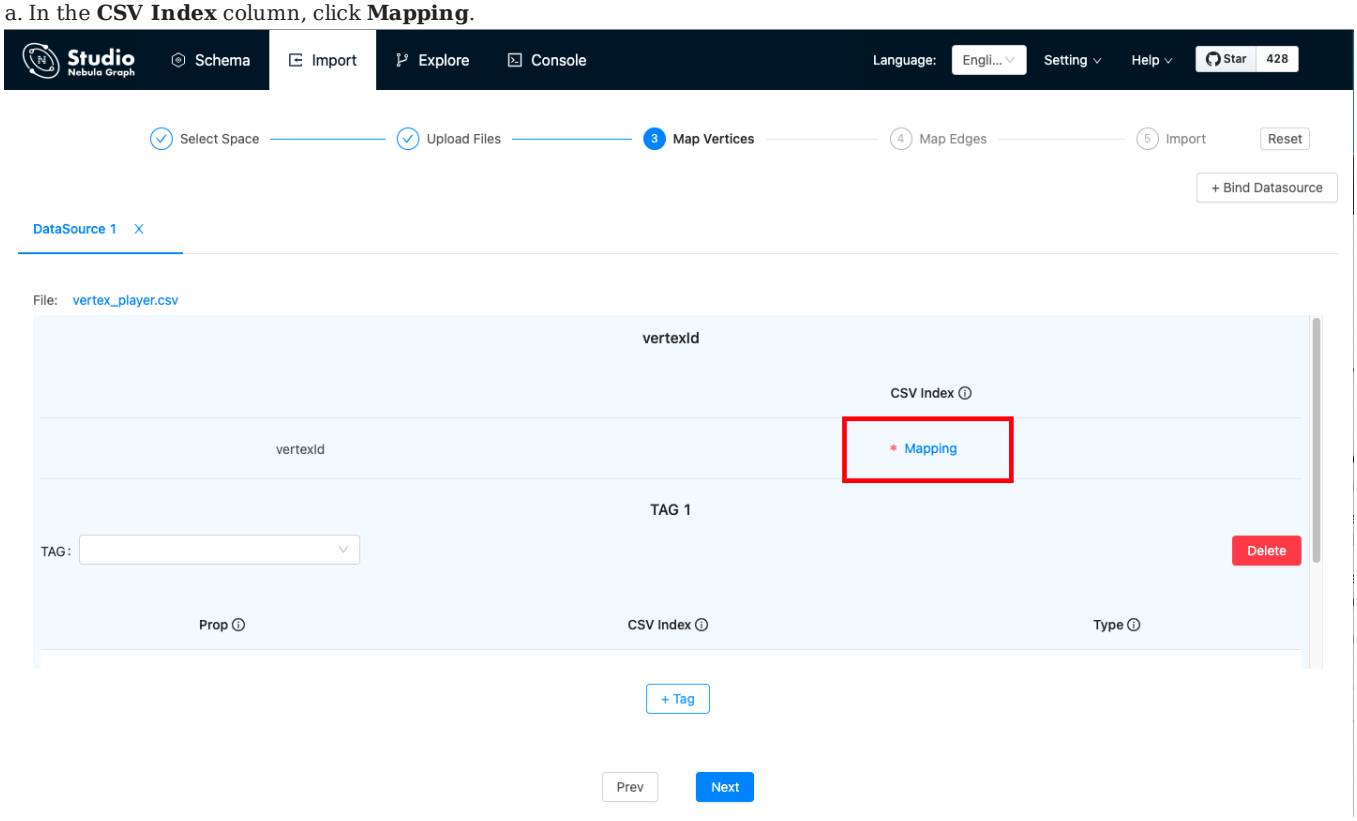
To batch import data, follow these steps:

1. In the toolbar, click the **Import** tab.
2. On the **Select Space** page, choose a graph space name. In this example, **basketballplayer** is used. And then click the **Next** button.
3. On the **Upload Files** page, click the **Upload Files** button and then choose CSV files. In this example, `edge_serve.csv`, `edge_follow.csv`, `vertex_player.csv`, and `vertex_team.csv` are chosen.

> **Note**
>
> You can choose multiple CSV files at the same time. The CSV file used in this article can be downloaded in the Design a schema.

4. On the **Select Files** page, do a check of the file size and click **Preview** or **Delete** in the **Operations** column to make sure that all source data is correct. And then click the **Next** button.
5. On the **Map Vertices** page, click the **+ Bind Datasource** button, and in the dialog box, choose a CSV file. In this example, `vertex_player.csv` or `vertex_team.csv` is chosen.
6. In the **DataSource X** tab, click the **+ Tag** button.
7. In the **vertexId** section, do these operations:

   a. In the **CSV Index** column, click **Mapping**.



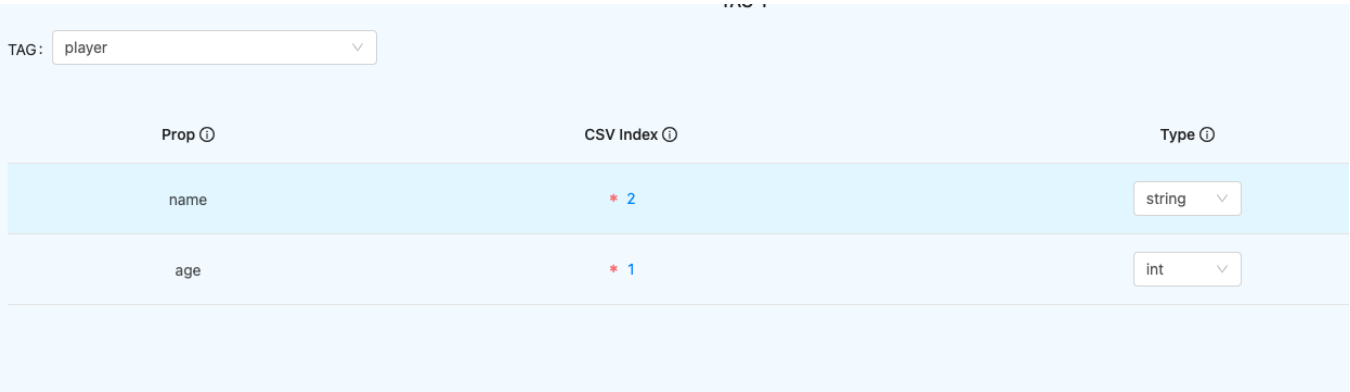
   b. In the dialog box, choose a column from the CSV file. In this example, the only one cloumn of `vertex_player.csv` is chosen to generate VIDs representing players and the `playerID` column of `vertex_player.csv` is chosen to generate VIDs representing players.

   !!! Note

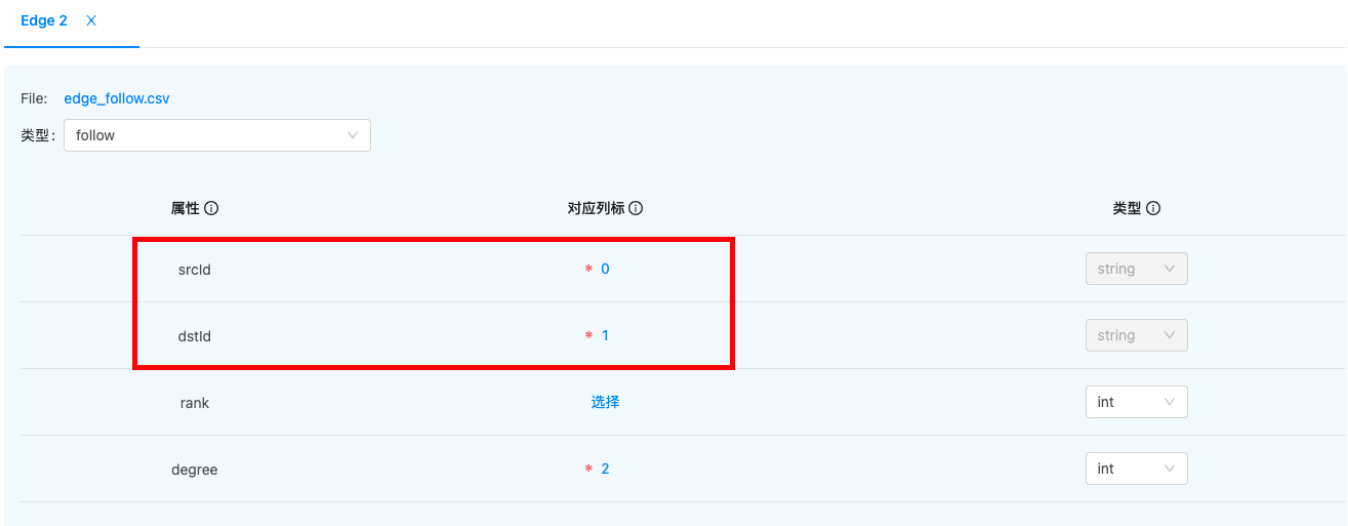
```
   In the same graph space, the VID is always unique and cannot be repeated. For VID information, see [VID](../../1.introduction/3.vid.md) "Click to enter the
Nebula Graph Manual".
```

8. In the **TAG 1** section, do these operations:
   a. In the **TAG** drop-down list, choose a tag name. In this example, **player** is used for the `vertex_player.csv` file, and **team** is used for the `vertex_team.csv` file.

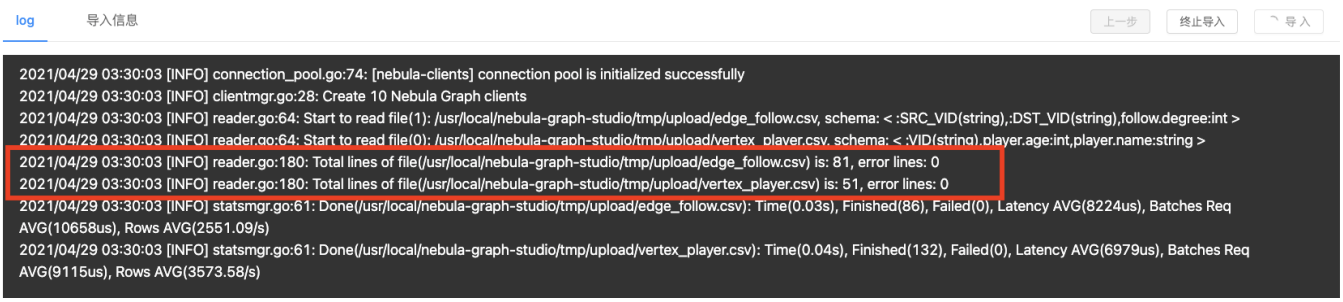
b. In the property list, click **Mapping** to choose a data column from the CSV file as the value of a property. In this example, for the **player** tag, choose **Column 1** for the `age` property and set its type to **int**. And choose **Column 2** for the `name` property and set its type to **string**.



9. (Optional) If necessary, repeat Step 5 through Step 8 for more tags.
10. When the configuration is done, click the **Next** button.
    When **Config validation was successful** prompts, data mapping for the vertices is successful.
11. On the **Map Edges** page, click the **+ Bind Datasource** button, and in the dialog box, choose a CSV file. In this example, the `edge_follow.csv` file is chosen.
12. In the **Type** drop-down list, choose an edge type name. In this example, **follow** is chosen.
13. In the property list, click **Mapping** to choose a column from the `edge_follow.csv` file as values of a property for the edges. **srcId** and **dstId** are the VIDs of the source vertex and destination vertex of an edge. In this example, **srcId** must be set to the VIDs of the player and **dstId** must be set to the VIDs of another player. **Rank** is optional.



14. When the configuration is done, click the **Next** button.
15. On the **Import** page, click the **Start Import** button. On the **log** window, you can see the import progress. The consumed time depends on the data volume. During data import, you can click the **Stop Import** button to stop data import. When the **log** window shows information as follows, the data import is done.



Next to do

When the data are imported to v2.5.0, you can query graph data.

Last update: September 3, 2021

11.4.4 Query graph data

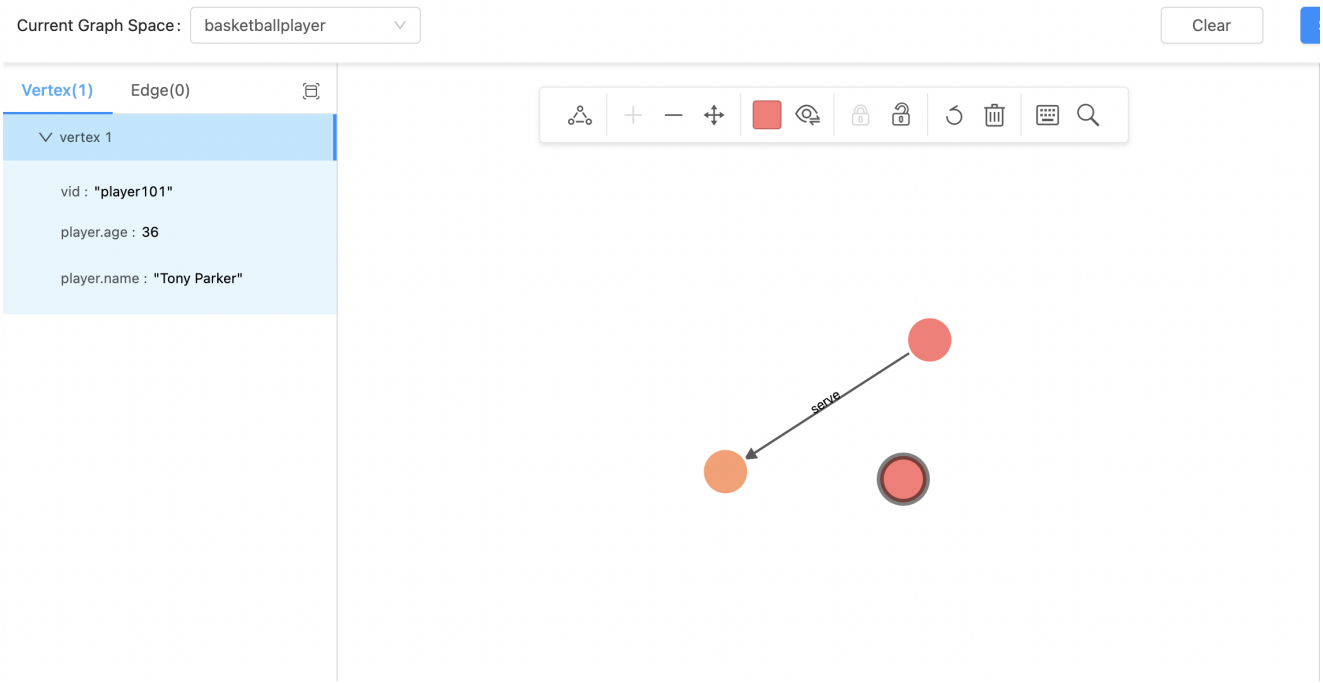
When data is imported, you can use the **Console** page or the **Explore** page to query graph data.

> **Note**
>
> Users can also perform the following query operations online through Studio.

For example, if you want to query the edge properties of the player named `player100` to the team named `team204`, you can perform these optional operations:

- On the **Console** tab: Run `FETCH PROP ON serve "player100" -> "team204";`. The result window shows all the property information of this vertex. When the result returns, click the **View Subgraph** button and then you can view the vertex information in a visualized way.

- On the **Explore** tab: Click the **Start with Vertices** button. In the dialog box, enter **player101** and then click the **Add** button. On the board, you can see the vertex. Move your mouse pointer on the vertex to see the vertex details, as shown in the preceding figure.



Last update: September 2, 2021

11.5 Operation guide

11.5.1 Use Schema

Operate graph spaces

When Studio is connected to Nebula Graph, 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 Nebula Graph.

STUDIO VERSION

Studio of v3.0.0 or later versions supports this function. For more information, see check updates.

PREREQUISITES

To operate a graph space on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to Nebula Graph.
- Your account has the authority of GOD. It means that:
  - If the authentication is enabled in Nebula Graph, you can use `root` and any password to sign in to Studio.
  - If the authentication is disabled in Nebula Graph, you must use `root` and its password to sign in to Studio.

CREATE A GRAPH SPACE

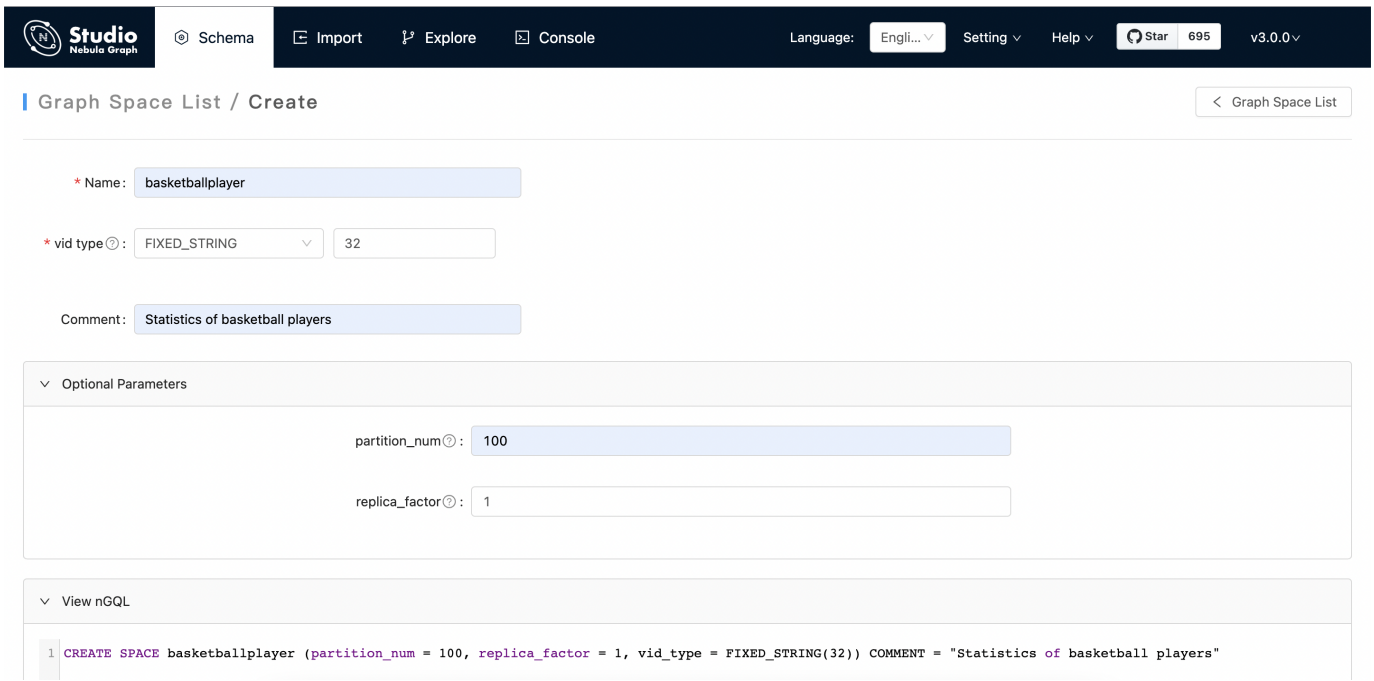
To create a graph space on the **Schema** page, follow these steps:

1. In the toolbar, click the **Schema** tab.
2. On the **Graph Space List** page, click the **+ Create** button.
3. On the **Create** page, do these settings:
   - **Name**: Specify a name to the new graph space. In this example, `basketballplayer` is used. The name must be distinct in the database. The name cannot be used keywords or reserved keywords as identifiers. For more information, see keywords.
   - **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**: The remarks of a certain property or the space itself. 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"
   ```

4. 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.

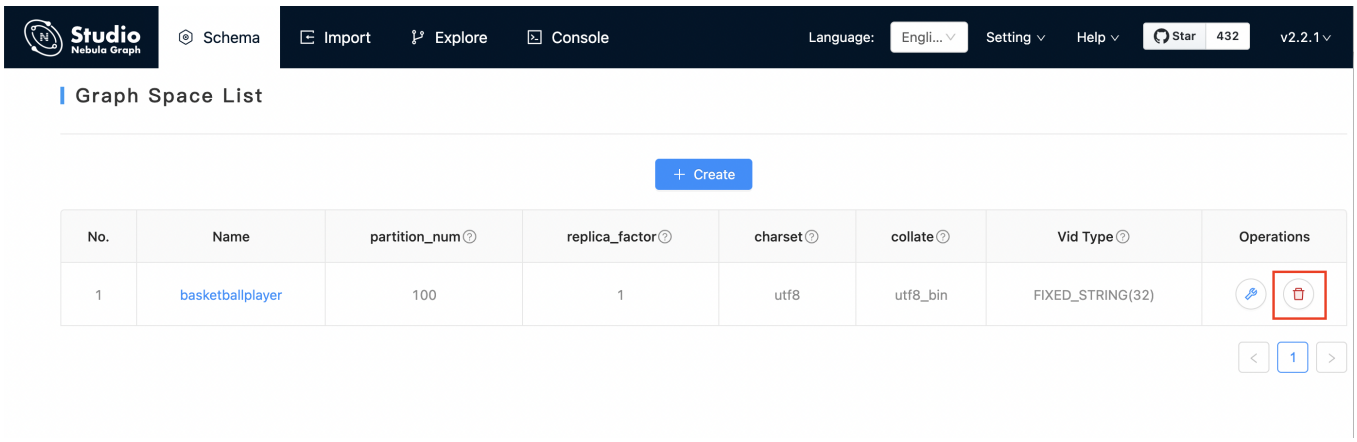


DELETE A GRAPH SPACE

> **Danger**
>
> Deleting the space will delete all the data in it, and the deleted data cannot be restored if it is not backed up.

To delete a graph space on the **Schema** page, follow these steps:

1. In the toolbar, click the **Schema** tab.
2. In the graph space list, find a graph space and then the button  in the **Operations** column.



3. On the dialog box, confirm the information and then click the **OK** button. When the graph space is deleted successfully, it is removed from the **graph space list**.

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: September 3, 2021

Operate tags

After a graph space is created in Nebula Graph, 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.

STUDIO VERSION

Studio of v3.0.0 or later versions supports this function. For more information, see check updates.

PREREQUISITES

To operate a tag on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to Nebula Graph.
- A graph space is created.
- Your account has the authority of GOD, ADMIN, or DBA.

CREATE A TAG

To create a tag on the **Schema** page, 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 the button  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:

   a. **Name**: Specify an appropriate name for the tag. In this example, `player` and `team` are specified.

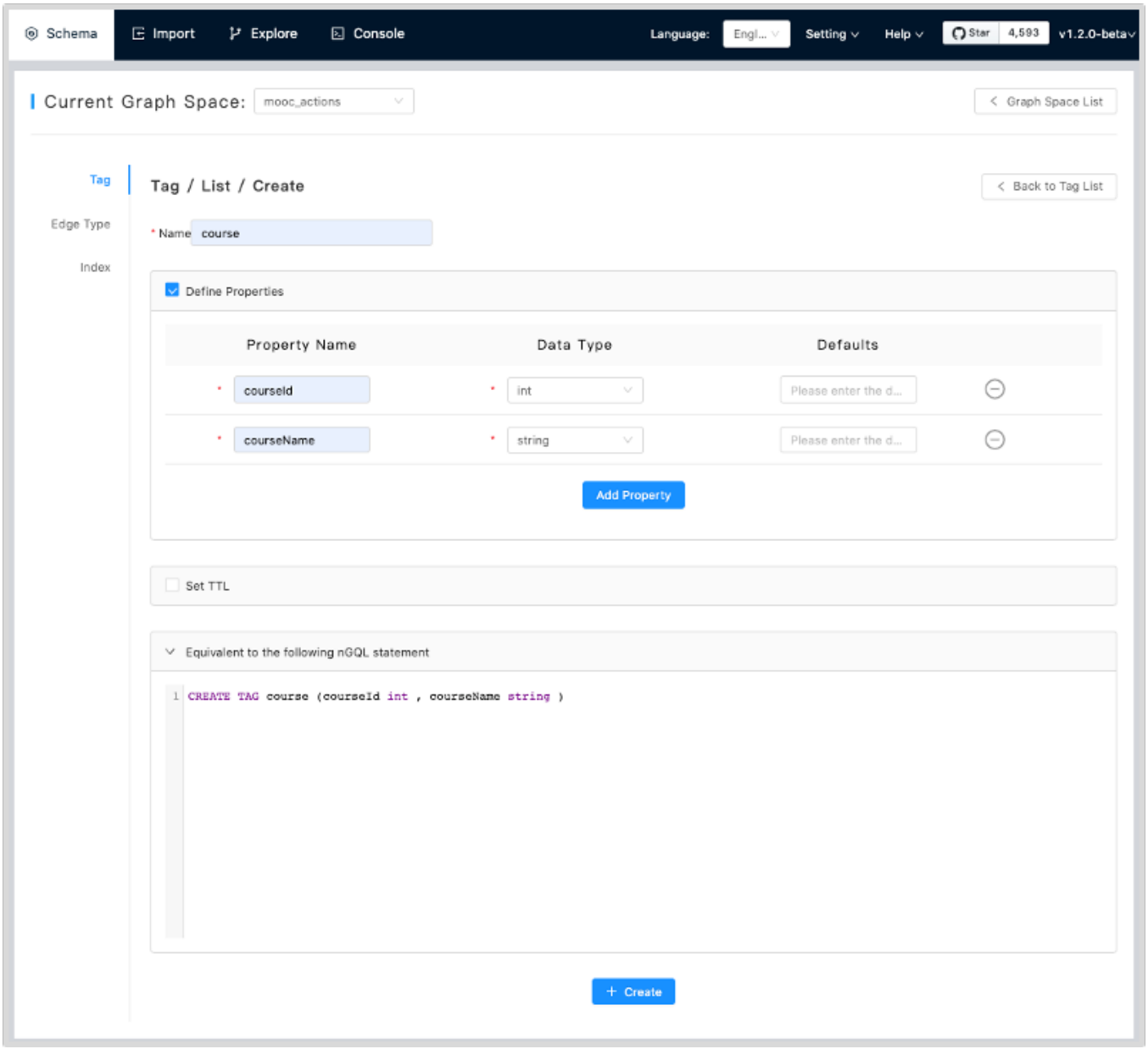
   b. (Optional) If necessary, in the upper left corner of the **Define Properties** panel, click the check box to expand the panel and do these settings:

   ```
   - To define a property: Enter a property name, a data type, and a default value.

   - To add multiple properties: Click the **Add Property** button and define more properties.

   - To cancel a defined property: Besides the **Defaults** column, click the button ![Icon of deletion](https://docs-cdn.nebula-graph.com.cn/nebula-studio-docs/st-ug-020.png "Cancel").
   ```

   c. (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`. 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.



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

To edit a tag on the **Schema** page, 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 the button  in the **Operations** column.
3. In **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 the button  in the **Operations** column.
5. On the **Edit** page, do these settings:
   - To edit a Comment: Click **Edit** under the **Name**.
   - 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 and then 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 the TTL configuration.
   - To edit the TTL configuration: On the **Set TTL** panel, click **Edit** and then change the configuration of `TTL_COL` and `TTL_DURATION`.
   - 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.
6. When the configuration is done, in the **Equivalent to the following nGQL statement** panel, you can see the equivalent `ALTER TAG` statement.

DELETE A TAG

> **Danger**
>
> Confirm the impact before deleting the tag. The deleted data cannot be restored if it is not backed up.

To delete a tag on the **Schema** page, follow these steps:

1. In the toolbar, click the **Schema** tab.
2. In **Graph Space List**, find a graph space, and then click its name or the button  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 the button  in the **Operations** column.
5. CLick **OK**.

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: September 3, 2021

Operate edge types

After a graph space is created in Nebula Graph, 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.

STUDIO VERSION

Studio of v3.0.0 or later versions supports this function. For more information, see check updates.

PREREQUISITES

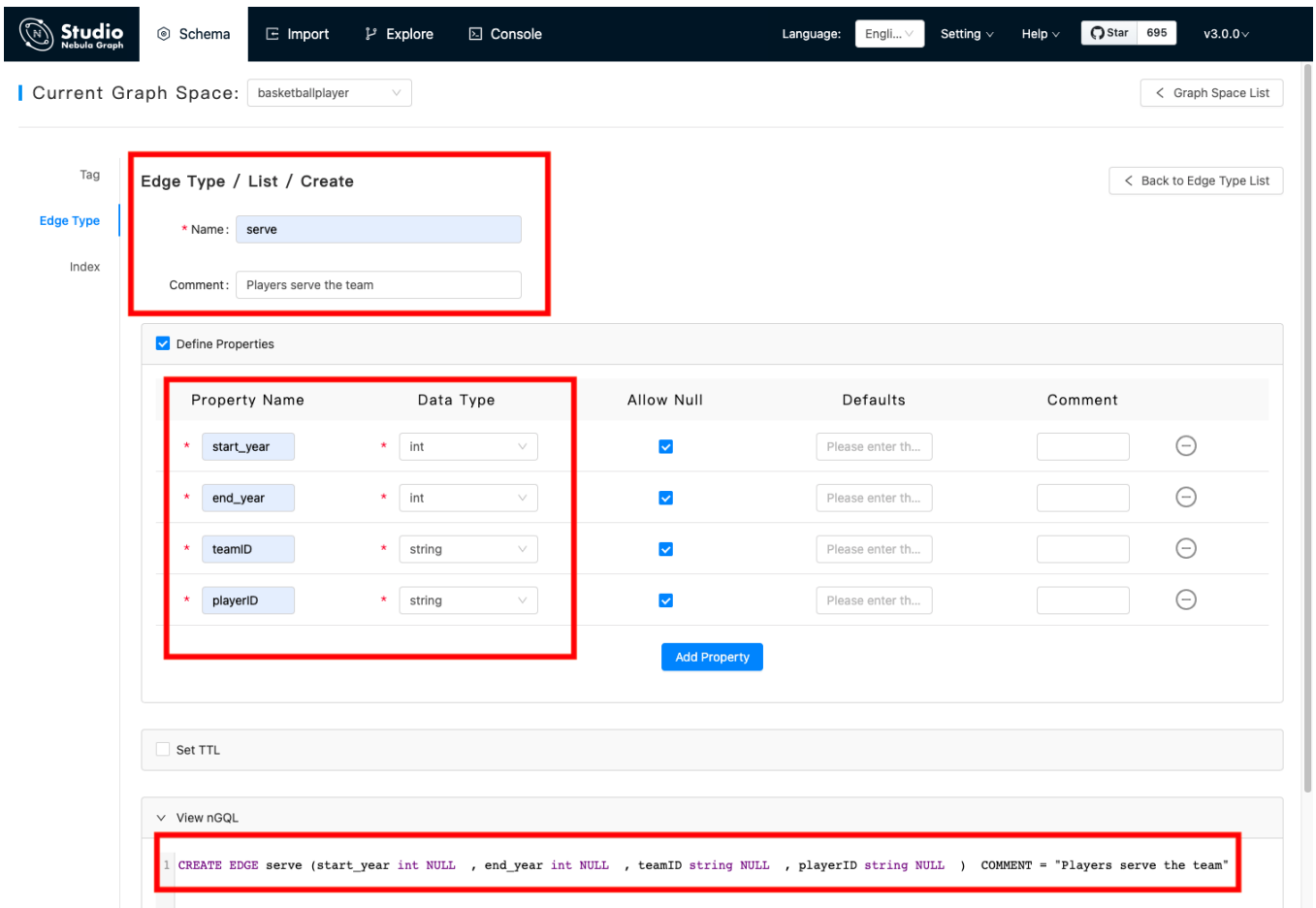
To operate an edge type on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to Nebula Graph.
- A graph space is created.
- Your account has the authority of GOD, ADMIN, or DBA.

CREATE AN EDGE TYPE

To create an edge type on the **Schema** page, 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 the button 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** page, do these settings:
   - **Name**: Specify an appropriate name for the edge type. In this example, `serve` is used.
   - (Optional) If necessary, under the name, click the **Comment** to input content.
   - (Optional) If necessary, in the upper left corner of the **Define Properties** panel, click the check box to expand the panel and do these settings:
     - To define a property: Enter a property name, a data type, and a default value.
     - To add multiple properties: Click the **Add Property** button and define more properties.
     - To delete a defined property: Besides the **Defaults** column, click the button .
   - (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`. 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.



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

To edit an edge type on the **Schema** page, 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 the button  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:
   - To edit a Comment: Click **Edit** under the Name.
   - 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 edit the TTL configuration: On the **Set TTL** panel, click **Edit** and then change the configuration of `TTL_COL` and `TTL_DURATION`.
   - 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.
6. When the configuration is done, in the **Equivalent to the following nGQL statement** panel, you can see the equivalent `ALTER EDGE` statement.

DELETE AN EDGE TYPE

> **Danger**
>
> Confirm the impact before deleting the Edge type. The deleted data cannot be restored if it is not backed up.

To delete an edge type on the **Schema** page, follow these steps:

1. In the toolbar, click the **Schema** tab.
2. In **Graph Space List**, find a graph space and then click its name or click the button  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 **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: September 3, 2021

Operate 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. You can create two index types: Tag Index and Edge type Index. 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 nGQL Manual.

STUDIO VERSION

Studio of v3.0.0 or later versions supports this function. For more information, see check updates.

PREREQUISITES

To operate an index on the **Schema** page of Studio, you must do a check of these:

- Studio is connected to Nebula Graph.
- A graph Space, Tags, and Edge Types are created.
- Your account has the authority of GOD, ADMIN, or DBA.

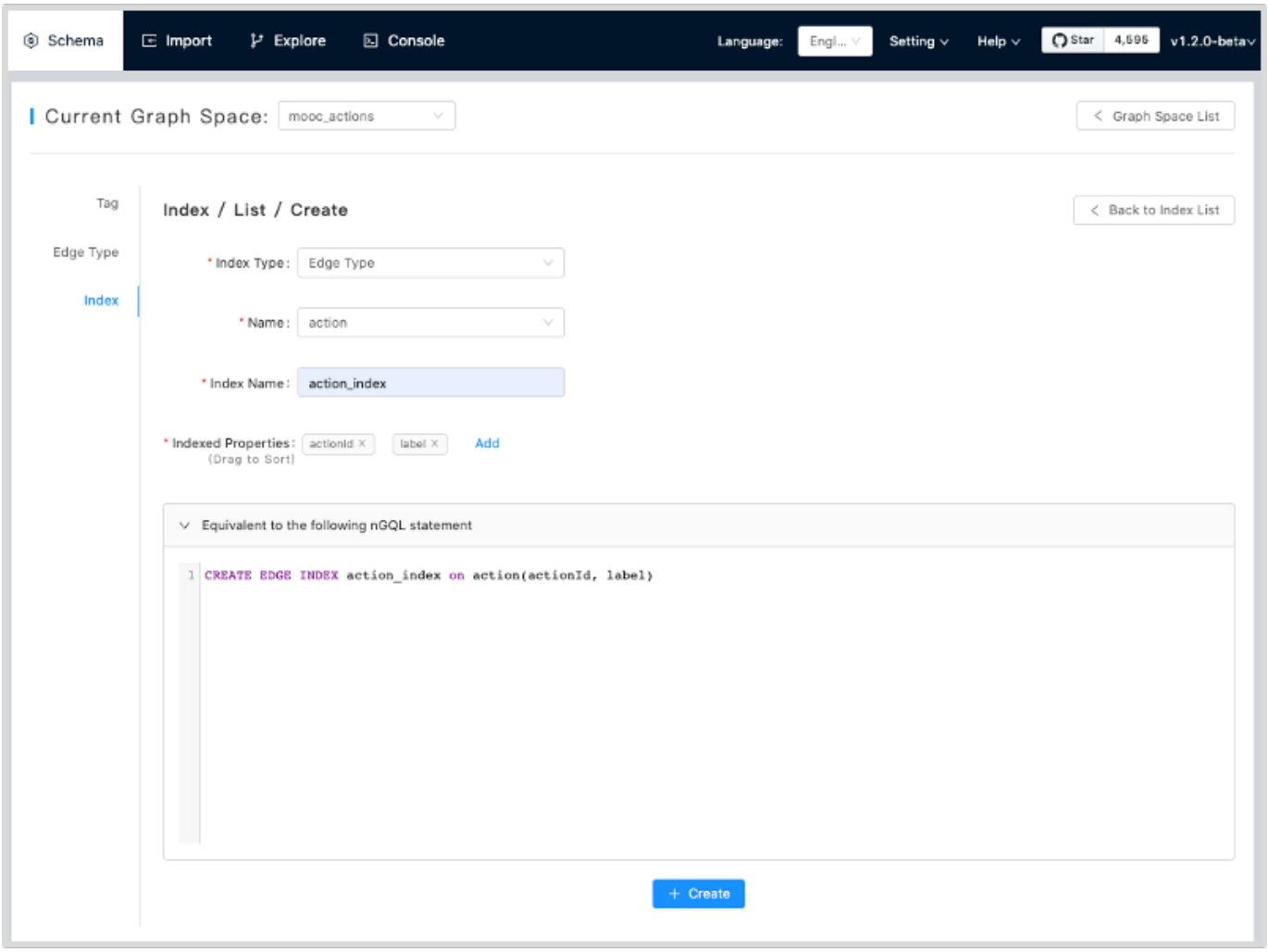
CREATE AN INDEX

To create an index on the **Schema** page, follow these steps:

1. In the toolbar, click the **Schema** tab.
2. On the **Graph Space List** page, find a graph space, and then click its name or the button 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.
   - **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.
   - **Indexed Properties**: Click **Add**, 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.

     > **Note**
     >
     > The order of the indexed properties has an effect on the result of the `LOOKUP` statement. For more information, see nGQL Manual.

   - **Comment**: The remarks of a certain property or the index itself. The maximum length is 256 bytes. By default, there will be no comments on an index. But in this example, `follow_index` is used.
6. When the settings are done, the **Equivalent to the following nGQL statement** panel shows the statement equivalent to the settings.



1. Confirm the settings and then click the **+ Create** button. When an index is created, the index list shows the new index.

VIEW INDEXES

To view indexes on the **Schema** page, follow these steps:

1. In the toolbar, click the **Schema** tab.
2. In the graph space list, find a graph space, and then click its name or the button 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.

DELETE AN INDEX

To delete an index on **Schema**, follow these steps:

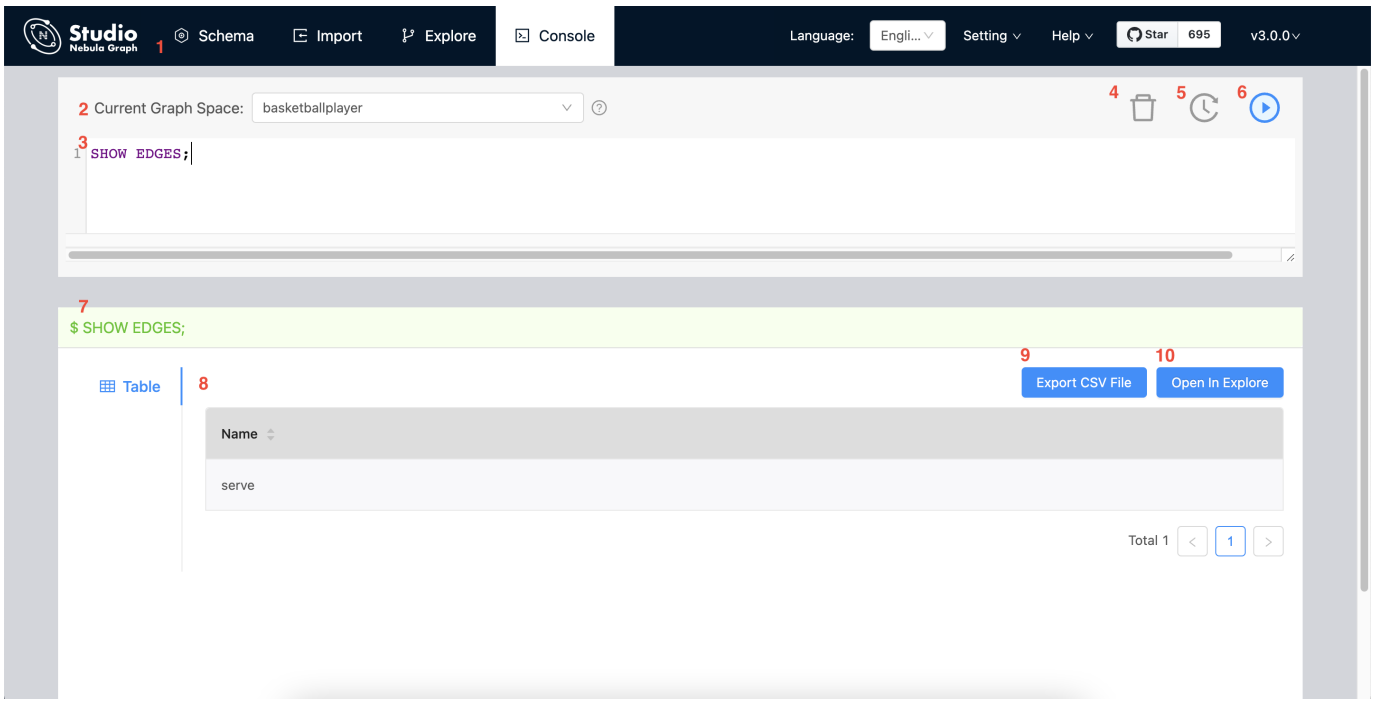
1. In the toolbar, click the **Schema** tab.
2. In the graph space list, find a graph space, and then click its name or the button  in the **Operations** column.
3. In the **Current Graph Space** field, confirm the name of the graph space. If necessary, you can choose another name to change the graph space.
4. Click the **Index** tab, find an index and then the button  in the **Operations** column.
5. Click **OK** to confirm in the pop-up dialog box.

Last update: September 3, 2021

11.5.2 Use Console

Console

Studio console interface as shown in following.



The following table lists various functions on the console interface.

number	function	descriptions
1	toolbar	Click the **Console** tab to enter the console page.
2	select a space	Select a space in the Current Graph Space list. **descriptions**: Studio does not support running the `USE <space_name>` statements directly in the input box.
3	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, separated by `;`.
4	clean input box	Click  button to clear the content entered in the input box.
5	history list	Click  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.
6	run	After inputting the nGQL statement in the input box, click  button to indicate the operation to start running the statement.
7	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.
8	result window	Display the results of the statement execution. If the statement returns results, the results window will display the returned results in tabular form.
9	export CSV file	After running the nGQL statement and return the result, click the **Export CSV File** button to export the result as a CSV file.
10	open in explore	According to the running nGQL statement, the user can click the graph exploration function key to import the returned results into graph exploration for visual display, such as open in explore and view subgraphs.

Last update: September 3, 2021

Open in Explore

With the **Open in Explore** function, you can run nGQL statements on the **Console** page to query vertex or edge data and then view the result on the **Explore** page in a visualized way.

SUPPORTED VERSIONS

Studio of v3.0.0 or later versions supports this function. For more information, see check updates.

PREREQUISITES

To use the **Open in Explore** function, you must do a check of these:

- Studio is connected to Nebula Graph. For more information, see Connect to Nebula Graph.
- A dataset exists in the database. For more information, see Import data.

QUERY AND VISUALIZE EDGE DATA

To query edge data on the **Console** page and then view the result on the **Explore** page, follow these steps:

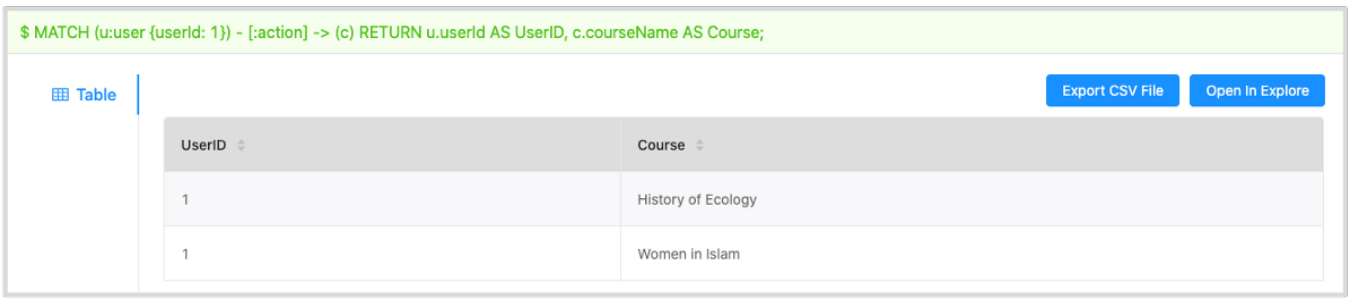
1. In the toolbar, click the **Console** tab.
2. In the **Current Graph Space** field, choose a graph space name. In this example, **basketballplayer** is chosen.
3. In the input box, enter an nGQL statement and click the button .

> **Note**
>
> The query result must contain the VIDs of the source vertex and the destination vertex of an edge.

Here is an nGQL statement example.

```
nebula> GO FROM "player102" OVER serve YIELD serve._src,serve._dst;
```

In the query result, you can see the start year and end year of the service team for the player whose playerId is `palyer102`. As shown below.



4. Click the **Open in Explore** button.
5. In the dialog box, configure as follows:

   a. Click **Edge Type**.

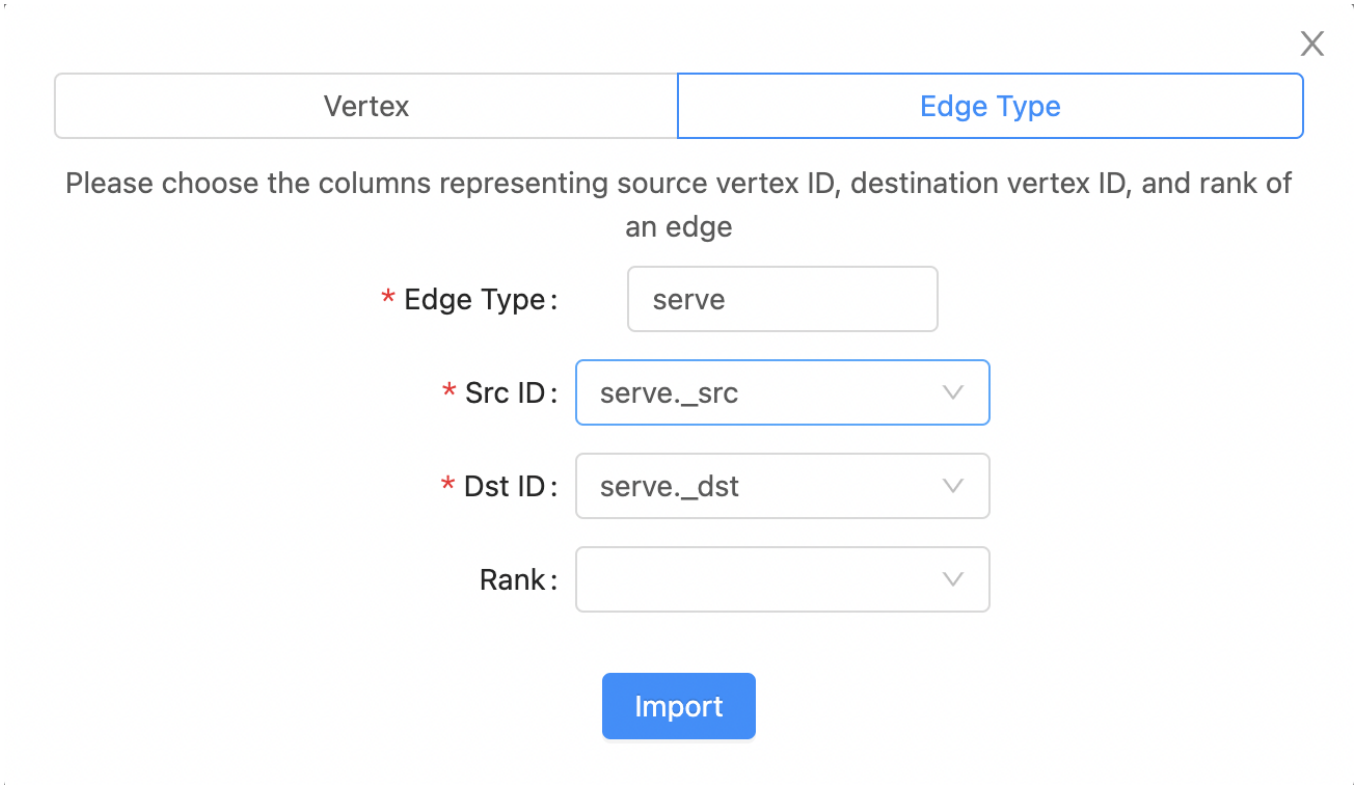
   b. In the **Edge Type** field, enter an edge type name. In this example, `serve` is used.

   c. In the **Src ID** field, choose a column name from the result table representing the VIDs of the source vertices. In this example, `serve._src` is chosen.

   d. In the **Dst ID** field, choose a column name from the result table representing the VIDs of the destination vertices. In this example, `serve._dst` is chosen.

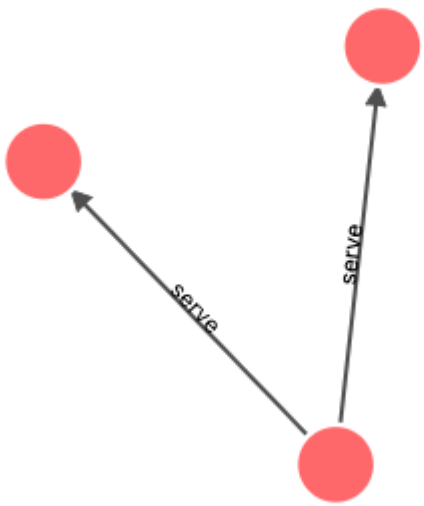
   e. (Optional) If the result table contains the ranking information of the edges, in the **Rank** field, choose a column name representing the `rank` of the edges. If no ranking information exists in the result, leave the **Rank** field blank.

   f. When the configuration is done, click the **Import** button.



6. If some data exists on the board of **Explore**, choose a method to insert data:
   - **Incremental Insertion**: Click this button to add the result to the existing data on the board.
   - **Insert After Clear**: Click this button to clear the existing data from the board and then add the data to the board.

When the data is inserted, you can view the visualized representation of the edge data.



QUERY AND VISUALIZE VERTEX DATA

To query vertex data on the **Console** page and then view the result on the **Explore** page, follow these steps:

1. In the toolbar, click the **Console** tab.
2. In the **Current Graph Space** field, choose a graph space name. In this example, **basketballplayer** is chosen.
3. In the input box, enter an nGQL statement and click the button .

> **Note**
>
> The query result must contain the VIDs of the vertices.

Here is an nGQL statement example.

```
nebula> FETCH PROP ON player "player100" YIELD player.name;
```

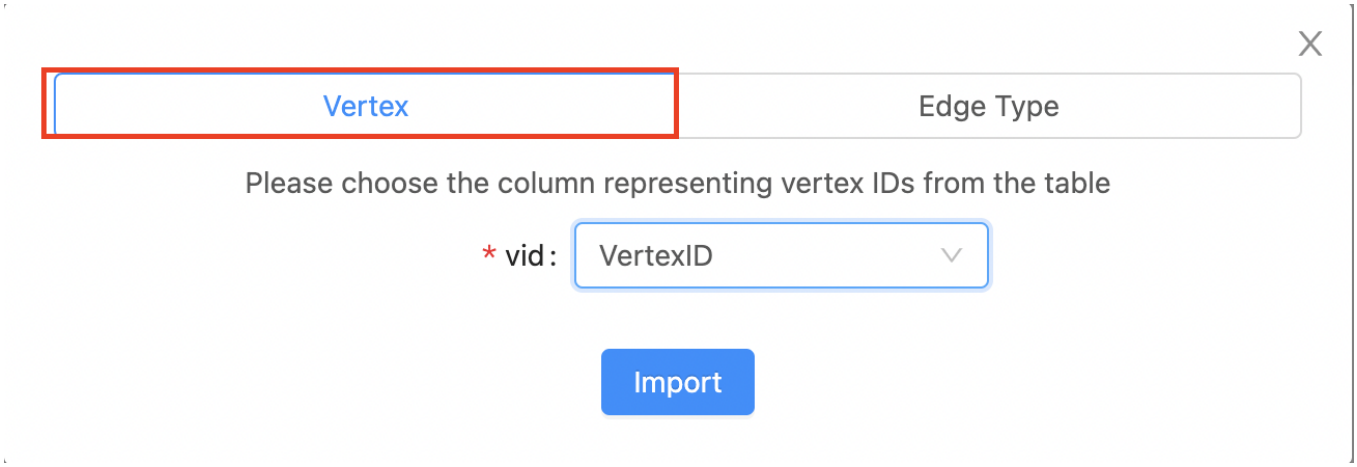
The query result gives the information of the player whose `playerId` is `player100`, as shown in this figure.

4. Click the **Open in Explore** button.
5. In the dialog box, configure as follows:

   a. Click **Vertex**.

   b. In the **Vertex ID** field, choose a column name from the result table representing the VIDs of the vertices. In this example, `VertexID` is chosen.

   c. When the configuration is done, click the **Import** button.



6. If some data exists on the board of **Explore**, choose a method to insert data:
   - **Incremental Insertion**: Click this button to add the queried result to the existing data on the board.
   - **Insert After Clear**: Click this button to clear the existing data from the board and then add the data.

When the data is inserted, you can view the visualized representation of the vertex data.

NEXT TO DO

On the **Explore** page, you can expand the board to explore and analyze graph data.

Last update: September 3, 2021

View subgraphs

With the **View Subgraphs** function, you can run a FIND SHORTEST | ALL PATH or a `GET SUBGRAPH` statement on the **Console** page and then view the result on the **Explore** page.

STUDIO VERSION

Studio of v3.0.0 supports this function. To update the version, see Check updates.

PREREQUISITES

To use the **View Subgraphs** function, you must do a check of these:

- The version of Studio is v3.0.0 or later.
- Studio is connected to Nebula Graph.
- A dataset exists in the database. In the example of this article, the **basketballplayer** dataset is used. For more information, see Import data.

> **Note**
>
> Users can view subgraphs online in Studio.

PROCEDURE

To query the paths or subgraph on the **Console** page and then view them on the **Explore** page, follow these steps:

1. In the navigation bar, click the **Console** tab.
2. In the **Current Graph Space** dropdown list, choose a graph space name. In this example, **baskteballplayer** is chosen.
3. In the input box, enter a `FIND SHORTEST PATH`, `FIND ALL PATH`, or `GET SUBGRAPH` statement and click **Run**.

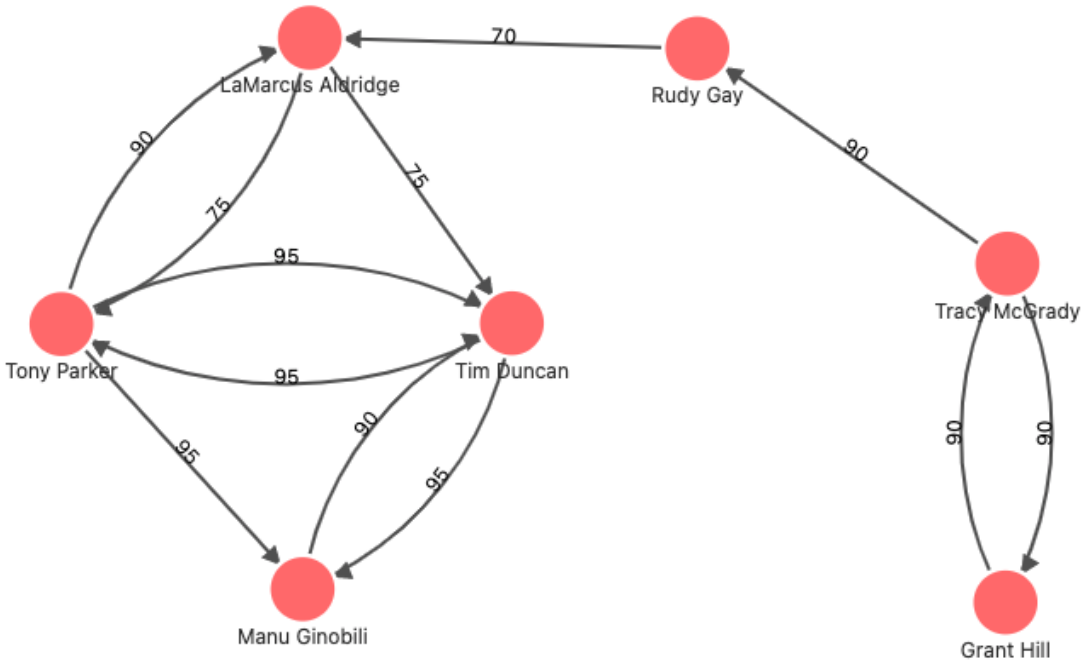
   Here is an nGQL statement example.

```
// Run FIND ALL PATH
nebula> FIND ALL PATH FROM "player114" to "player100" OVER follow;
```

   Take the `FIND ALL PATH` for example, query the path information as shown in this figure.

4. Click the **View Subgraphs** button.
5. (Optional) If some data exists on the board of **Explore**, choose a method to insert data:
   - **Incremental Insertion**: Click this button to add the result to the existing data on the board.
   - **Insert After Clear**: Click this button to clear the existing data from the board and then add the data to the board.

When the data is inserted, you can view the visualized representation of the paths.



NEXT TO DO

On the **Explore** page, you can expand the graph to explore and analyze graph data.

Last update: September 3, 2021

11.6 Troubleshooting

11.6.1 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 Nebula Graph database Graph service. If no changes are made, the port defaults to `9669` . Even if Nebula Graph 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 NEBULA GRAPH SERVICE IS NORMAL

Check Nebula Graph service status. Regarding the operation of viewing services:

- If you compile and deploy Nebula Graph on a Linux server, refer to the Nebula Graph service.
- If you use Nebula Graph deployed by Docker Compose and RPM, refer to the Nebula Graph service status and ports.

If the Nebula Graph service is normal, proceed to Step 4 to continue troubleshooting. Otherwise, please restart Nebula Graph service.

> **Note**
>
> If you used `docker-compose up -d` to satrt Nebula Graph before, you must run the `docker-compose down` to stop Nebula Graph.

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 Nebula Graph's Graph service network connection is normal.

If the connection fails, check according to the following steps:

- If Studio and Nebula Graph are on the same machine, check if the port is exposed.
- If Studio and Nebula Graph are not on the same machine, check the network configuration of the Nebula Graph server, such as firewall, gateway, and port.

If you cannot connect to the Nebula Graph service after troubleshooting with the above steps, please go to the Nebula Graph forum for consultation.

Last update: September 3, 2021

11.6.2 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

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      0.0.0.0:32782->8080/tcp
nebula-web-docker_importer_1   nebula-importer --port=569 ...   Up      0.0.0.0:32783->5699/tcp
nebula-web-docker_nginx_1      /docker-entrypoint.sh ngin ...   Up      0.0.0.0:7001->7001/tcp, 80/tcp
nebula-web-docker_web_1        docker-entrypoint.sh npm r ...   Up      0.0.0.0:32784->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 Nebula Graph before, you must run the `docker-compose down` to stop Nebula Graph.

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 Nebula Graph are on the same machine, check if the port is exposed.
- If Studio and Nebula Graph are not on the same machine, check the network configuration of the Nebula Graph server, such as firewall, gateway, and port.

If you cannot connect to the Nebula Graph service after troubleshooting with the above steps, please go to the Nebula Graph forum for consultation.

Last update: September 1, 2021

11.6.3 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:

1. Confirm that Nebula Graph is the latest version. If you use Docker Compose to deploy the Nebula Graph 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.

Last update: September 3, 2021

12. Nebula Dashboard

12.1 What is Nebula Dashboard

Nebula Dashboard (Dashboard for short) is a visualization tool that monitors the status of machines and services in Nebula Graph clusters.

12.1.1 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.
- Features of the enterprise package (TODO: planning)

> **Enterpriseonly**
>
> Dashboard has two editions. One is open source, the other is for enterprises. Most of the functions are available in both. The functions only supported by the enterprise package will be marked and explained.

12.1.2 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.

12.1.3 Precautions

- The monitoring data will be updated per 7 seconds by default.
- 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.

Last update: September 24, 2021

12.2 Deploy Dashboard

The deployment of Dashboard involve five services. This topic will describe how to deploy Dashboard in detail.

12.2.1 Nebula Graph releases

The correspondence between the Dashboard release and the Nebula Graph release is as follows.

Dashboard	Nebula Graph
1.0.1	2.x

12.2.2 Port

The deployment of Dashboard occupies the following ports:

- 9200
- 9100
- 9090
- 8090
- 7003

12.2.3 Download Dashboard

Download the tar package as needed, and it is recommended to select the latest version.

Dashboard package	Nebula Graph version
nebula-graph-dashboard-1.0.1.x86_64.tar.gz	v2.5.0

12.2.4 Service

Run `tar -xvf 1.0.1.tar.gz` to decompress the installation package. There are 5 services in the `nebula-graph-dashboard`. The descriptions are as follows.

Name	Description
node-exporter	Collects the source information of machines in the cluster, including the CPU, memory, load, disk, and network.
nebula-stats-exporter	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).
prometheus	The time series database that stores monitoring data.
nebula-http-gateway	Provides HTTP ports for cluster services to execute nGQL statements to interact with the Nebula Graph database.
nebula-graph-dashboard	Provides the Dashboard service. Note that its name is the same as its superordinate. The following `nebula-graph-dashboard` refers to this service.

The above five services should be deployed as follows.

12.2.5 Procedure

Deploy `node-exporter`

> **Note**
>
> You need to deploy the `node-exporter` service on each machine in the cluster.

To start the service, run the following statement in `node-exporter`:

```
$ nohup ./node-exporter --web.listen-address=":9100" &
```

After the service is started, you can enter `<IP>:9100` in the browser to check whether the service is started normally.

Deploy `nebula-stats-exporter`

> **Note**
>
> You only need to deploy the `nebula-stats-exporter` service on the machine where the `nebula-graph-dashboard` service is installed.

1. Modify the `config.yaml` file in `nebula-stats-exporter` to deploy the HTTP ports of all the services. The example is as follows:

```
version: v0.0.4
clusters:
  - name: nebula
    instances:
      - name: metad0
        endpointIP: 192.168.8.157
        endpointPort: 19559
        componentType: metad
      - name: metad1
        endpointIP: 192.168.8.155
        endpointPort: 19559
        componentType: metad
      - name: metad2
        endpointIP: 192.168.8.154
        endpointPort: 19559
        componentType: metad
      - name: graphd0
        endpointIP: 192.168.8.157
        endpointPort: 19669
        componentType: graphd
      - name: graphd1
        endpointIP: 192.168.8.155
        endpointPort: 19669
        componentType: graphd
      - name: graphd2
        endpointIP: 192.168.8.154
        endpointPort: 19669
        componentType: graphd
      - name: storaged0
        endpointIP: 192.168.8.157
        endpointPort: 19779
        componentType: storaged
      - name: storaged1
        endpointIP: 192.168.8.155
        endpointPort: 19779
        componentType: storaged
      - name: storaged2
        endpointIP: 192.168.8.154
        endpointPort: 19779
        componentType: storaged
```

2. Run the following statement to start the service:

```
$ nohup  ./nebula-stats-exporter --listen-address=":9200" --bare-metal --bare-metal-config=./config.yaml &
```

After the service is started, you can enter `<IP>:9200` in the browser to check whether the service is started normally.

Deploy `prometheus`

> **Note**
>
> You only need to deploy the `prometheus` service on the machine where the `nebula-graph-dashboard` service is installed.

1. Modify the `prometheus.yaml` file in `prometheus` to deploy the IP addresses and ports of the `node-exporter` service and the `nebula-stats-exporter`. The example is as follows:

```
global:
  scrape_interval:     5s
  evaluation_interval: 5s
scrape_configs:
  - job_name: 'nebula-exporter'
    static_configs:
      - targets: [
          '192.168.xx.100:9200',  # The IP address and port of the nebula-stats-exporter service.
          '192.168.xx.101:9200'
        ]
  - job_name: 'node-exporter'
    static_configs:
      - targets: [
          '192.168.xx.101:9100'  # The IP address and port of the node-exporter service.
        ]
```

   - scrape_interval: The interval for collecting the monitoring data, which is 1 minute by default.
   - evaluation_interval: The interval for running alarm rules, which is 1 minute by default.

2. Run the following statement to start the service.

```
$ nohup ./prometheus --config.file=./prometheus.yaml &
```

After the service is started, you can enter `<IP>:9090` in the browser to check whether the service is started normally.

Deploy `nebula-http-gateway`

> **Note**
>
> You only need to deploy the `nebula-http-gateway` service on the machine where the `nebula-graph-dashboard` service is installed.

To start the service, run the following statement in `nebula-http-gateway`:

```
$ nohup ./nebula-httpd &
```

After the service is started, you can enter `<IP>:8090` in the browser to check whether the service is started normally.

How to deploy the `nebula-graph-dashboard` service

1. Modify the `custom.json` file in `nebula-graph-dashboard/static/` to deploy the IP address and port of the Graph Service. The example is as follows:

```
{
    "connection": {
        "ip": "192.168.xx.4",
        "port": 9669
    },
    "alias": {
        "ip:port": "instance1"
    },
    "chartBaseLine": {

    }
}
...
```

2. To start the service, run the following statement in `nebula-graph-dashboard`:

```
$ npm run start
```

After the service is started, you can enter `<IP>:7003` in the browser to check whether the service is started normally.

12.2.6 Stop Dashboard

You can enter `kill <pid>` to stop Dashboard. The examples are as follows:

```
$ kill $(lsof -t -i :9100) # stop the node-exporter service
$ kill $(lsof -t -i :9200) # stop the nebula-stats-exporter service
$ kill $(lsof -t -i :9090) # stop the prometheus service
$ kill $(lsof -t -i :8090) # stop the nebula-http-gateway service
$ cd nebula-graph-dashboard
$ npm run stop # stop the nebula-graph-dashboard service
```

Last update: September 3, 2021

12.3 Connect Dashboard

After Dashboard is deployed, you can log in and use Dashboard on the browser.

12.3.1 Prerequisites

- The Dashboard services are started. For more information, see Deploy Dashboard.
- We recommend you to use the Chrome browser of the version above 58. Otherwise, there may be compatibility issues.

12.3.2 Procedure

1. Confirm the IP address of the machine where the `nebula-graph-dashboard` service is installed. Enter `<IP>:7003` in the browser to open the login page.
2. Enter the username and the passwords of the Nebula Graph database and click the login button.
   - 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.

Nebula Dashboard

Account Login

root

••••••

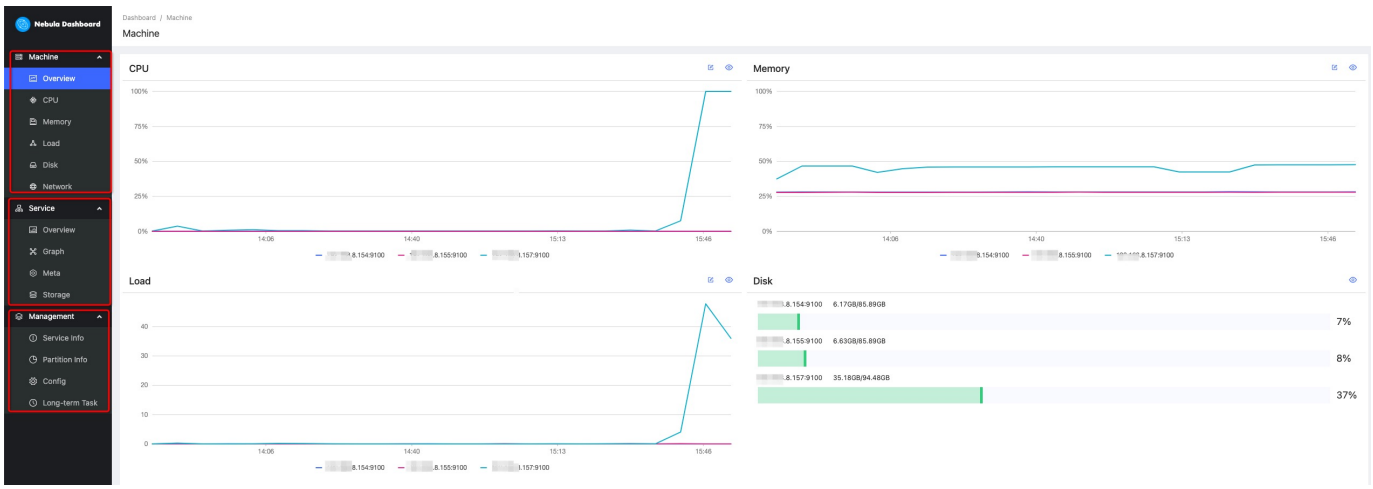
Login

Last update: September 2, 2021

12.4 Dashboard

Nebula Dashboard consists of three parts: Machine, Service, and Management. This topic will describe them in detail.

12.4.1 Overview



12.4.2 Machine

Machine consists of the following parts:

- Overview

  You can check the fluctuations of CPU, Memory, Load, Disk, Network In, and Network Out in the past 24 hours.

  For details of certain monitoring metrics, you can click the  symbol in the upper right corner, or click the monitoring metrics on the left.

- CPU, Memory, Load, Disk, Network

  It shows the detailed monitoring data of the machine from the above dimensions.

  - By default, you can check the monitoring data up to 14 days before. The alternative can be 1 hour, 6 hours, 12 hours, 1 day, 3 days, 7 days, or 14 days in the past.
  - You can choose the machine and monitoring metrics that you want to check. For more information, see monitor parameter.
  - You can set a base line as a reference.

12.4.3 Service

Service consists of the following parts:

- Overview

  You can check the fluctuations of monitoring metrics of various services in the past 24 hours. You can also switch to the **Version** page to view the IP addresses and versions of all services.

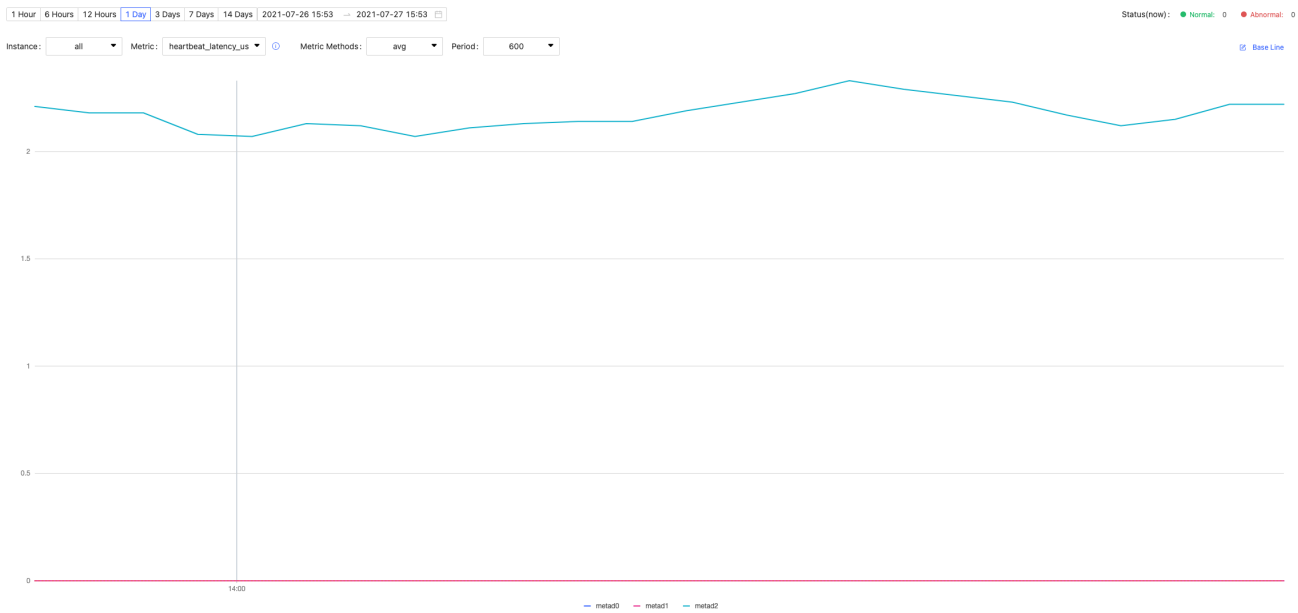
  For details of certain monitoring metrics, you can click the  symbol in the upper right corner, or click the services on the left.

  > **Note**
  >
  > The overview page of the current open source edition only supports setting two monitoring metrics for each service. You can adjust it by clicking the **Set up** button.

- Graph, Meta, Storage

  It shows the detailed monitoring data of the above services.

  - By default, you can check the monitoring data up to 14 days before. The alternative can be 1 hour, 6 hours, 12 hours, 1 day, 3 days, 7 days, or 14 days in the past.
  - You can choose the machine that you want to check the monitoring data, monitoring metrics, metric methods, and period. For more information, see monitor parameter.
  - You can set a base line as a reference.
  - You can check the status of the current service.



12.4.4 Management

> **Note**
>
> Non-root users can view the service information and the partition information with spatial permissions, but cannot view the configuration and long-term tasks.

Management consists of the following parts:

- Service Info

  It shows the basic information of the Storage Service, including the information of the host, the number of leaders, the distribution of partitions, and the distribution of leaders.

- Partition Info

  You can check the information of partitions in different graph spaces. The descriptions are as follows.

Parameter	Description
`Partition ID`	The ID of the partition.
`Leader`	The IP address and the port of the leader.
`Peers`	The IP addresses and the ports of all the replicas.
`Losts`	The IP addresses and the ports of replicas at fault.

- Config

  It shows the configuration of each service. Dashboard does not support online modification of configurations for now. For details, see configurations.

- Long-term Task

  It shows the information of all jobs. Dashboard does not support online management of jobs for now. For details, see job statements.

12.4.5 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

Last update: September 2, 2021

12.5 Metrics

This topic will describe the monitoring metrics in Nebula Dashboard.

> **Note**
>
> The default unit in **Disk** and **Network** 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.

12.5.1 Machine

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 (including caches).
`memory_actual_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`	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.

12.5.2 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.

Graph

Parameter	Description
num_queries	The number of queries.
num_slow_queries	The number of slow queries.
query_latency_us	The average latency of queries.
slow_query_latency_us	The average latency of slow queries.
num_query_errors	The number of queries in error.

Meta

Parameter	Description
heartbeat_latency_us	The latency of heartbeats.
num_heartbeats	The number of heartbeats.

Storage

Parameter	Description
add_edges_latency_us	The average latency of adding edges.
add_vertices_latency_us	The average latency of adding vertices.
delete_edges_latency_us	The average latency of deleting edges.
delete_vertices_latency_us	The average latency of deleting vertices.
forward_tranx_latency_us	The average latency of transmitting.
get_neighbors_latency_us	The average latency of querying neighbors.

Last update: September 24, 2021

13. Nebula Explorer

13.1 What is Nebula Explorer

Nebula Explorer (Explorer in short) is a browser-based visualization tool. It is used with the Nebula Graph core to visualize interaction with graph data. Even if there is no experience in graph database, you can quickly become a graph exploration expert.

> **Enterpriseonly**
>
> Explorer is only available in the enterprise version.

> **Note**
>
> You can also try some functions online in Explorer.

13.1.1 Scenarios

You can use Explorer in one of these scenarios:

- You need to quickly find neighbor relationships from complex relationships, analyze suspicious targets, and display graph data in a visual manner.
- For large-scale data sets, the data needs to be filtered, analyzed, and explored in a visual manner.

13.1.2 Features

Explorer has these features:

- Easy to use and user-friendly: Explorer can be deployed in simple steps. And use simple visual interaction, no need to conceive nGQL sentences, easy to realize graph exploration.
- Flexible: Explorer supports querying data through VID, Tag, Subgraph.
- Multiple operations: Explorer supports operations such as expanding operations on multiple vertexes, querying the common neighbors of multiple vertexes, and querying the path between the start vertex and the end vertex.
- Various display: Explorer supports modifying the color and icon of the vertex in the canvas to highlight key nodes. You can also freely choose the data display mode in `dagre`, `force`, and `circular`.

13.1.3 Authentication

Authentication is not enabled in Nebula Graph by default. Users can log into Studio with the `root` account and any password.

When Nebula Graph enables authentication, users can only sign into Studio with the specified account. For more information, see Authentication.

Last update: September 24, 2021

13.2 Deploy and connect

13.2.1 Deploy Explorer

This topic describes how to deploy Explorer locally by RPM, and tar package.

RPM-based Explorer

PREREQUISITES

Before you deploy Explorer, you must do a check of these:

- The Nebula Graph services are deployed and started. For more information, see Nebula Graph Database Manual.
- Before the installation starts, the following ports are not occupied.

Port	Description
7002	Web service provided by Explorer
8070	Nebula-http-gateway service

- The Linux distribution is CentOS, installed with Node.js of version above v10.16.0+ and Go of version above 1.13.

> **Caution**
>
> At present, the package provided by Nebula Explorer can be used in Linux environment only. If users use mac or other environments, clone http-gateway repo, modify the `httpport = 8070` in the `nebula-http-gateway/conf/app.conf` file, and use the `make` command to compile and start Nebula Explorer.

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:

> **Enterpriseonly**
>
> Explorer is only available in the enterprise version. Click Pricing to see more.

2. Use `sudo rpm -i <rpm>` to install RPM package.

   For example, install Explorer, use the following command:

```
$ sudo rpm -i nebula-graph-explorer-<version>.x86_64.rpm
```

UNINSTALL

Users can uninstall Explorer using the following command:

```
$ sudo rpm -e nebula-graph-explorer-<version>.x86_64
```

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.

```
$ sudo sh ./scripts/start.sh
```

- Stop the service manually

```
$ sudo sh ./scripts/stop.sh
```

tar-based Explorer

PREREQUISITES

Before you deploy Explorer, you must do a check of these:

- The Nebula Graph services are deployed and started. For more information, see Nebula Graph Database Manual.
- Before the installation starts, the following ports are not occupied.

Port	Description
7002	Web service provided by Explorer
8070	Nebula-http-gateway service

- The Linux distribution is CentOS, installed with Node.js of version above v10.16.0+ and Go of version above 1.13.

> **Caution**
>
> At present, the package provided by Nebula Explorer can be used in Linux environment only. If users use mac or other environments, clone http-gateway repo, modify the `httpport = 8070` in the `nebula-http-gateway/conf/app.conf` file, and use the `make` command to compile and start Nebula Explorer.

INSTALL

1. Select and download the tar package according to your needs. It is recommended to select the latest version. Common links are as follows:

> **Enterpriseonly**
>
> Explorer is only available in the enterprise version. Click Pricing to see more.

2. Use `tar -xvf` to decompress the tar package.

```
tar -xvf nebula-graph-explorer-<version>.tar.gz
```

PROCEDURE

> **Note**
>
> The root directory `nebula-graph-explorer` has two installation packages: nebula-graph-explorer and nebula-http-gateway. You need to deploy and start the services separately on the same machine to complete the deployment of Explorer.'

1. Deploy and start nebula-http-gateway.

```
$ cd nebula-http-gateway
$ nohup ./nebula-httpd &
```

2. Deploy and start nebula-graph-explorer.

```
$ cd nebula-graph-explorer
$ npm run start
```

STOP SERVICE

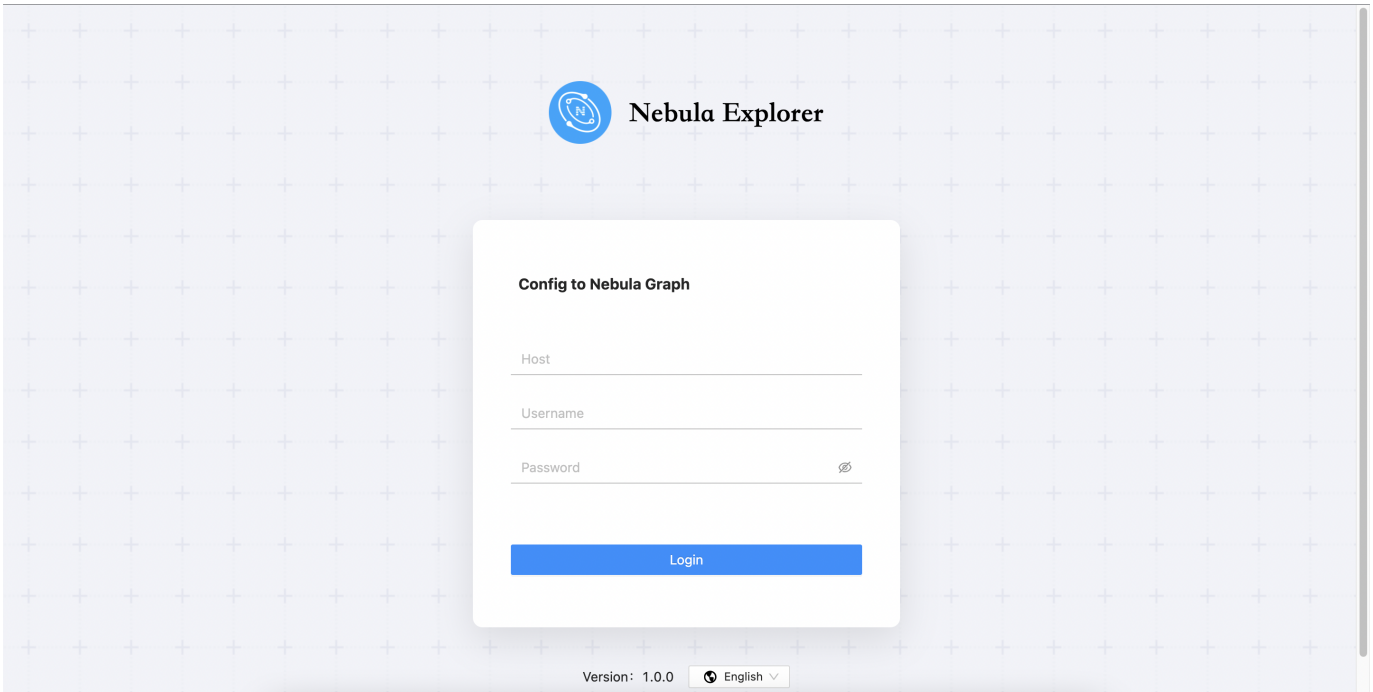
You can use `kill pid` to stop the service:

```
$ kill $(lsof -t -i :8070) # stop nebula-http-gateway
$ cd nebula-graph-explorer
$ npm run stop # stop nebula-graph-explorer
```

Next to do

When Explorer is started, use `http://ip address:7002` to get access to Explorer.

Seeing the following login interface, Explorer is successfully connected to Nebula Graph.



After entering the Explorer login interface, you need to connect to Nebula Graph. For more information, refer to Connecting to the Nebula Graph.

Last update: September 24, 2021

13.2.2 Connect to Nebula Graph

After successfully launching Explorer, you need to configure to connect to Nebula Graph. This topic describes how Explorer connects to the Nebula Graph database.

Prerequisites

Before connecting to the Nebula Graph database, you need to confirm the following information:

- The Nebula Graph services and Explorer are started. For more information, see Deploy Explorer.
- You have the local IP address and the port used by the Graph service of Nebula Graph. The default port is `9669`.
- You have a Nebula Graph account and its password.

> **Note**
>
> If authentication is enabled in Nebula Graph and different role-based accounts are created, you must use the assigned account to connect to Nebula Graph. If authentication is disabled, you can use the `root` and any password to connect to Nebula Graph. For more information, see Nebula Graph Database Manual.

Procedure

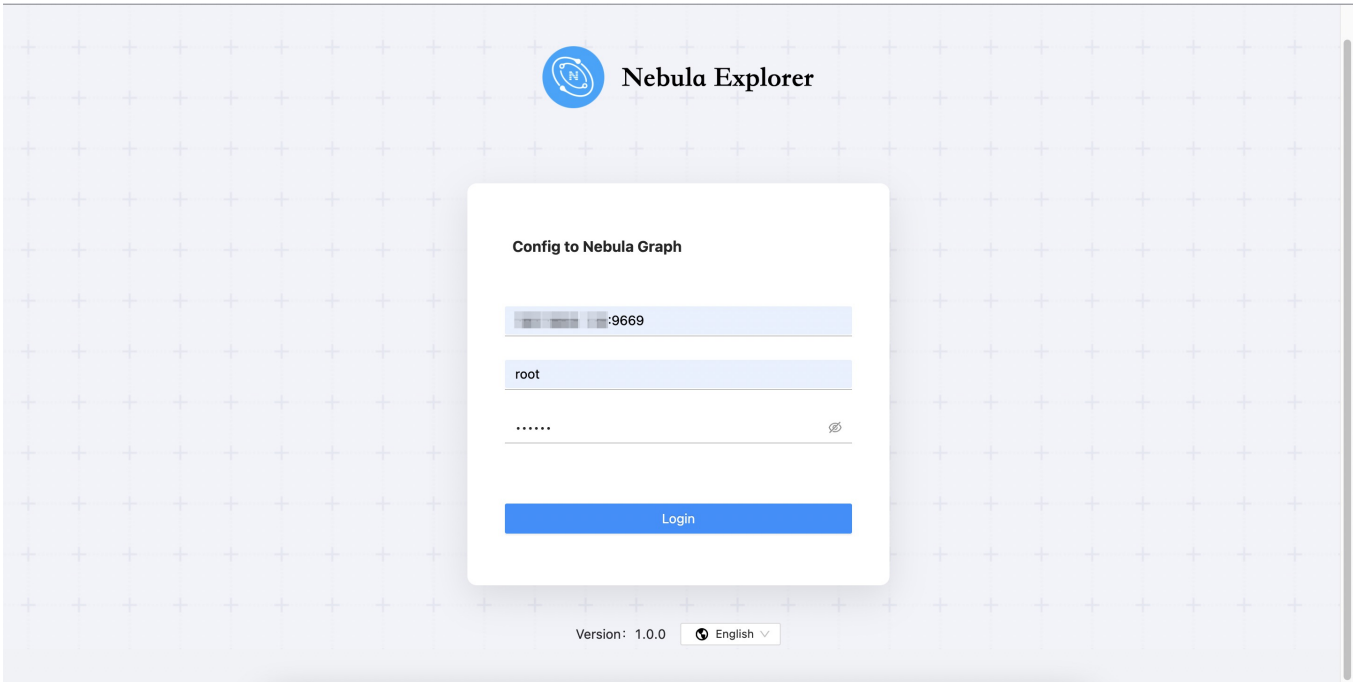
To connect Explorer to Nebula Graph, follow these steps:

1. On the **Config Server** page of Explorer, configure these fields:

   - **Host**: Enter the IP address and the port of the Graph service of Nebula Graph. The valid format is `IP:port`. The default port is `9669`.

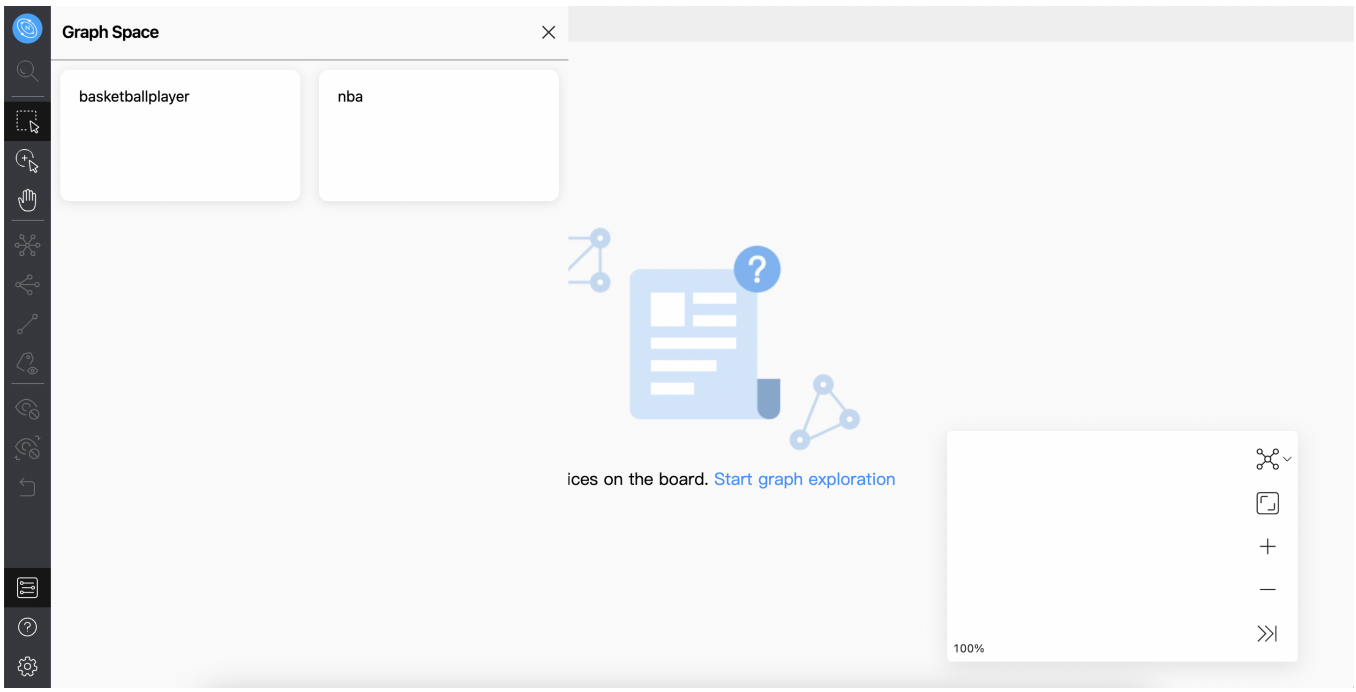
     > **Note**
     >
     > When Nebula Graph and Explorer are deployed on the same machine, you must enter the IP address of the machine, but not `127.0.0.1` or `localhost`, in the **Host** field.

   - **Username** and **Password**: Fill in the log in account according to the authentication settings of Nebula Graph.
     - 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.



2. After the configuration, click the **Connect** button.

   If you can see the interface as shown in the below, it means you have successfully connected to the Nebula Graph database.

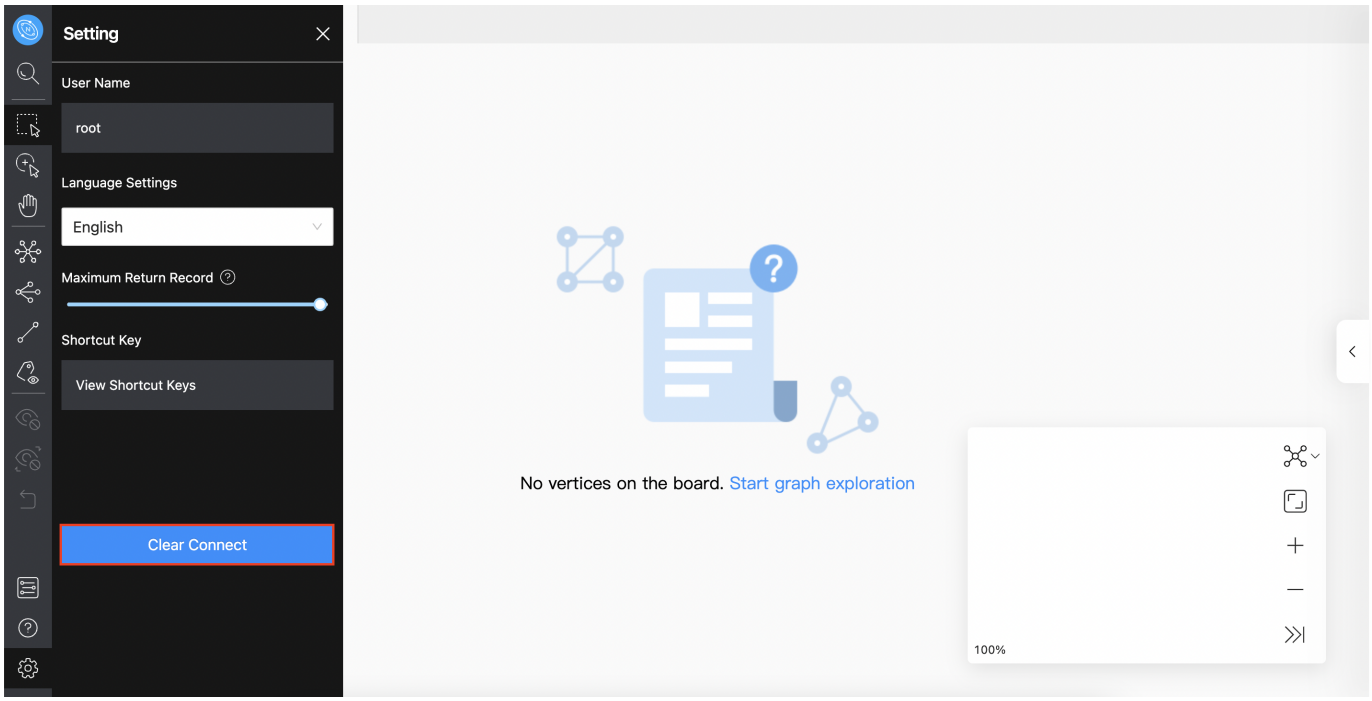


One session continues for up to 30 minutes. If you do not operate Explorer within 30 minutes, the active session will time out and you must connect to Nebula Graph again.

Last update: September 2, 2021

13.2.3 Reset connection

When Explorer is still connected to a Nebula Graph database, in the toolbar, select Settings > clear connect, as shown in the below:



After that, if the **configuration database** page is displayed on the browser, it means that Explorer has successfully disconnected from the Nebula Graph.

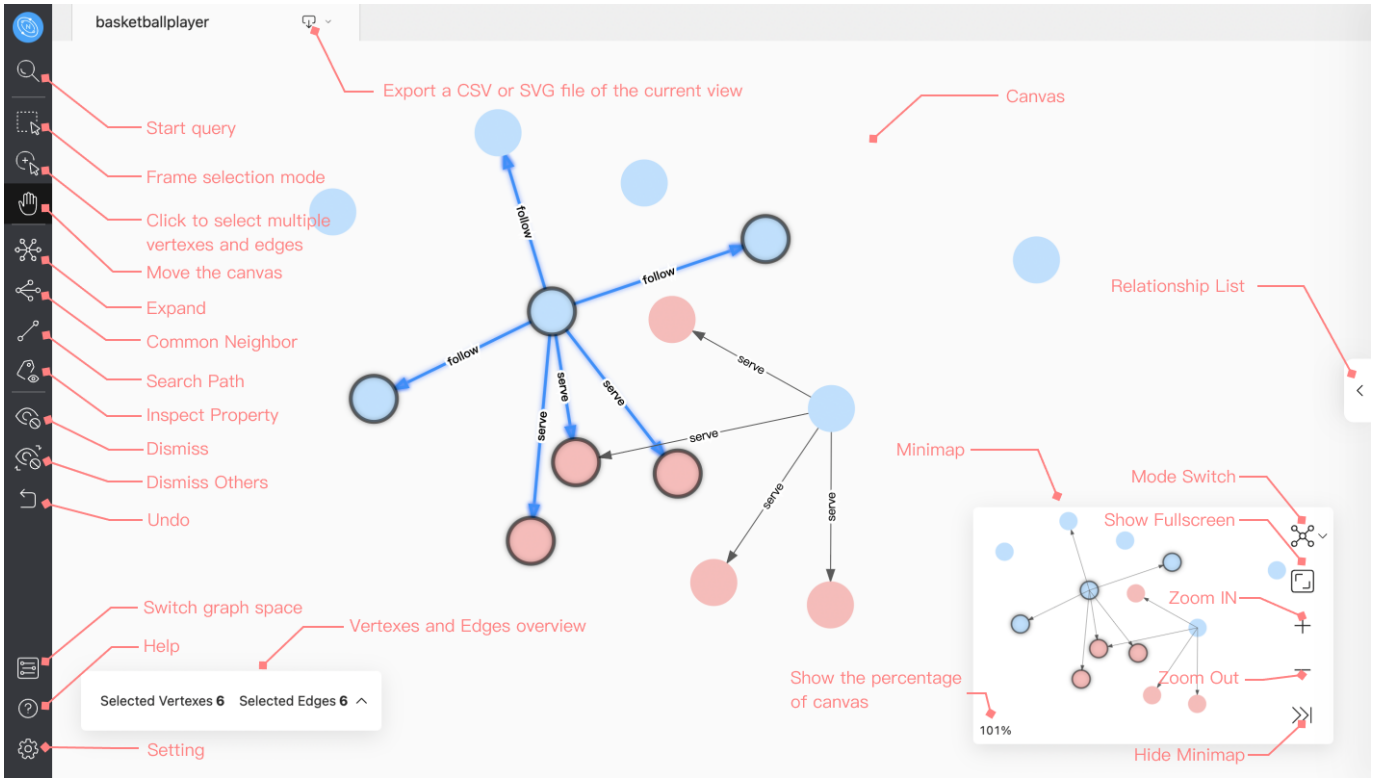
Last update: September 2, 2021

13.3 Operation guide

13.3.1 Page Overview

This topic describes Explorer main page.

Overview



The main page of Explorer is divided into five parts:

- Tab bar
- Sidebar
- Canvas
- Minimap
- Relationship list

Tab bar

- Export: Export a CSV or PNG file of the current view.

Sidebar

The sidebar consists of five parts. You can click the buttons to explore the graph, modify the content of the vertexes on the canvas, etc.

- Start query: Before exploring, the user needs to query the vertexes and display them in the canvas.
- Canvas operation: Including frame selection of vertexes in the canvas, dragging the canvas, and selecting multiple vertexes and edges.
- Graph exploration and expansion: Including functions such as vertexes expansion, finding common neighbors of multiple vertexes, finding the path of two vertexes, and inspecting the property.
- Hide and undo: Hide the data displayed in the canvas and undo the previous operation.
- Settings and help: Switch graph space, find help, modify settings, etc.

START QUERY

- Start: Click the  icon to query the data and display it on the page through VID, Tag and sub-graph.

CANVAS OPERATION

- Frame selection mode: Click the  icon to support frame selection of vertexes and edges in the canvas.
- Click to select multiple vertexes and edges: Click the  icon, you can easily click the vertexes and edges in the canvas, and click the blank space to cancel the selection.
- Move the canvas: Click the  icon to drag the position of the canvas.

For more information, see Canvas Operation.

GRAPH EXPLORATION AND EXPANSION

- Expand: Click the  icon, select the vertexes on the page and perform custom expansion, including direction, steps, filter conditions, etc.
- Common neighbor: Click the  icon, select at least two vertexes on the page and view their common neighbors.
- Search path: Click the  icon to query the path of `all paths`, `Shortest path` or `Noloop path` between the start vertex and the end vertex.
- Inspect property: Click the  icon to choose whether to display the property values of vertexes or edges in the canvas.

For more information, see Graph exploration and expansion.

HIDE AND UNDO

- Dismiss: Click the  icon to hide the selected vertexes and edges in the canvas.
- Dismiss others: Click the  icon to hide all unselected vertexes and edges in the canvas.
- Undo: Click the  icon to undo the operation in the previous step.

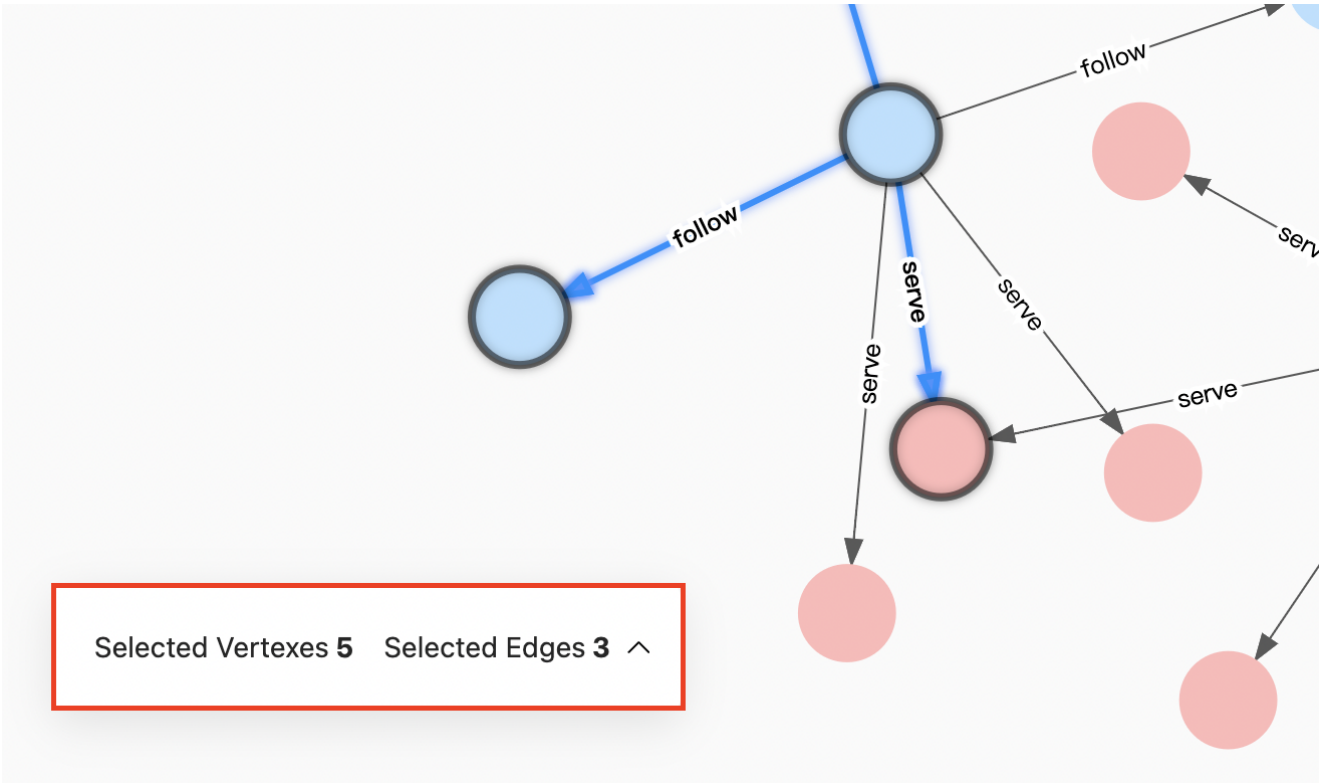
SETTINGS AND HELP

- Switch graph space: Click the  icon to switch the current graph space.
- Help: Click the  icon to see more information.
- Setting: Click the  icon to view usernames and shortcut keys, modify language settings, clear Explorer connect, etc.

Canvas

The canvas is mainly divided into:

- Canvas: Display the data queried by VID, Tag or subgraph.
- Vertexes and Edges overview: It is hidden by default and only displayed when the vertex and edge are selected on the current canvas. Click on the icon in the following, and the user can open the menu to view the detailed data of the selected vertexes and edges in the current canvas.



For more information, see canvas operation.

Minimap

You can use the button on the minimap to switch the graph mode, display the vertexes in the canvas in full screen, collapse the minimap, zoom in or zoom out the canvass, etc. At the same time, the percentage of the graph in the canvas to the total graph is displayed in the lower-left corner of the minimap.

- Switch mode: You can switch the display mode of the graph in the canvas.

icon			
mode	force	dagre	circular

Relationship list

Click the icon on the right, you can open the menu, view the number of tags and edges in the canvas, search for tags and edges, and also support modifying the color and icon of the vertex.

Last update: September 2, 2021

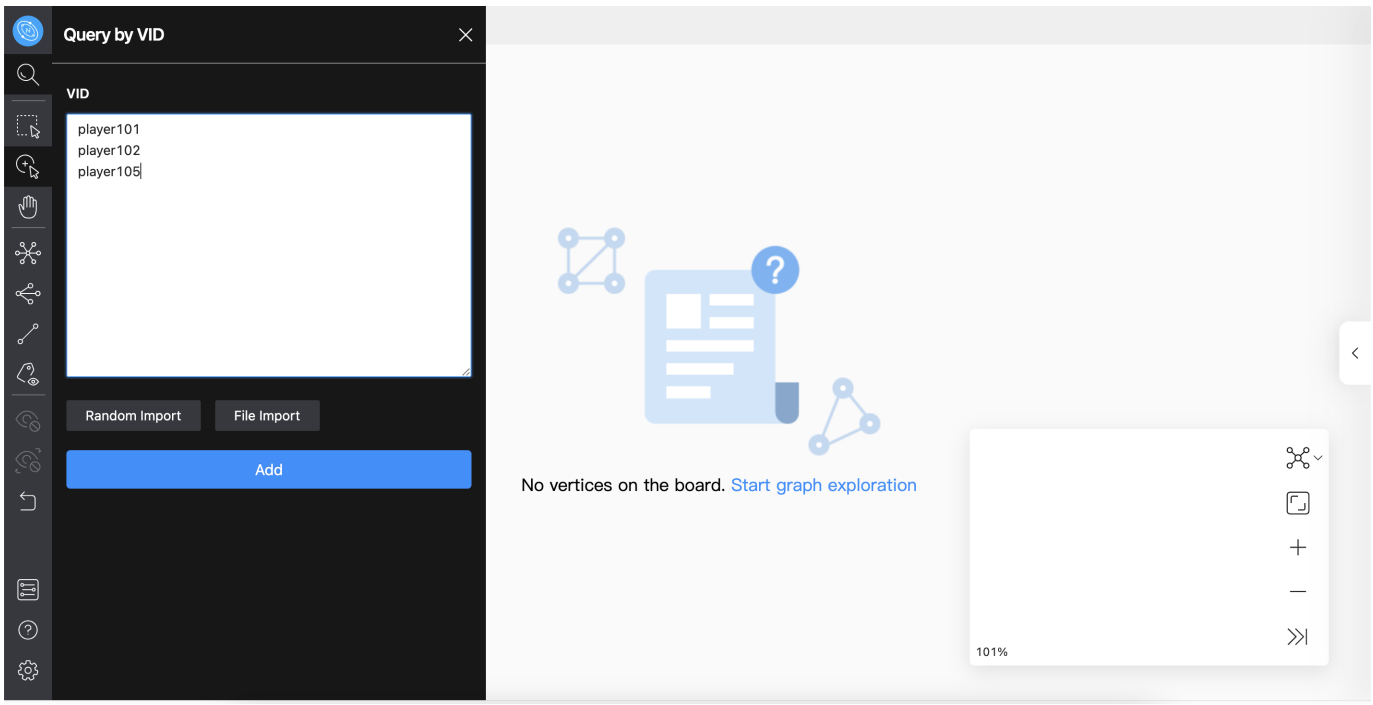
13.3.2 Start query

In Explorer, you can choose the following query methods to display data:

- VID
- Tag
- Subgraph

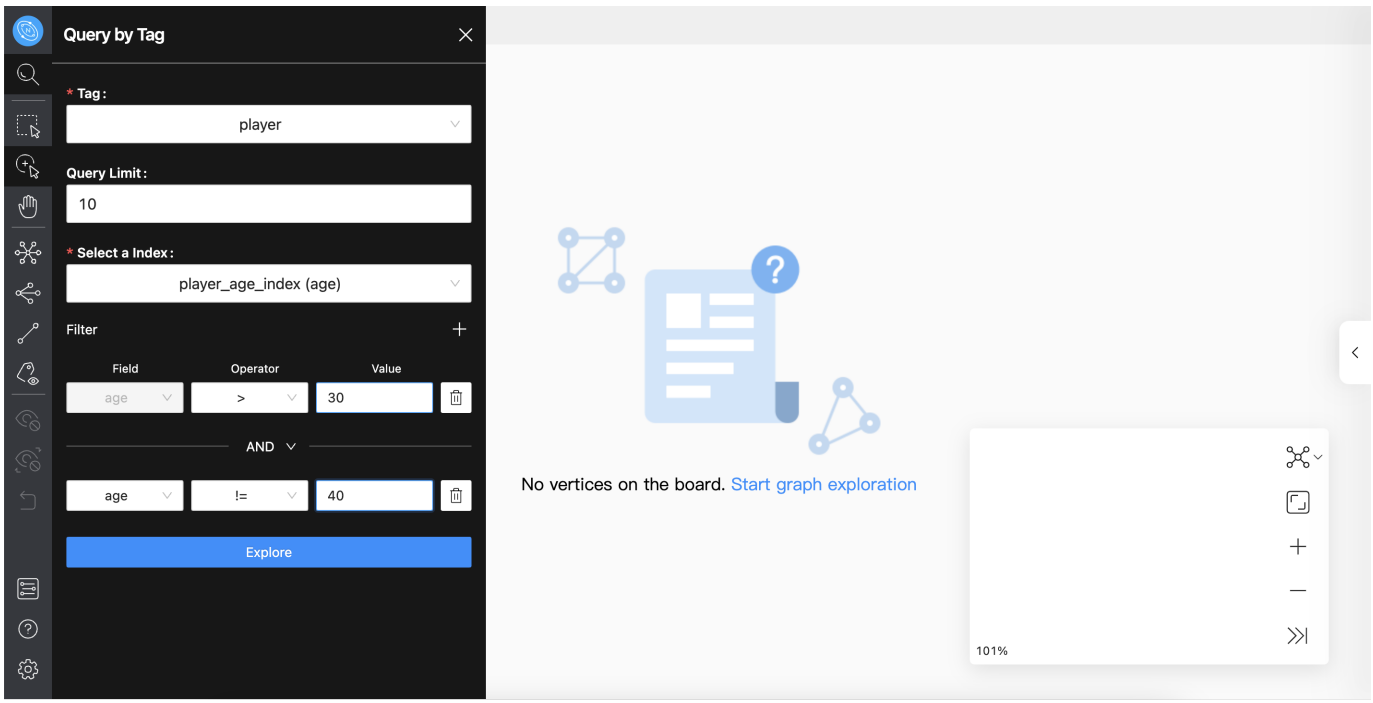
Query by VID

You can query data by entering the VIDs or other data for VID generation, and a row only supports one data. It also supports random import of data and file import of data. After confirming the addition, the data will be displayed in the canvas. An example is given below:



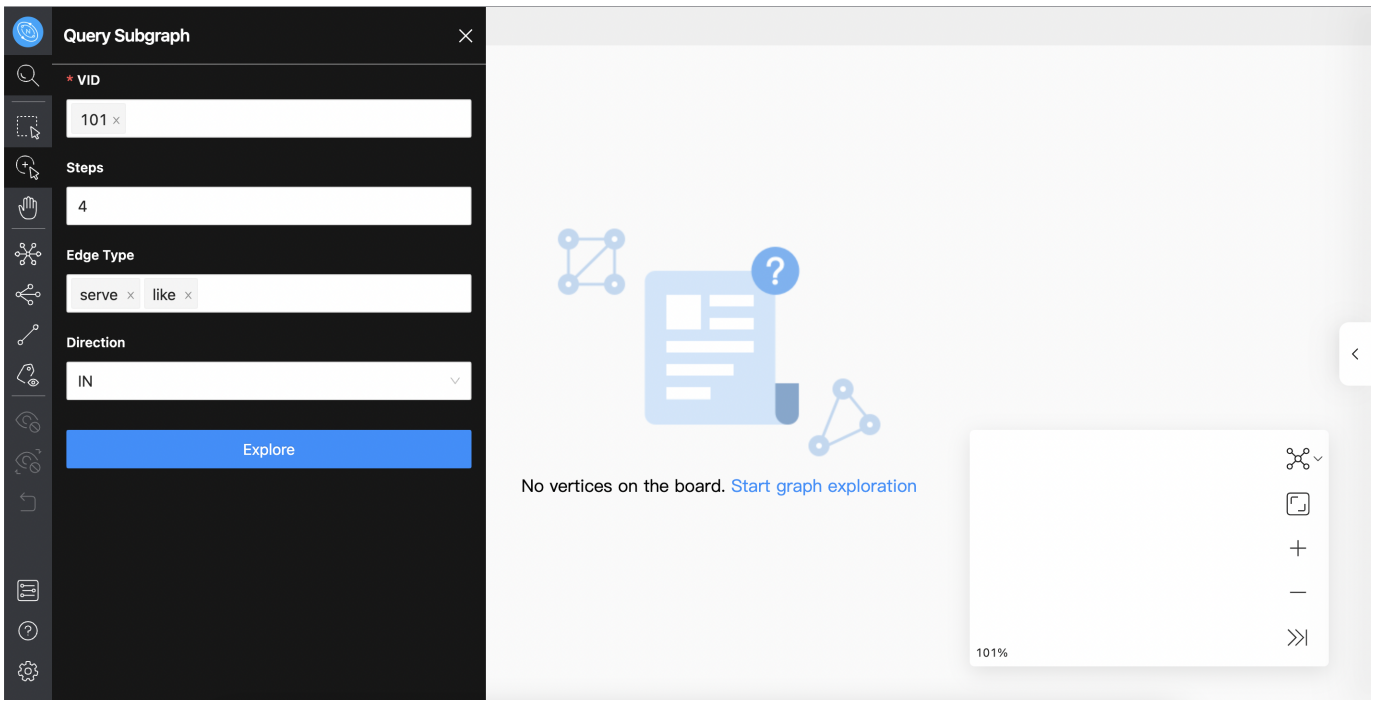
Query by Tag

The required values are Tag and Index. You can limit the number of output results and filter the results. The following query 10 players whose age is greater than 30 years old and not equal to 40 years old, examples are as follows:



Query by Subgraph

The required value is VID. You can view the subgraph of one or more vertexes, and you can specify the number of steps, edge types, and the direction of inflow and outflow of the subgraph. The following is an example of an incoming edge with a VID value of 101, the number of steps of 4, and edge types of `server` and `like`:



Last update: September 2, 2021

13.3.3 Graph exploration and expansion

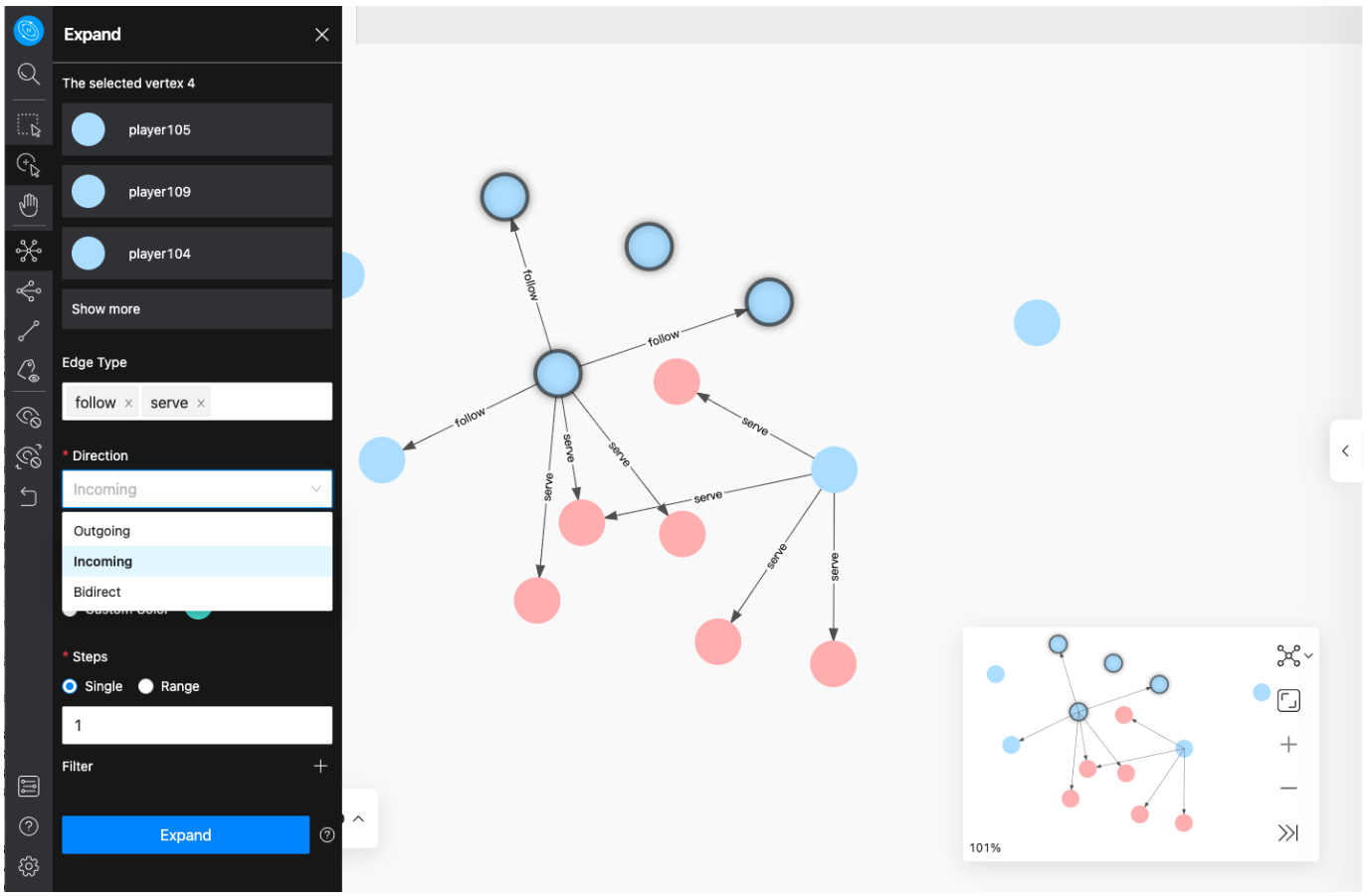
Graph exploration and expansion is divided into the following four parts:

- Expand
- Common neighbor
- Search path
- Inspect property

Expand

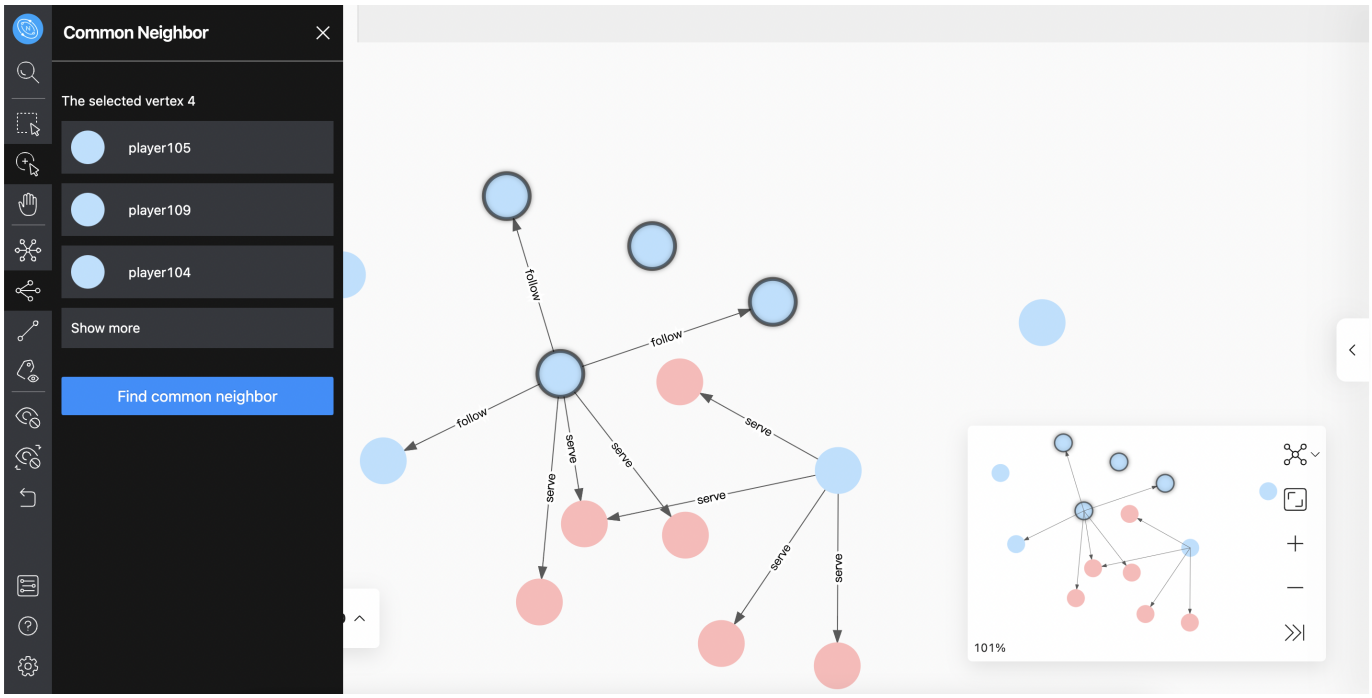
In the sidebar, click the  icon to open the **Expand** window. You can double-click a vertex to expand directly. You can also select multiple vertexes in the canvas, modify the edge type in the operation bar, select the inflow and outflow of the edge, modify the color of the vertex, specify the number of expansion steps and custom filter conditions.

> **Note**
>
> After the configuration in the panel is modified, the current configuration will be saved, and the current configuration will be expanded when double-clicking or right-clicking to quickly expand.



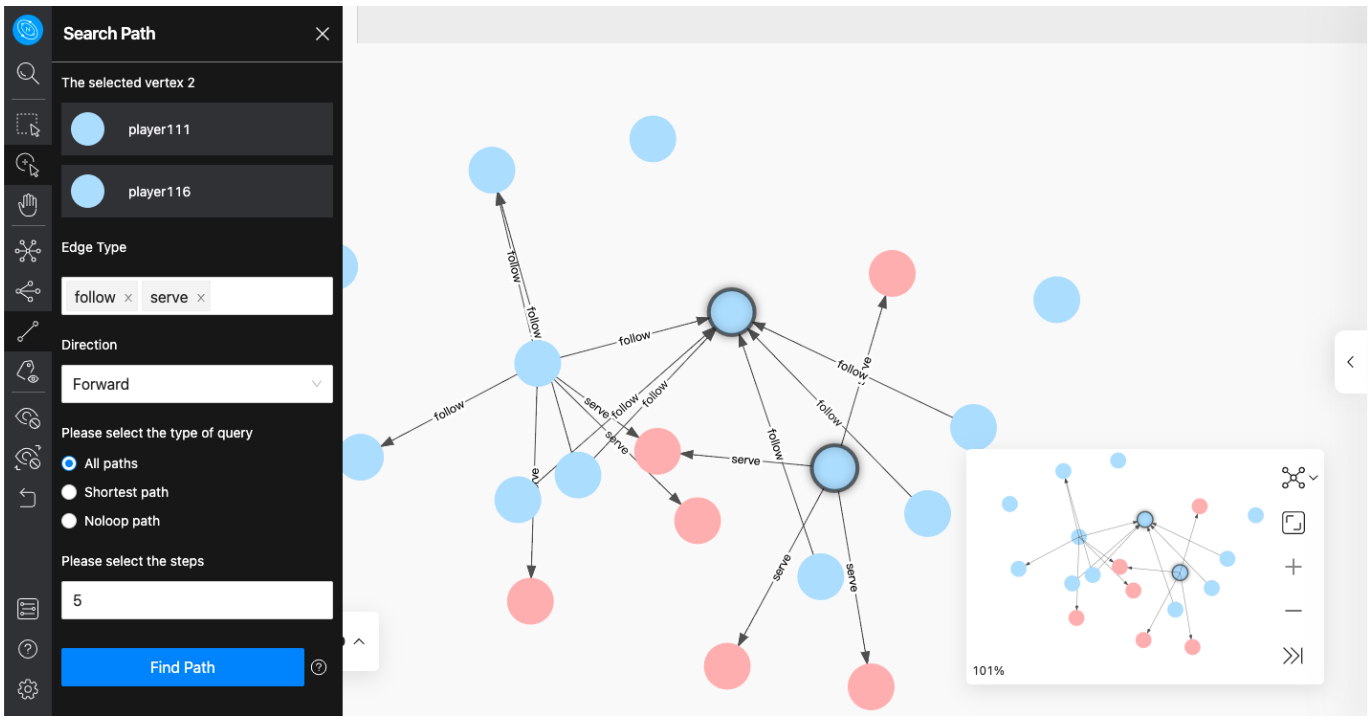
Common neighbor

In the sidebar, click the  icon to open the **Common Neighbor** window. You can select two or more vertexes on the canvas and query their common neighbors. When the selected vertexes have no common neighbor, the default returns **There is no data**.



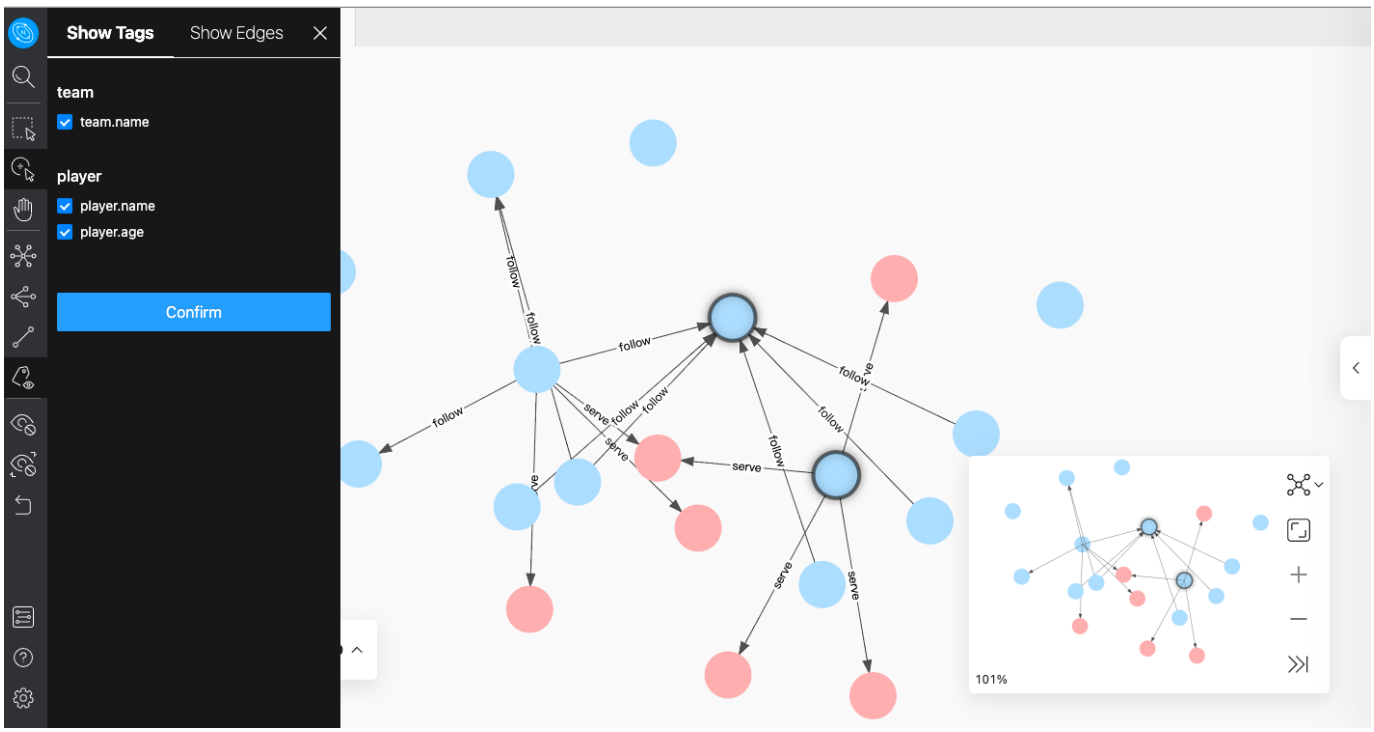
Search path

In the sidebar, click the findpath icon to open the **Search path** window. You can select two vertexes in the canvas. By default, the first vertex selected is the starting point, and the second vertex is the ending point. You can customize the type and direction of the edge, specify the number of expansion steps, and choose to query the following three paths: `All path`, `Shortest path` and `Noloop path`.



Inspect property

In the sidebar, click the  icon to open the **Inspect property** window. You can choose to show or hide the property of vertexes or edges in the canvas. After clicking confirm, the property will be displayed on the canvas only when the zoom ratio is greater than 100%, and automatically hidden when the zoom ratio is less than 100%.



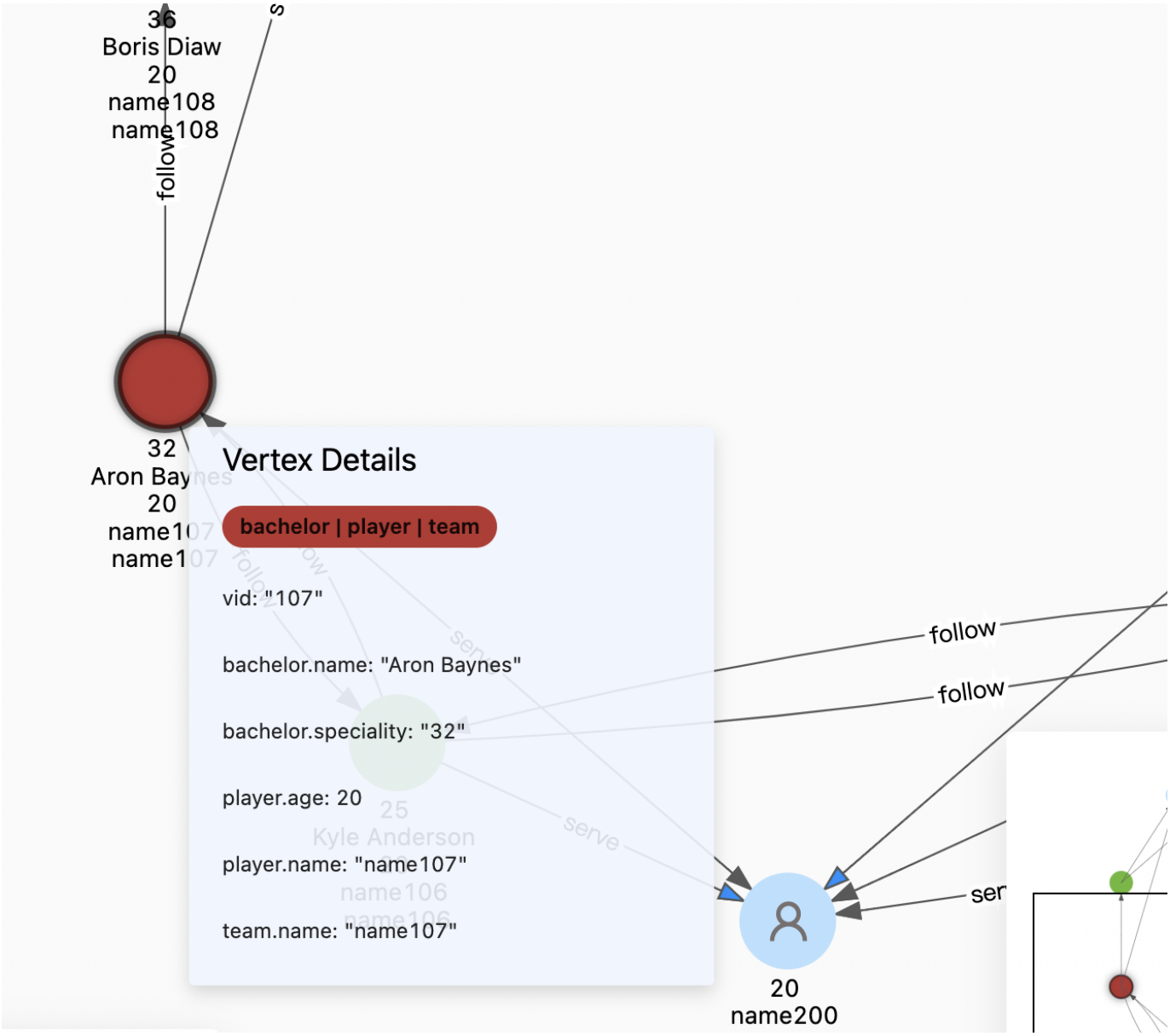
Last update: September 2, 2021

13.3.4 Canvas operation

This topic describes operation in canvas.

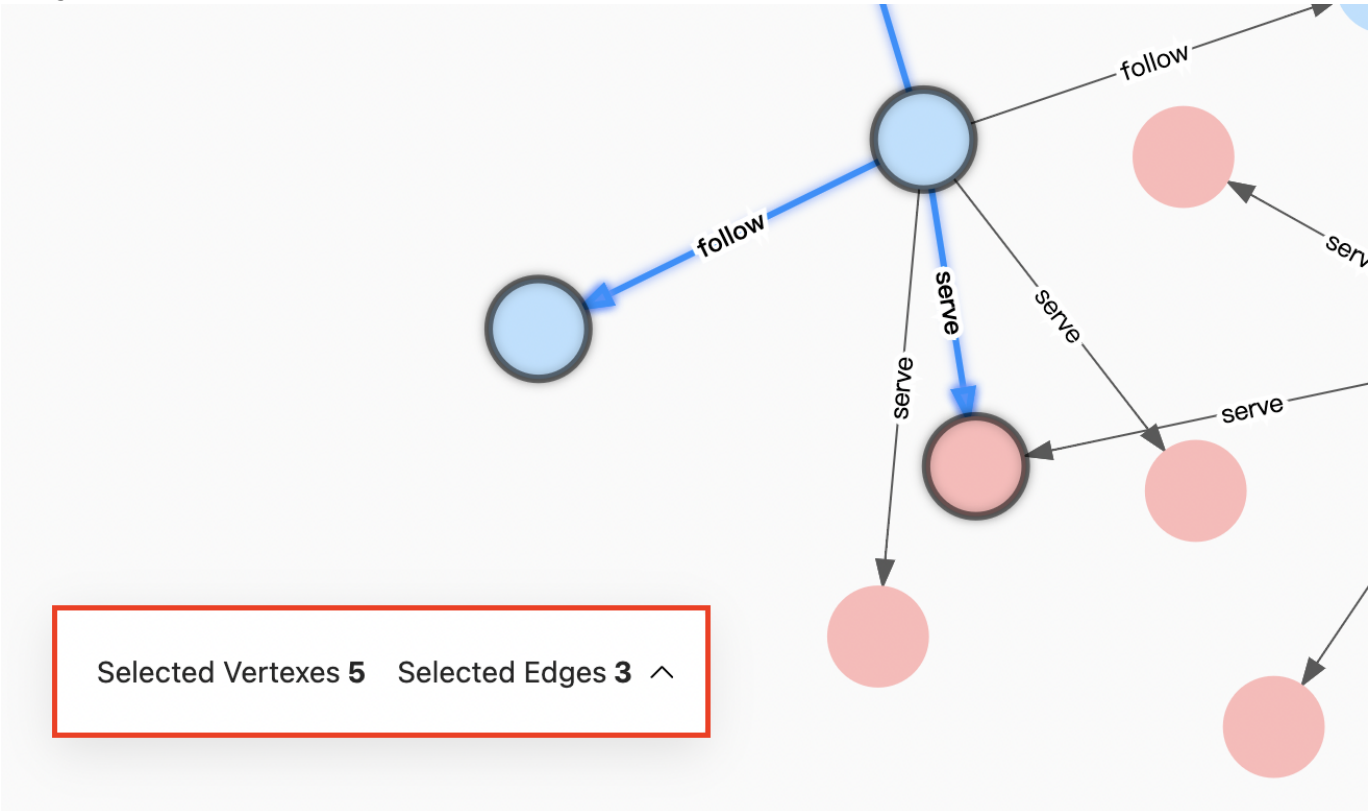
Display vertexes and edges

Move the mouse to the vertex or edge to view in detail. The following shows the detailed information of the vertex with `VID 107`:



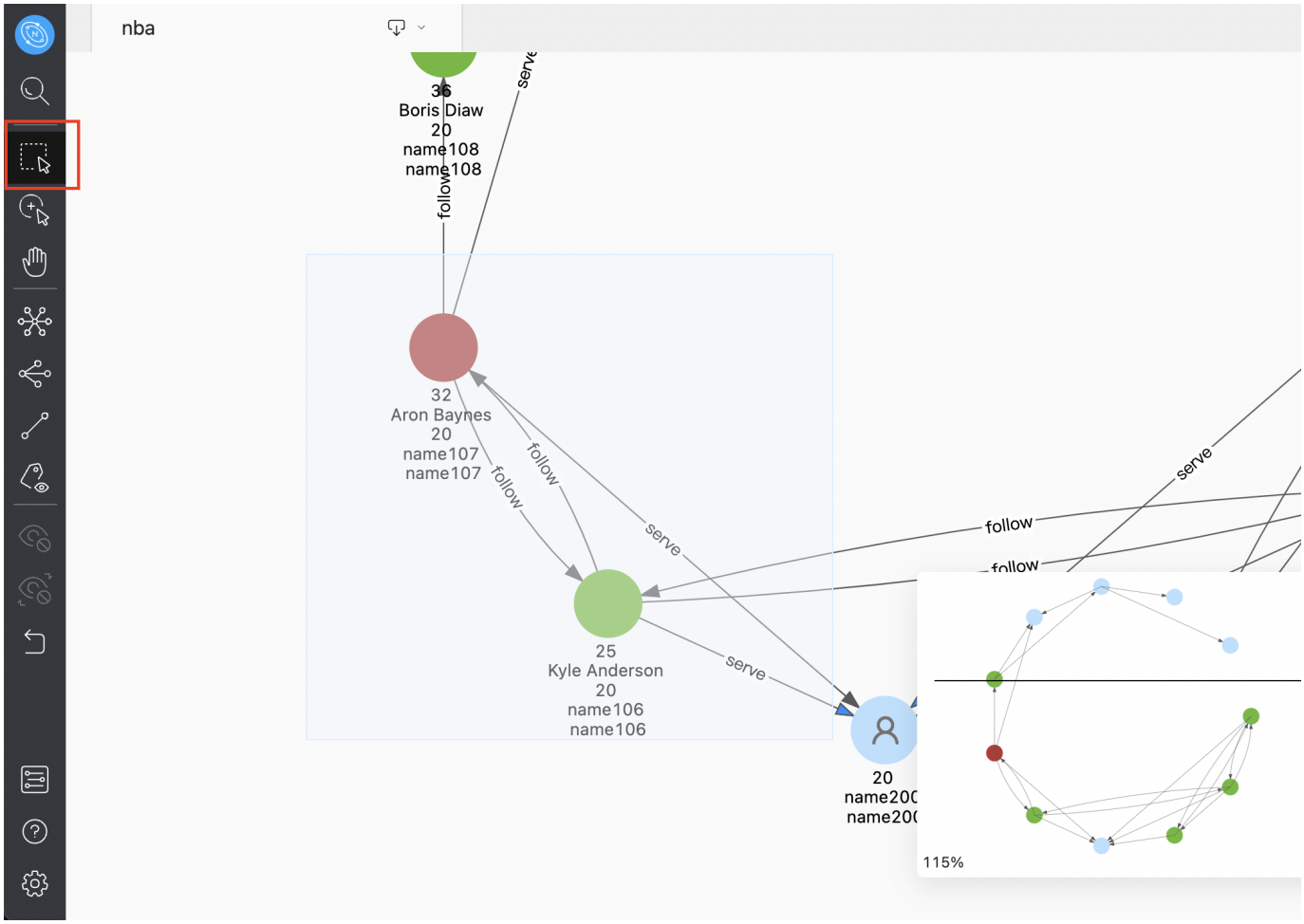
Batch selection

Explorer supports batch selection and views the data of multiple vertexes and edges. The detailed data can be opened and viewed in the vertexes and edges overview at the lower-left corner of the canvas. It also supports exporting CSV files of selected vertexes or edges.



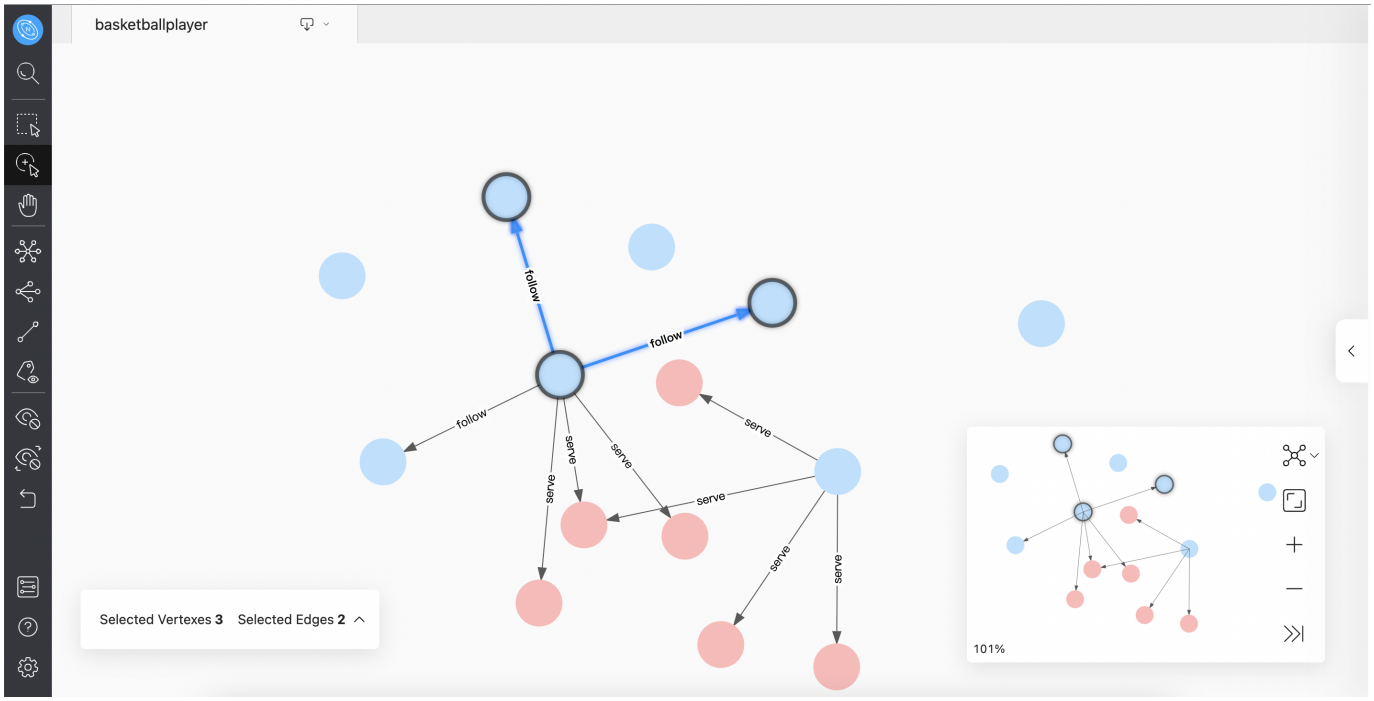
FRAME SELECTION

After clicking the  icon, hold down the left button to drag and select multiple vertexes and edges. Examples are as follows:



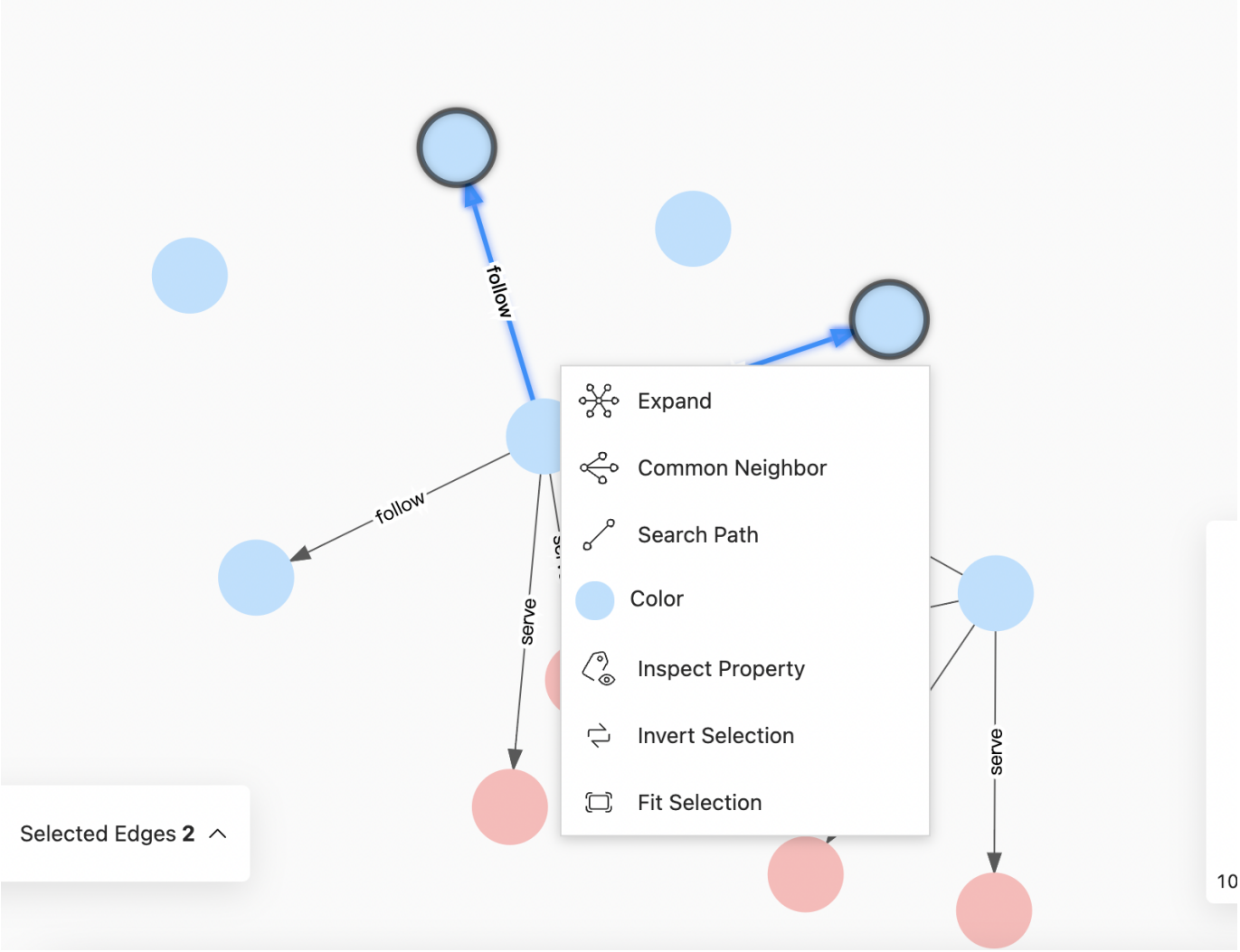
CLICK TO SELECT MULTIPLE VERTEXES AND EDGES

Click the  icon or hold down Shift, click and select multiple vertexes and edges with the mouse, and click the blank space to cancel the selection. Examples are as follows:



Quick operation

You can select one or more vertexes and edges, and right-click in the blank area to expand the vertexes, search the path between two vertexes, and show or hide their property on the canvas. The number of vertexes and edges you choose will affect the operations that can be performed. For more information, see Graph Exploration Expansion.

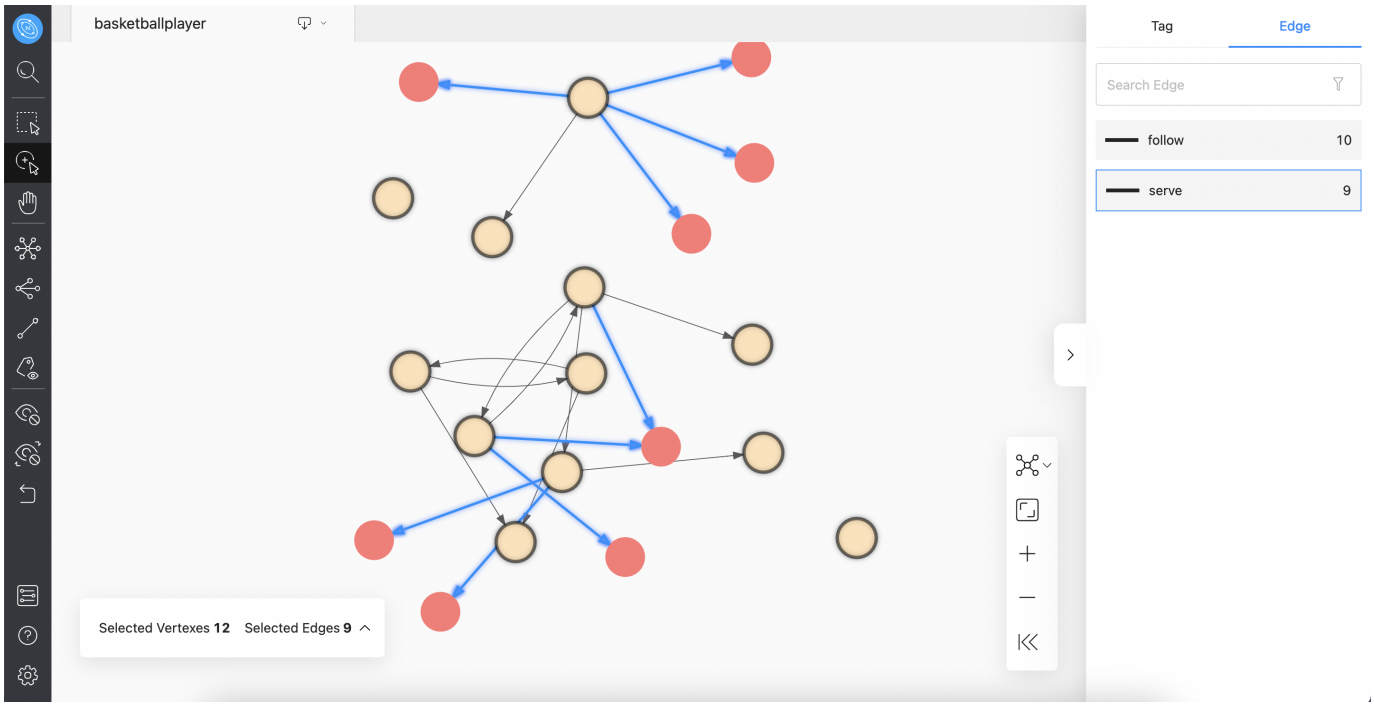


Click **Fit Selection** to move the selected data to the center of the canvas for users to view.

Last update: September 2, 2021

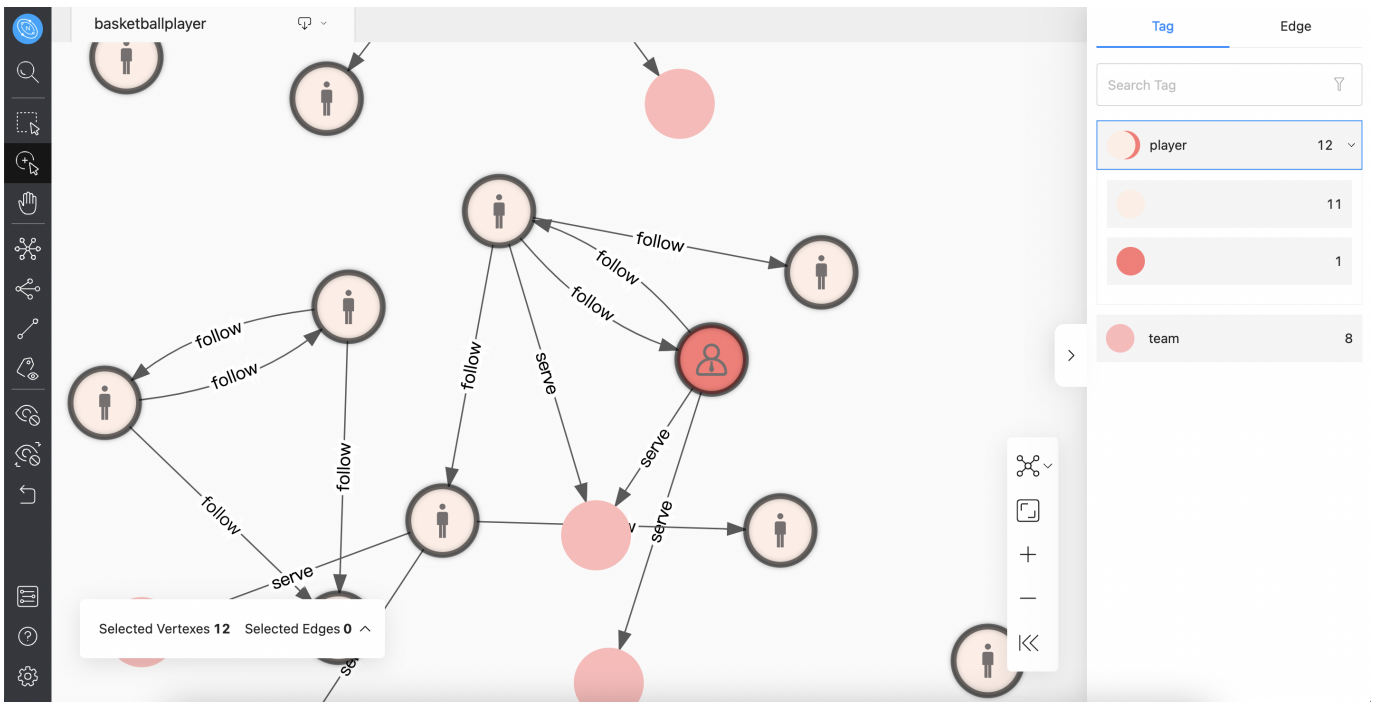
13.3.5 Relationship list

You can select vertexes and edges in the relationship list. Select 12 points where Tag is `player`, and select 9 edges where Edge is `serve`. An example is as follows:



At the same time, you can modify the color and icon of the Tag to make the key nodes more prominent.

By default, VID with identical tags have the same color, and it is also allowed to manually modify the color of a vertex or a group of vertexes with identical tags. For example, if the vertex label is `player`, modify the color of one of the vertexes, and you can click to view it in the relationship list. The example is as follows:



Last update: September 2, 2021

13.3.6 Shortcuts

This document lists the shortcuts supported in Explorer.

Operation	Description
Shift + 'Enter'	Expand
Shift + '-'	Zoom out
Shift + '+'	Zoom in
Shift + 'l'	Display
Shift + 'z'	Undo
Selected + Shift + 'del'	Delete

Last update: September 2, 2021

14. Nebula Importer

14.1 Nebula Importer

Nebula Importer (Importer) is a standalone import tool for CSV files with Nebula Graph. Importer can read the local CSV file and then import the data into the Nebula Graph database.

14.1.1 Scenario

Importer is used to import the contents of a local CSV file into the Nebula Graph.

14.1.2 Advantage

- Lightweight and fast: no complex environment can be used, fast data import.
- Flexible filtering: You can flexibly filter CSV data through configuration files.

14.1.3 Prerequisites

Before using Nebula Importer, make sure:

- Nebula Graph service has been deployed. There are currently three deployment modes:
  - Deploy Nebula Graph with Docker Compose
  - Install Nebula Graph with RPM or DEB package
  - Install Nebula Graph by compiling the source code
- Schema is created in Nebula Graph, including space, Tag and Edge type, or set by parameter `clientSettings.postStart.commands`.
- Golang environment has been deployed on the machine running the Importer. For details, see Build Go environment.

14.1.4 Steps

Configure the YAML file and prepare the CSV file to be imported to use the tool to batch write data to Nebula Graph.

Source code compile and run

1. Clone repository.

```
$ git clone -b 2.5.0 https://github.com/vesoft-inc/nebula-importer.git
```

> **Note**
>
> Use the correct branch. Nebula Graph 1.x and 2.x have different RPC protocols, so:
>
> - The Nebula Importer V1 branch can only connect to Nebula Graph 1.x.
> - The Nebula Importer Master branch and v2 branch can connect to Nebula Graph 2.x.

2. Access the directory `nebula-importer`.

```
$ cd nebula-importer
```

3. Compile the source code.

```
$ make build
```

4. Start the service.

```
$ ./nebula-importer --config <yaml_config_file_path>
```

> **Note**
>
> For details about the YAML configuration file, see configuration file description at the end of topic.

NO NETWORK COMPILATION MODE

If the server cannot be connected to the Internet, it is recommended to upload the source code and various dependency packages to the corresponding server for compilation on the machine that can be connected to the Internet. The operation steps are as follows:

1. Clone repository.

```
$ git clone -b 2.5.0 https://github.com/vesoft-inc/nebula-importer.git
```

2. Use the following command to download and package the dependent source code.

```
$ cd nebula-importer
$ go mod vendor
$ cd .. && tar -zcvf nebula-importer.tar.gz nebula-importer
```

3. Upload the compressed package to a server that cannot be connected to the Internet.
4. Unzip and compile.

```
$ tar -zxvf nebula-importer.tar.gz
$ cd nebula-importer
$ go build -mod vendor cmd/importer.go
```

Run in Docker mode

Instead of installing the Go locale locally, you can use Docker to pull the image of the Nebula Importer and mount the local configuration file and CSV data file into the container. The command is as follows:

```
$ docker run --rm -ti \
    --network=host \
    -v <config_file>:<config_file> \
    -v <csv_data_dir>:<csv_data_dir> \
    vesoft/nebula-importer:<version>
    --config <config_file>
```

- `<config_file>`: The absolute path to the local YAML configuration file.
- `<csv_data_dir>`: The absolute path to the local CSV data file.
- `<version>`: Nebula Graph 2.x Please fill in 'v2'.

> **Note**
>
> A relative path is recommended. If you use a local absolute path, check that the path maps to the path in the Docker.

14.1.5 Configuration File Description

Nebula Importer uses configuration(`nebula-importer/examples/v2/example.yaml`) files to describe information about the files to be imported, the Nebula Graph server, and more. You can refer to the example configuration file: Configuration without Header/ Configuration with Header. This section describes the fields in the configuration file by category.

Basic configuration

The example configuration is as follows:

```
version: v2
description: example
removeTempFiles: false
```

Parameter	Default value	Required	Description
`version`	v2	Yes	Target version of Nebula Graph.
`description`	example	No	Description of the configuration file.
`removeTempFiles`	false	No	Whether to delete temporarily generated logs and error data files.

Client configuration

The client configuration stores the configurations associated with Nebula Graph.

The example configuration is as follows:

```
clientSettings:
  retry: 3
  concurrency: 10
  channelBufferSize: 128
  space: test
  connection:
    user: user
    password: password
    address: 192.168.*.13:9669,192.168.*.14:9669
  postStart:
    commands: |
      UPDATE CONFIGS storage:wal_ttl=3600;
      UPDATE CONFIGS storage:rocksdb_column_family_options = { disable_auto_compactions = true };
    afterPeriod: 8s
  preStop:
    commands: |
      UPDATE CONFIGS storage:wal_ttl=86400;
      UPDATE CONFIGS storage:rocksdb_column_family_options = { disable_auto_compactions = false };
```

Parameter	Default value	Required	Description
`clientSettings.retry`	3	No	Retry times of nGQL statement execution failures.
`clientSettings.concurrency`	10	No	Number of Nebula Graph client concurrency.
`clientSettings.channelBufferSize`	128	No	Cache queue size per Nebula Graph client.
`clientSettings.space`	-	Yes	Specifies the Nebula Graph space to import the data into. Do not import multiple spaces at the same time to avoid performance impact.
`clientSettings.connection.user`	-	Yes	Nebula Graph user name.
`clientSettings.connection.password`	-	Yes	The password for the Nebula Graph user name.
`clientSettings.connection.address`	-	Yes	Addresses and ports for all Graph services.
`clientSettings.postStart.commands`	-	No	Configure some of the operations to perform after connecting to the Nebula Graph server, and before inserting data.
`clientSettings.postStart.afterPeriod`	-	No	The interval, between executing the above `commands` and executing the insert data command, such as `8s`.
`clientSettings.preStop.commands`	-	No	Configure some of the actions you performed before disconnecting from the Nebula Graph server.

File configuration

File configuration Stores the configuration of data files and logs, and details about the Schema.

FILE AND LOG CONFIGURATION

The example configuration is as follows:

```
logPath: ./err/test.log
files:
  - path: ./student_without_header.csv
    failDataPath: ./err/studenterr.csv
    batchSize: 128
    limit: 10
    inOrder: false
    type: csv
    csv:
      withHeader: false
      withLabel: false
      delimiter: ","
```

Parameter	Default value	Required	Description
`logPath`	-	No	Path for exporting log information, such as errors during import.
`files.path`	-	Yes	Path for storing data files. If a relative path is used, the path is merged with the current configuration file directory. You can use an asterisk (*) for fuzzy matching to import multiple files with similar names, but the files need to be the same structure.
`files.failDataPath`	-	Yes	Insert the failed data file storage path, so that data can be written later.
`files.batchSize`	128	No	The number of statements inserting data in a batch.
`files.limit`	-	No	Limit on the number of rows of read data.
`files.inOrder`	-	No	Whether to insert rows in the file in order. If the value is set to `false`, the import rate decreases due to data skew.
`files.type`	-	Yes	The file type.
`files.csv.withHeader`	`false`	Yes	Whether there is a header.
`files.csv.withLabel`	`false`	Yes	Whether there is a label.
`files.csv.delimiter`	`","`	Yes	Specifies the delimiter for the CSV file. A string delimiter that supports only one character.

SCHEMA CONFIGURATION

Schema configuration describes the Meta information of the current data file. Schema types are vertex and edge. Multiple vertexes or edges can be configured at the same time.

- vertex configuration

The example configuration is as follows:

```
schema:
  type: vertex
  vertex:
    vid:
      type: string
      index: 0
    tags:
      - name: student
        props:
          - name: name
            type: string
            index: 1
          - name: age
            type: int
            index: 2
          - name: gender
```

```
          type: string
          index: 3
```

Parameter	Default value	Required	Description
`files.schema.type`	-	Yes	Schema type. Possible values are `vertex` and `edge`.
`files.schema.vertex.vid.type`	-	No	The data type of the vertex ID. Possible values are `int` and `string`.
`files.schema.vertex.vid.index`	-	No	The vertex ID corresponds to the column number in the CSV file.
`files.schema.vertex.tags.name`	-	Yes	Tag name.
`files.schema.vertex.tags.props.name`	-	Yes	Tag property name, which must match the Tag property in the Nebula Graph.
`files.schema.vertex.tags.props.type`	-	Yes	Property data type, supporting `bool`, `int`, `float`, `double`, `timestamp` and `string`.
`files.schema.vertex.tags.props.index`	-	No	Property corresponds to the sequence number of the column in the CSV file.

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.

- edge configuration

The example configuration is as follows:

```
schema:
  type: edge
  edge:
    name: follow
    withRanking: true
    srcVID:
      type: string
      index: 0
    dstVID:
      type: string
      index: 1
    rank:
      index: 2
    props:
      - name: degree
```

```
      type: double
      index: 3
```

Parameter	Default value	Required	Description
`files.schema.type`	-	Yes	Schema type. Possible values are `vertex` and `edge`.
`files.schema.edge.name`	-	Yes	Edge type name.
`files.schema.edge.srcVID.type`	-	No	[illegible]ID[illegible].
`files.schema.edge.srcVID.index`	-	No	The data type of the starting vertex ID of the edge.
`files.schema.edge.dstVID.type`	-	No	The data type of the destination vertex ID of the edge.
`files.schema.edge.dstVID.index`	-	No	The destination vertex ID of the edge corresponds to the column number in the CSV file.
`files.schema.edge.rank.index`	-	No	The rank value of the edge corresponds to the column number in the CSV file.
`files.schema.edge.props.name`	-	Yes	The Edge Type property name must match the Edge Type property in the Nebula Graph.
`files.schema.edge.props.type`	-	Yes	Property data type, supporting `bool`, `int`, `float`, `double`, `timestamp` and `string`.
`files.schema.edge.props.index`	-	No	Property corresponds to the sequence number of the column in the CSV file.

14.1.6 About the CSV file header

According to whether the CSV file has a header or not, the Importer needs to make different Settings on the configuration file. For relevant examples and explanations, please refer to:

- Configuration without Header
- Configuration with Header

Last update: September 16, 2021

14.2 Configuration with Header

For a CSV file with header, you need to set `withHeader` to `true` in the configuration file, indicating that the first behavior in the CSV file is the header. The header content has special meanings.

> **Caution**
>
> If the CSV file contains headers, the Importer will parse the Schema of each row of data according to the headers and ignore the vertex or edge settings in the YAML file.

14.2.1 Sample files

The following is an example of a CSV file with header:

- sample of vertex

  Example data for `student_with_header.csv`:

  ```
  :VID(string),student.name:string,student.age:int,student.gender:string
  student100,Monica,16,female
  student101,Mike,18,male
  student102,Jane,17,female
  ```

  The first column is the vertex ID, followed by the properties `name`, `age`, and `gender`.

- sample of edge

  Example data for `follow_with_header.csv`:

  ```
  :SRC_VID(string),:DST_VID(string),:RANK,follow.degree:double
  student100,student101,0,92.5
  student101,student100,1,85.6
  student101,student102,2,93.2
  student100,student102,1,96.2
  ```

  The first two columns are the start vertex ID and destination vertex ID, respectively. The third column is rank, and the fourth column is property `degree`.

14.2.2 Header format description

The header defines the start vertex, the destination vertex, the rank, and some special functions by keywords as follows:

- `:VID` (mandatory): Vertex ID. Need to use `:VID(type)` form to set data type, for example `:VID(string)` or `:VID(int)`.
- `:SRC_VID` (mandatory): The start vertex ID of the edge. The data type needs to be set in the form `:SRC_VID(type)`.
- `:DST_VID` (mandatory): The destination vertex ID of the edge. The data type needs to be set in the form `:DST_VID(type)`.
- `:RANK` (optional): The rank value of the edge.
- `:IGNORE` (optional): Ignore this column when inserting data.
- `:LABEL` (optional): Insert (+) or delete (-) the row. Must be column 1. For example:

  ```
  :LABEL,
  +,
  -,
  ```

> **Note**
>
> All columns except the `:LABEL` column can be sorted in any order, so for larger CSV files, the user has the flexibility to set the header to select the desired column.

For Tag or Edge type properties, the format is `<tag_name/edge_name>.<prop_name>:<prop_type>`, described as follows:

- `<tag_name/edge_name>`: Tag or Edge type name.
- `<prop_name>`: property name.
- `<prop_type>`: property type. Support `bool`, `int`, `float`, `double`, `timestamp` and `string`, default `string`.

Such as `student.name:string`, `follow.degree:double`.

14.2.3 Sample configuration

```
# Connected to the Nebula Graph version, set to v2 when connected to 2.x.
version: v2

description: example

# Whether to delete temporarily generated logs and error data files.
removeTempFiles: false

clientSettings:

  # Retry times of nGQL statement execution failures.
  retry: 3

  # Number of Nebula Graph client concurrency.
  concurrency: 10

  # Cache queue size per Nebula Graph client.
  channelBufferSize: 128

  # Specifies the Nebula Graph space to import the data into.
  space: student

  # Connection information.
  connection:
    user: root
    password: nebula
    address: 192.168.*.13:9669

  postStart:
    # Configure some of the operations to perform after connecting to the Nebula Graph server, and before inserting data.
    commands: |
      DROP SPACE IF EXISTS student;
      CREATE SPACE IF NOT EXISTS student(partition_num=5, replica_factor=1, vid_type=FIXED_STRING(20));
      USE student;
      CREATE TAG student(name string, age int,gender string);
      CREATE EDGE follow(degree int);

    # The interval between the execution of the above command and the execution of the insert data command.
    afterPeriod: 15s

  preStop:
    # Configure some of the actions you performed before disconnecting from the Nebula Graph server.
    commands: |

# Path of the error log file.
logPath: ./err/test.log

# CSV file Settings.
files:

    # Path for storing data files. If a relative path is used, the path is merged with the current configuration file directory. The first data file in this example
is vertex data.
  - path: ./student_with_header.csv

    # Insert the failed data file storage path, so that data can be written later.
    failDataPath: ./err/studenterr.csv

    # The number of statements inserting data in a batch.
    batchSize: 10

    # Limit on the number of rows of read data.
    limit: 10

    # Whether to insert rows in the file in order. If the value is set to false, the import rate decreases due to data skew.
    inOrder: true

    # File type. Currently, only CSV files are supported.
    type: csv

    csv:
      # Whether there is a header.
      withHeader: true

      # Whether there is a LABEL.
      withLabel: false
```

```
    # Specifies the delimiter for the CSV file. A string delimiter that supports only one character.
    delimiter: ","

  schema:
    # Schema type. Possible values are vertex and edge.
    type: vertex

  # The second data file in this example is edge data.
- path: ./follow_with_header.csv
  failDataPath: ./err/followerr.csv
  batchSize: 10
  limit: 10
  inOrder: true
  type: csv
  csv:
    withHeader: true
    withLabel: false
  schema:
    # The type of Schema is edge.
    type: edge
    edge:
      # Edge type name.
      name: follow

      # Whether to include rank.
      withRanking: true
```

Note

The data type of the vertex ID must be the same as the data type of the statement in `clientSettings.postStart.commands` that creates the graph space.

Last update: September 2, 2021

14.3 Configuration without Header

For CSV files without header, you need to set `withHeader` to `false` in the configuration file, indicating that the CSV file contains only data (excluding the header of the first row). You may also need to set the data type and corresponding columns.

14.3.1 Sample files

The following is an example of a CSV file without header:

- sample of vertex

  Example data for `student_without_header.csv`:

  ```
  student100,Monica,16,female
  student101,Mike,18,male
  student102,Jane,17,female
  ```

  The first column is the vertex ID, followed by the properties `name`, `age`, and `gender`.

- sample of edge

  Example data for `follow_without_header.csv`:

  ```
  student100,student101,0,92.5
  student101,student100,1,85.6
  student101,student102,2,93.2
  student100,student102,1,96.2
  ```

  The first two columns are the start vertex ID and destination vertex ID, respectively. The third column is rank, and the fourth column is property `degree`.

14.3.2 Sample configuration

```
# Connected to the Nebula Graph version, set to v2 when connected to 2.x.
version: v2

description: example

# Whether to delete temporarily generated logs and error data files.
removeTempFiles: false

clientSettings:

  # Retry times of nGQL statement execution failures.
  retry: 3

  # Number of Nebula Graph client concurrency.
  concurrency: 10

  # Cache queue size per Nebula Graph client.
  channelBufferSize: 128

  # Specifies the Nebula Graph space to import the data into.
  space: student

  # Connection information.
  connection:
    user: root
    password: nebula
    address: 192.168.*.13:9669

  postStart:
    # Configure some of the operations to perform after connecting to the Nebula Graph server, and before inserting data.
    commands: |
      DROP SPACE IF EXISTS student;
      CREATE SPACE IF NOT EXISTS student(partition_num=5, replica_factor=1, vid_type=FIXED_STRING(20));
      USE student;
      CREATE TAG student(name string, age int,gender string);
      CREATE EDGE follow(degree int);

    # The interval between the execution of the above command and the execution of the insert data command.
    afterPeriod: 15s

  preStop:
    # Configure some of the actions you performed before disconnecting from the Nebula Graph server.
    commands: |

# Path of the error log file.
```

```
logPath: ./err/test.log

# CSV file Settings.
files:

    # Path for storing data files. If a relative path is used, the path is merged with the current configuration file directory. The first data file in this example
is vertex data.
  - path: ./student_without_header.csv

    # Insert the failed data file storage path, so that data can be written later.
    failDataPath: ./err/studenterr.csv

    # The number of statements inserting data in a batch.
    batchSize: 10

    # Limit on the number of rows of read data.
    limit: 10

    # Whether to insert rows in the file in order. If the value is set to false, the import rate decreases due to data skew.
    inOrder: true

    # File type. Currently, only CSV files are supported.
    type: csv

    csv:
      # Whether there is a header.
      withHeader: false

      # Whether there is a LABEL.
      withLabel: false

      # Specifies the delimiter for the CSV file. A string delimiter that supports only one character.
      delimiter: ","

    schema:
      # Schema type. Possible values are vertex and edge.
      type: vertex

      vertex:

        # Vertex ID Settings.
        vid:
           # The vertex ID corresponds to the column number in the CSV file. Columns in the CSV file are numbered from 0.
           index: 0

           # The data type of the vertex ID. The optional values are int and string, corresponding to INT64 and FIXED_STRING in the Nebula Graph, respectively.
           type: string

        # Tag Settings.
            # Tag name.
          - name: student

            # property Settings in the Tag.
            props:
                # property name.
              - name: name

                # Property data type.
                type: string

                # Property corresponds to the sequence number of the column in the CSV file.
                index: 1

              - name: age
                type: int
                index: 2
              - name: gender
                type: string
                index: 3

    # The second data file in this example is edge data.
  - path: ./follow_without_header.csv
    failDataPath: ./err/followerr.csv
    batchSize: 10
    limit: 10
    inOrder: true
    type: csv
    csv:
      withHeader: false
      withLabel: false
    schema:
      # The type of Schema is edge.
      type: edge
      edge:
        # Edge type name.
        name: follow

        # Whether to include rank.
        withRanking: true

        # Start vertex ID setting.
        srcVID:
           # Data type.
           type: string
```

```
            # The start vertex ID corresponds to the sequence number of a column in the CSV file.
            index: 0

        # Destination vertex ID.
        dstVID:
            type: string
            index: 1

        # rank setting.
        rank:
            # Rank Indicates the rank number of a column in the CSV file. If index is not set, be sure to set the rank value in the third column. Subsequent columns
set each property in turn.
            index: 2

        # Edge Type property Settings.
        props:
             # property name.
            - name: degree

              # Data type.
              type: double

              # Property corresponds to the sequence number of the column in the CSV file.
              index: 3
```

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.
- The data type of the vertex ID must be the same as the data type of the statement in `clientSettings.postStart.commands` that creates the graph space.
- If the index field is not specified, the CSV file must comply with the following rules:
  - In the vertex data file, the first column must be the vertex ID, followed by the properties, and must correspond to the order in the configuration file.
  - In the side data file, the first column must be the start vertex ID, the second column must be the destination vertex ID, if `withRanking` is `true`, the third column must be the rank value, and the following columns must be properties, and must correspond to the order in the configuration file.

Last update: September 1, 2021

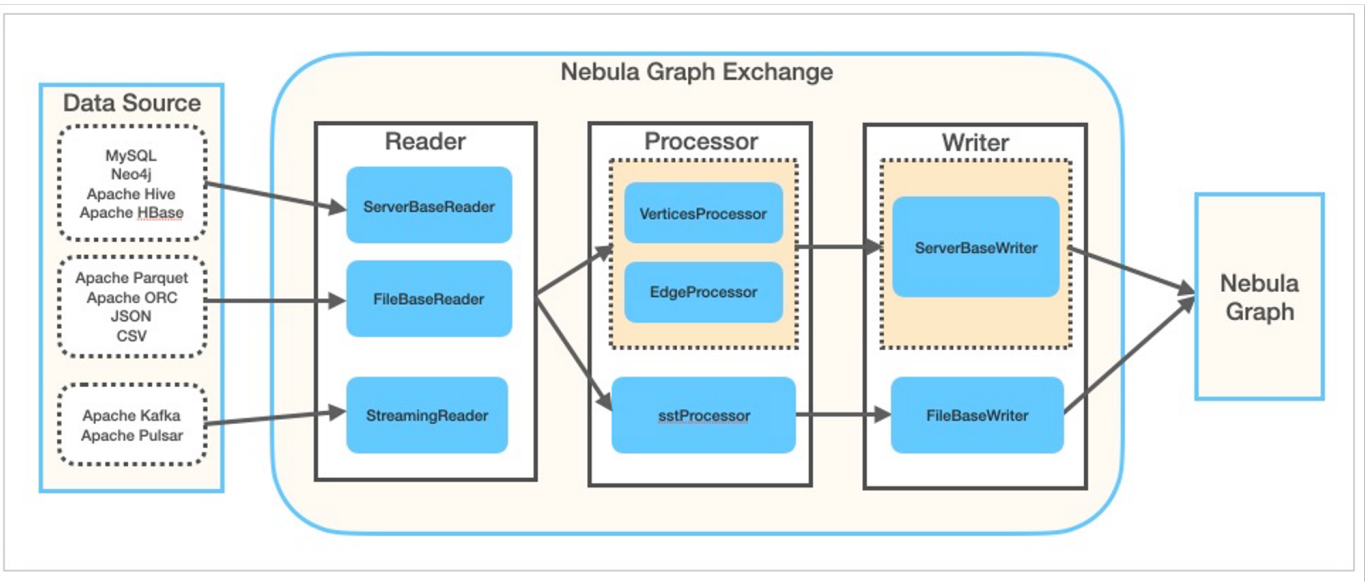
15. Nebula Exchange

15.1 Introduction

15.1.1 What is Nebula Exchange

Nebula Exchange (Exchange) is an Apache Spark™ application for bulk migration of cluster data to Nebula Graph 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 Nebula Graph at once. The following figure illustrates the process by which Exchange completes the data conversion and migration.



Scenario

Exchange is applicable to the following scenarios:

- Need to streaming data from Kafka, Pulsar platform, such as log files, online data, game players activities, social networking information and financial transactions within the hall or geospatial services, as well as from within the data center of the connected device or instrument into properties such as telemetry data diagram of data vertex or edge, and import the Nebula Graph 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 Nebula Graph database.
- A large volume of data needs to be generated into SST files that Nebula Graph can recognize and then imported into the Nebula Graph database.

Advantage

Exchange has the following advantages:

- Adaptable: Support for importing data into the Nebula Graph database in a variety of formats or from a variety of sources, making it easy to migrate data.
- SST import: Converts data from different sources into SST files for data import.
- Resumable data import: Resumable data import saves time and improves data import efficiency.

> **Note**
>
> Breakpoint continuation is currently supported only when Neo4j data is migrated.

- Asynchronous operation: An insert statement is generated in the source data, sent to the Graph service, and then the insert operation is performed.
- Flexibility: support to import multiple Tags and Edge types at the same time. Different tag and Edge type can be different data sources or formats.
- Statistics: Use the accumulator in Apache Spark™ 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 object-oriented style, which is easy to understand and operate.

Data source

Exchange 2.5.0 supports converting data from the following formats or sources into vertexes and edges that Nebula Graph can recognize, and then importing them into Nebula Graph 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
- Column database: ClickHouse
- Stream processing software platform: Apache Kafka®
- Publish/Subscribe messaging platform: Apache Pulsar 2.4.5

In addition to importing data as nGQL statements, Exchange supports generating **SST** files for data sources and then importing SST files via Console.

Last update: September 2, 2021

15.1.2 Limitations

This topic describes some of the limitations of using Exchange 2.x.

Nebula Graph releases

The correspondence between the Nebula Exchange release (the JAR version) and the Nebula Graph release is as follows.

Exchange client	Nebula Graph
2.5-SNAPSHOT	v2-nightly
2.5.0	2.5.0
2.1.0	2.0.0, 2.0.1
2.0.1	2.0.0, 2.0.1
2.0.0	2.0.0, 2.0.1

JAR packages are available in two ways: compile them yourself or download them from the Maven repository.

If you are using Nebula Graph 1.x, use Nebula Exchange 1.x.

Environment

Exchange 2.x supports the following operating systems:

- CentOS 7
- macOS

Software depend on

To ensure the normal operation of Exchange, ensure that the following software has been installed on the machine:

- Apache Spark: 2.4.x
- Java: 1.8
- Scala: 2.10.7, 2.11.12 or 2.12.10

Hadoop Distributed File System (HDFS) needs to be deployed in the following scenarios:

- Migrate HDFS data
- Generate SST file

Last update: September 2, 2021

15.2 Compile Exchange

This topic describes how to compile Nebula Exchange. Users can also download the compiled `.jar` file directly.

15.2.1 Prerequisites

- Install Maven.
- Download pulsar-spark-connector_2.11, and unzip it to `io/streamnative/connectors` directory of the local Maven library.

15.2.2 Steps

1. Clone the repository `nebula-spark-utils` in the `/` directory.

```
git clone -b v2.5 https://github.com/vesoft-inc/nebula-spark-utils.git
```

2. Switch to the directory `nebula-exchange`.

```
cd nebula-spark-utils/nebula-exchange
```

3. Package Nebula Exchange.

```
mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true
```

After the compilation is successful, you can view a directory structure similar to the following in the current directory.

```
.
├── README-CN.md
├── README.md
├── pom.xml
├── src
│   ├── main
│   └── test
└── target
    ├── classes
    ├── classes.timestamp
    ├── maven-archiver
    ├── nebula-exchange-2.x.y-javadoc.jar
    ├── nebula-exchange-2.x.y-sources.jar
    ├── nebula-exchange-2.x.y.jar
    ├── original-nebula-exchange-2.x.y.jar
    └── site
```

In the `target` directory, users can find the `exchange-2.x.y.jar` file.

> **Note**
>
> The JAR file version changes with the release of the Nebula 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`.

15.2.3 Failed to download the dependency package

If downloading dependencies fails at compile time:

- 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>
 <mirrorOf>central</mirrorOf>
 <name>aliyun maven</name>
 <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
</mirror>
```

Last update: September 1, 2021

15.3 Exchange configurations

15.3.1 Options for import

After editing the configuration file, run the following commands to import specified source data into the Nebula Graph database.

- First import

```
<spark_install_path>/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.x.y.jar_path> -c <application.conf_path>
```

- Import the reload file

  If some data fails to be imported during the first import, the failed data will be stored in the reload file. Use the parameter `-r` to import the reload file.

```
<spark_install_path>/bin/spark-submit --master "local" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.x.y.jar_path> -c <application.conf_path> -
r "<reload_file_path>"
```

Note

The version number of a JAR file is subject to the name of the JAR file that is actually compiled.

Note

If users use the yarn-cluster mode to submit a job, see the following command:

```
$SPARK_HOME/bin/spark-submit     --master yarn-cluster \
--class com.vesoft.nebula.exchange.Exchange \
--files application.conf \
--conf spark.driver.extraClassPath=./ \
--conf spark.executor.extraClassPath=./ \
nebula-exchange-2.5.0.jar \
-c application.conf
```

The following table lists command parameters.

Paramenter	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.
`-c` / `--config`	Yes	-	Specify the path of the configuration file.
`-h` / `--hive`	No	`false`	Indicate support for importing Hive data.
`-D` / `--dry`	No	`false`	Check whether the format of the configuration file meets the requirements, but it does not check whether the configuration items of `tags` and `edges` are correct. This parameter cannot be added when users 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.

Last update: September 1, 2021

15.3.2 Parameters in the configuration file

This document describes how to configure the file `application.conf` when users use Nebula Exchange.

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)
- Nebula Graph Configurations
- Vertex Configurations
- Edge Configurations

Spark Configurations

This document lists only some Spark parameters. For more information, see Spark Configuration.

Parameter	Type	Default Value	Required	Description
`spark.app.name`	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	`1G`	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	`1G`	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.

Parameter	Type	Default Value	Required	Description
`hive.warehouse`	string	-	Yes	The warehouse path in HDFS. Enclose the path in double quotes and start with `hdfs://`.
`hive.connectionURL`	string	-	Yes	The URL of a JDBC connection. For example, `"jdbc:mysql://127.0.0.1:3306/hive_spark?characterEncoding=UTF-8"`.
`hive.connectionDriverName`	string	`"com.mysql.jdbc.Driver"`	Yes	The driver name.
`hive.connectionUserName`	list[string]	-	Yes	The username for connections.
`hive.connectionPassword`	list[string]	-	Yes	The account password.

Nebula Graph Configurations

Parameter	Type	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 Nebula Graph.
`nebula.pswd`	string	-	Yes	The account password.
`nebula.space`	string	-	Yes	The name of the graph space where data needs to be imported.
`nebula.path.local`	string	`"/tmp"`	No	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.

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	Type	Default Value	Required	Description
`tags.name`	string	-	Yes	The tag name defined in Nebula Graph.
`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 `SST`.
`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 Nebula Graph, 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.batch`	int	`256`	Yes	The maximum number of vertices written into Nebula Graph in a single batch.
`tags.partition`	int	`32`	Yes	The number of Spark partitions.

SPECIFIC PARAMETERS OF PARQUET/JSON/ORC DATA SOURCES

Parameter	Type	Default Value	Required	Description
`tags.path`	string	-	Yes	The path of vertex data files in HDFS. Enclose the path in double quotes and start with `hdfs://`.

SPECIFIC PARAMETERS OF CSV DATA SOURCES

Parameter	Type	Default Value	Required	Description
`tags.path`	string	-	Yes	The path of vertex data files in HDFS. Enclose the path in double quotes and start with `hdfs://`.
`tags.separator`	string	`,`	Yes	The separator. The default value is a comma (,).
`tags.header`	bool	`true`	Yes	Whether the file has a header.

SPECIFIC PARAMETERS OF HIVE DATA SOURCES

Parameter	Type	Default Value	Required	Description
`tags.exec`	string	-	Yes	The statement to query data sources. For example, `select name,age from mooc.users`.

SPECIFIC PARAMETERS OF MAXCOMPUTE DATA SOURCES

Parameter	Type	Default Value	Required	Description
`tags.table`	string	-	Yes	The Maxcompute table name.
`tags.project`	string	-	Yes	The MaxCompute project name.
`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	Type	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 Neo4j server address.
`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 DATA SOURCES

Parameter	Type	Default Value	Required	Description
`tags.host`	string	-	Yes	The MySQL server address.
`tags.port`	string	-	Yes	The MySQL 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 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 basketball.team order by teamid;"`.

SPECIFIC PARAMETERS OF CLICKHOUSE DATA SOURCES

Parameter	Type	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.

SPECIFIC PARAMETERS OF HBASE DATA SOURCES

Parameter	Type	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 which a table belongs to.

SPECIFIC PARAMETERS OF PULSAR DATA SOURCES

Parameter	Type	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.
`tags.options.<topic|topics| topicsPattern>`	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	Type	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 OF SST DATA SOURCES

Parameter	Type	Default Value	Required	Description
`tags.path`	string	-	Yes	The path of the source file specified to generate 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	Type	Default Value	Required	Description
`edges.name`	string	-	Yes	The edge type name defined in Nebula Graph.
`edges.type.source`	string	-	Yes	The data source of edges. For example, `csv`.
`edges.type.sink`	string	`client`	Yes	The method specified to import data. Optional values are `client` and `SST`.
`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 Nebula Graph, 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 starting vertices of edges. For example, `_c0` indicates a value in the first column that is used as a starting vertex of an edge.
`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 a destination vertex of an edge.
`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 Nebula Graph in a single batch.
`edges.partition`	int	`32`	Yes	The number of Spark partitions.

Last update: September 2, 2021

15.4 Use Nebula Exchange

15.4.1 Import data from CSV files

This topic provides an example of how to use Exchange to import Nebula Graph data stored in HDFS or local CSV files.

To import a local CSV file to Nebula Graph, see Nebula Importer.

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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and Nebula Graph 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 NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

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 FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings CSV data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
```

```
}

# Nebula Graph 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"]
    meta:["127.0.0.1:9559"]
  }

  # The account entered must have write permission for the Nebula Graph space.
  user: root
  pswd: nebula

  # Fill in the name of the graph space you want to write data to in the Nebula Graph.
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# Processing vertex
tags: [
  # Set information about Tag player.
  {
    name: player
    type: {
      # Specify the data source file format, set to CSV.
      source: csv

      # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula Graph.
    # The sequence of fields and nebula.fields must correspond to each other.
    # If multiple column names need to be specified, separate them by commas.
    nebula.fields: [age, name]

    # Specify a column of data in the table as the source of vertex VID in the Nebula Graph.
    # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
    vertex: {
      field:_c0
      # policy:hash
    }

    # The delimiter specified. The default value is comma.
    separator: ","

    # If the CSV file have header, set the header to true.
    # If the CSV file does not have header, set the header to false. The default value is false.
    header: false

    # Number of pieces of data written to Nebula Graph in a single batch.
    batch: 256

    # Number of Spark partitions
    partition: 32
  }

  # Set Tag Team information.
  {
    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 edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow
      type: {
        # Specify the data source file format, set to CSV.
        source: csv

        # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula Graph.
      # The sequence of fields and nebula.fields must correspond to each other.
      nebula.fields: [degree]

      # Specify a column as the source for the starting and destination vertexes.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      source: {
        field: _c0
      }
      target: {
        field: _c1
      }

      # The delimiter specified. The default value is comma.
      separator: ","

      # (optionally) Specify a column as the source of the rank.

      #ranking: rank

      # If the CSV file have header, set the header to true.
      # If the CSV file does not have header, set the header to false. The default value is false.
      header: false

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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 NEBULA GRAPH

Run the following command to import CSV data into Nebula Graph. 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-2.5.0.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.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 6: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.2 Import data from JSON files

This topic provides an example of how to use Exchange to import Nebula Graph 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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and Nebula Graph 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 NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create 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 FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings JSON data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
  }

  # Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
    }

    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula

    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }

  # Processing vertex
  tags: [
    # Set information about Tag player.
    {
      name: player
      type: {
        # Specify the data source file format, set to JSON.
        source: json

        # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula Graph.
      # 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 Nebula Graph.
      # 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 Nebula Graph.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      vertex: {
        field:id
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256
```

```
      # Number of Spark partitions
      partition: 32
    }

    # Set Tag Team information.
    {
      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 edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow
      type: {
        # Specify the data source file format, set to JSON.
        source: json

        # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula Graph.
      # 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 Nebula Graph.
      # The sequence of fields and nebula.fields must correspond to each other.
      nebula.fields: [degree]

      # Specify a column as the source for the starting and destination vertexes.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      source: {
        field: src
      }
      target: {
        field: dst
      }


      # (optionally) Specify a column as the source of the rank.
      #ranking: rank

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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 NEBULA GRAPH

Run the following command to import JSON data into Nebula Graph. 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-2.5.0.jar_path> -c <json_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 6: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.3 Import data from ORC files

This topic provides an example of how to use Exchange to import Nebula Graph data stored in HDFS or local ORC files.

To import a local ORC file to Nebula Graph, see Nebula Importer.

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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and Nebula Graph 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 NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

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 FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings ORC data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
  }

  # Nebula Graph 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"]
```

```
    meta:["127.0.0.1:9559"]
  }

  # The account entered must have write permission for the Nebula Graph space.
  user: root
  pswd: nebula

  # Fill in the name of the graph space you want to write data to in the Nebula Graph.
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}

# Processing vertex
tags: [
  # Set information about Tag player.
  {
    name: player
    type: {
      # Specify the data source file format, set to ORC.
      source: orc

      # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula Graph.
    # If multiple values need to be specified, separate them with commas.
    fields: [age,name]

    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the Nebula Graph.
    # 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 Nebula Graph.
    # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
    vertex: {
      field:id
    }

    # Number of pieces of data written to Nebula Graph in a single batch.
    batch: 256

    # Number of Spark partitions
    partition: 32
  }

  # Set Tag Team information.
  {
    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 edge
edges: [
  # Set information about Edge Type follow
  {
    # The corresponding Edge Type name in Nebula Graph.
    name: follow
    type: {
      # Specify the data source file format, set to ORC.
      source: orc
```

```
        # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula Graph.
      fields: [degree]

      # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the Nebula Graph.
      # The sequence of fields and nebula.fields must correspond to each other.
      nebula.fields: [degree]

      # Specify a column as the source for the starting and destination vertexes.
      # The values of vertex must be consistent with the fields in the Parquet file.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      source: {
        field: src
      }
      target: {
        field: dst
      }


      # (optionally) Specify a column as the source of the rank.
      #ranking: rank

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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 NEBULA GRAPH

Run the following command to import ORC data into Nebula Graph. 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-2.5.0.jar_path> -c <orc_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 6: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.4 Import data from Parquet files

This topic provides an example of how to use Exchange to import Nebula Graph data stored in HDFS or local Parquet files.

To import a local Parquet file to Nebula Graph, see Nebula Importer.

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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- If files are stored in HDFS, ensure that the Hadoop service is running properly.
- If files are stored locally and Nebula Graph 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 NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

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 FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings Parquet data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    executor: {
        memory:1G
    }

    cores {
      max: 16
    }
  }

  # Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
    }

    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula

    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }

  # Processing vertex
  tags: [
    # Set information about Tag player.
    {
      name: player
      type: {
        # Specify the data source file format, set to Parquet.
        source: parquet

        # Specifies how to import the data into Nebula Graph: 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 around the path, starting with file://, for example, "file:///tmp/xx.parquet".
      path: "hdfs://192.168.*.13:9000/data/vertex_player.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 Nebula
Graph.
      # If multiple values need to be specified, separate them with commas.
      fields: [age,name]

      # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the Nebula Graph.
      # 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 Nebula Graph.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      vertex: {
        field:id
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set Tag Team information.
    {
      name: team
      type: {
        source: parquet
        sink: client
      }
      path: "hdfs://192.168.*.13:9000/data/vertex_team.parquet"
      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 edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow
      type: {
```

```
        # Specify the data source file format, set to Parquet.
        source: parquet

        # Specifies how to import the data into Nebula Graph: 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 around the 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 Nebula
Graph.
      # If multiple values need to be specified, separate them with commas.
      fields: [degree]

      # Specify the column names in the follow table in fields, and their corresponding values are specified as properties in the Nebula Graph.
      # The sequence of fields and nebula.fields must correspond to each other.
      nebula.fields: [degree]

      # Specify a column as the source for the starting and destination vertexes.
      # The values of vertex must be consistent with the fields in the Parquet file.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      source: {
        field: src
      }
      target: {
        field: dst
      }


      # (optionally) Specify a column as the source of the rank.
      #ranking: rank

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about Edge Type serve.
    {
      name: serve
      type: {
        source: parquet
        sink: client
      }
      path: "hdfs://192.168.*.13:9000/data/edge_serve.parquet"
      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 NEBULA GRAPH

Run the following command to import Parquet data into Nebula Graph. 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-2.5.0.jar_path> -c <parquet_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-
exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 6: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.5 Import data from HBase

This topic provides an example of how to use Exchange to import Nebula Graph 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                                COLUMN+CELL
 player100                         column=cf:age, timestamp=1618881347530, value=42
 player100                         column=cf:name, timestamp=1618881354604, value=Tim Duncan
 player101                         column=cf:age, timestamp=1618881369124, value=36
 player101                         column=cf:name, timestamp=1618881379102, value=Tony Parker
 player102                         column=cf:age, timestamp=1618881386987, value=33
 player102                         column=cf:name, timestamp=1618881393370, value=LaMarcus Aldridge
 player103                         column=cf:age, timestamp=1618881402002, value=32
 player103                         column=cf:name, timestamp=1618881407882, value=Rudy Gay
 ...

hbase(main):003:0> scan "team"
ROW                                COLUMN+CELL
 team200                           column=cf:name, timestamp=1618881445563, value=Warriors
 team201                           column=cf:name, timestamp=1618881453636, value=Nuggets
 ...

hbase(main):004:0> scan "follow"
ROW                                COLUMN+CELL
 player100                         column=cf:degree, timestamp=1618881804853, value=95
 player100                         column=cf:dst_player, timestamp=1618881791522, value=player101
 player101                         column=cf:degree, timestamp=1618881824685, value=90
 player101                         column=cf:dst_player, timestamp=1618881816042, value=player102
 ...

hbase(main):005:0> scan "serve"
ROW                                COLUMN+CELL
 player100                         column=cf:end_year, timestamp=1618881899333, value=2016
 player100                         column=cf:start_year, timestamp=1618881890117, value=1997
 player100                         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
- Hadoop: 2.9.2, Pseudo-distributed deployment
- HBase: 2.2.7
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- The Hadoop service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

For more information, see Quick start workflow.

STEP 2: MODIFY CONFIGURATION FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings HBase data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
```

```
  cores {
    max: 16
  }
}


# Nebula Graph 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"]
    meta:["127.0.0.1:9559"]
  }
  # The account entered must have write permission for the Nebula Graph space.
  user: root
  pswd: nebula
  # Fill in the name of the graph space you want to write data to in the Nebula Graph.
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# Processing vertex
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 Nebula Graph.
    name: player
    type: {
      # Specify the data source file format, set to HBase.
      source: hbase
      # Specifies how to import the data into Nebula Graph: 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 Nebula Graph.
    # 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 Nebula Graph.
    # For example, if rowkey is the source of the VID, enter rowkey.
    vertex:{
        field:rowkey
    }


    # Number of pieces of data written to Nebula Graph in a single batch.
    batch: 256

    # Number of Spark partitions
    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
  }

]
```

```
  # Processing edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow

      type: {
        # Specify the data source file format, set to HBase.
        source: hbase

        # Specifies how to import the data into Nebula Graph: 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 Nebula Graph.
      # 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.

      source:{
          field:rowkey
      }

      # In target, use a column in the follow table as the source of the edge's destination vertex.
      target:{
          field:dst_player
      }


      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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
      }

      batch: 256
      partition: 32
    }
  ]
}
```

STEP 3: IMPORT DATA INTO NEBULA GRAPH

Run the following command to import HBase data into Nebula Graph. 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-2.5.0.jar_path> -c <hbase_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 5: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.6 Import data from MySQL

This topic provides an example of how to use Exchange to import Nebula Graph data stored in MySQL.

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 player;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| playerid | varchar(30) | YES  |     | NULL    |       |
| age      | int         | YES  |     | NULL    |       |
| name     | varchar(30) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+

mysql> desc team;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| teamid | varchar(30) | YES  |     | NULL    |       |
| name   | varchar(30) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+

mysql> desc follow;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| src_player | varchar(30) | YES  |     | NULL    |       |
| dst_player | varchar(30) | YES  |     | NULL    |       |
| degree     | int         | YES  |     | NULL    |       |
+------------+-------------+------+-----+---------+-------+

mysql> desc serve;
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| playerid   | varchar(30) | YES  |     | NULL    |       |
| teamid     | varchar(30) | YES  |     | NULL    |       |
| start_year | int         | YES  |     | NULL    |       |
| end_year   | int         | YES  |     | 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
- Hadoop: 2.9.2, Pseudo-distributed deployment
- MySQL: 8.0.23
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- The Hadoop service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

For more information, see Quick start workflow.

STEP 2: MODIFY CONFIGURATION FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
```

```
  cores {
    max: 16
  }
}

# Nebula Graph 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"]
    meta:["127.0.0.1:9559"]
  }
  # The account entered must have write permission for the Nebula Graph space.
  user: root
  pswd: nebula
  # Fill in the name of the graph space you want to write data to in the Nebula Graph.
  space: basketballplayer
  connection {
    timeout: 3000
    retry: 3
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# Processing vertex
tags: [
  # Set information about Tag player.
  {
    # The Tag name in Nebula Graph.
    name: player
    type: {
      # Specify the data source file format, set to MySQL.
      source: mysql
      # Specifies how to import the data into Nebula Graph: Client or SST.
      sink: client
    }

    host:192.168.*.*
    port:3306
    database:"basketball"
    table:"player"
    user:"test"
    password:"123456"
    sentence:"select playerid, age, name from basketball.player order by playerid;"

    # Specify the column names in the player table in fields, and their corresponding values are specified as properties in the Nebula Graph.
    # 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 Nebula Graph.
    vertex: {
      field:playerid
    }

    # Number of pieces of data written to Nebula Graph in a single batch.
    batch: 256

    # Number of Spark partitions
    partition: 32
  }
  # Set Tag Team information.
  {
    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 basketball.team order by teamid;"

    fields: [name]
    nebula.fields: [name]
    vertex: {
      field: teamid
    }
```

```
      batch: 256
      partition: 32
    }

  ]

  # Processing edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow

      type: {
        # Specify the data source file format, set to MySQL.
        source: mysql

        # Specifies how to import the data into Nebula Graph: Client or SST.
        sink: client
      }

      host:192.168.*.*
      port:3306
      database:"basketball"
      table:"follow"
      user:"test"
      password:"123456"
      sentence:"select src_player,dst_player,degree from basketball.follow order by src_player;"

      # Specify the column names in the follow table in Fields, and their corresponding values are specified as properties in the Nebula Graph.
      # 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.

      source: {
        field: src_player
      }

      # In target, use a column in the follow table as the source of the edge's destination vertex.
      target: {
        field: dst_player
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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 basketball.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 NEBULA GRAPH

Run the following command to import MySQL data into Nebula Graph. 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-2.5.0.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.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 5: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.7 Import data from ClickHouse

This topic provides an example of how to use Exchange to import data stored on ClickHouse into Nebula Graph.

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
- ClickHouse: docker deployment yandex/clickhouse-server tag: latest(2021.07.01)
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- The Hadoop service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

For more information, see Quick start workflow.

STEP 2: MODIFY CONFIGURATION FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings ClickHouse data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }

# Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
    }
    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
```

```
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # Processing vertex
  tags: [
    # Set information about Tag player.
    {
      name: player
      type: {
        # Specify the data source file format, set to ClickHouse.
        source: clickhouse
        # Specifies how to import the data into Nebula Graph: Client or SST.
        sink: client
      }

      # JDBC URL of ClickHouse
      url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"

      user:"user"
      password:"123456"

      # Number of ClickHouse partition
      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 Nebula Graph.
      # 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 Nebula Graph.
      vertex: {
        field:playerid
        # policy:hash
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set Tag Team information.
    {
      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 edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow

      type: {
        # Specify the data source file format, set to ClickHouse.
        source: clickhouse

        # Specifies how to import the data into Nebula Graph: Client or SST.
        sink: client
      }

      # JDBC URL of ClickHouse
      url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"

      user:"user"
      password:"123456"
```

```
      # Number of ClickHouse partition
      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 Nebula Graph.
      # 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.
      source: {
        field:src_player
      }

      # In target, use a column in the follow table as the source of the edge's destination vertex.
      target: {
        field:dst_player
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about Edge Type serve
    {
      name: serve
      type: {
        source: clickhouse
        sink: client
      }
      url:"jdbc:clickhouse://192.168.*.*:8123/basketballplayer"
      user:"user"
      password:"123456"
      numPartition:"5"
      sentence:"select * 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 NEBULA GRAPH

Run the following command to import ClickHouse data into Nebula Graph. 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-2.5.0.jar_path> -c <clickhouse_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 5: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.8 Import data from Neo4j

This topic provides an example of how to use Exchange to import Nebula Graph data stored in Neo4j.

Implementation method

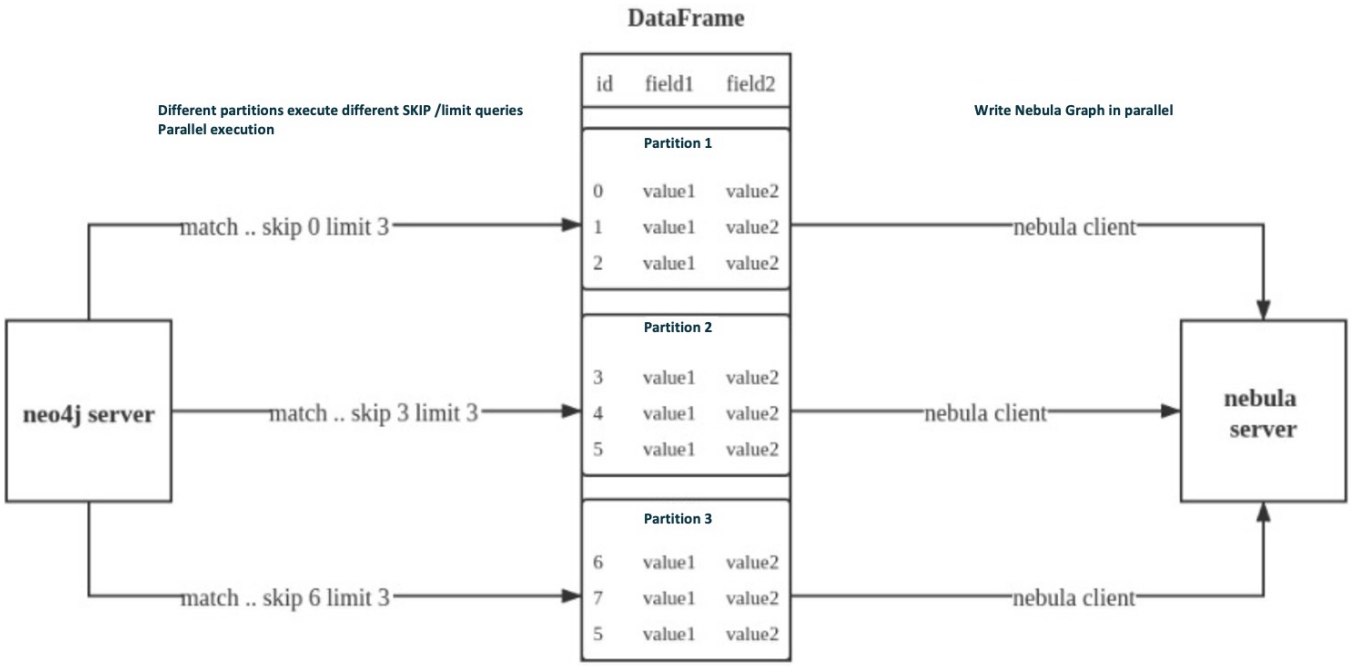
Exchange uses **Neo4j Driver 4.0.1** to read Neo4j data. Before batch export, you need to write Cypher statements that are automatically executed based on labels and relationship types and the number of Spark partitions in the configuration file to improve data export performance.

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 continued state, Reader calculates the offset and size that each Spark partition should have.
3. In each Spark partition, The Exchange Reader 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 Nebula Graph 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: 1.7 GHz Quad-Core Intel Core i7
  - CPU cores: 14
  - memory: 16 GB
- Spark: Stand-alone, 2.4.6 pre-build for Hadoop 2.7
- Neo4j: 3.5.20 Community Edition
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create 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 user manual.

STEP 3: MODIFY CONFIGURATION FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings Neo4j data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }

    driver: {
      cores: 1
      maxResultSize: 1G
    }

    executor: {
        memory:1G
    }

    cores:{
      max: 16
    }
  }


  # Nebula Graph configuration
  nebula: {
    address:{
      graph:["127.0.0.1:9669"]
      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 vertex
  tags: [


    # Set information about Tag player.
    {
      name: player
      type: {
        # Specify the data source file format, set to Neo4j.
        source: neo4j
         # Specifies how to import the data into Nebula Graph: Client or SST.
        sink: client
      }
      # Neo4j Server IP address and port number.
      server: "bolt://192.168.*.*:7687"

      user: neo4j
      password:neo4j

      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
      }
      partition: 10
      batch: 1000
      check_point_path: /tmp/test
   }
  # Set Tag Team information.
  {
      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
      }
      partition: 10
      batch: 1000
      check_point_path: /tmp/test
   }
  ]

  # Processing edge
  edges: [
    # Set information about Edge Type follow
    {
      name: follow
      type: {
        source: neo4j
        sink: client
      }
      server: "bolt://192.168.*.*:7687"
      user: neo4j
      password:neo4j
      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
      }
      target: {
        field: dst
      }
      partition: 10
      batch: 1000
      check_point_path: /tmp/test
    }
   # Set information about Edge Type serve
   {
      name: serve
      type: {
        source: neo4j
        sink: client
      }
      server: "bolt://192.168.*.*:7687"
      user: neo4j
      password:neo4j
      database:neo4j
      exec: "match (a:player)-[r:serve]->(b:team) return a.id as src, b.id as dst, r.start_year as start_year, r.end_year as end_year  order by id(r)"
      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source: {
        field: src
      }
      target: {
        field: dst
      }
      partition: 10
      batch: 1000
      check_point_path: /tmp/test
    }
   ]
}
```

Exec Configuration Description

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 as sorting properties. If there is no index, users can also observe the default sort and select the appropriate properties for sorting to improve efficiency. If the default sort cannot be found, users can sort by the ID of the vertex or relationship and set the `partition` to a small value to reduce the sorting pressure of Neo4j.

> **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

Nebula Graph 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 Nebula Graph'S ID and the value is duplicated in Neo4j, duplicate ids will result. One and only one of their corresponding data will be stored in the Nebula Graph, and the others will be overwritten. Because the data import process is concurrently writing data to Nebula Graph, the resulting saved data is not guaranteed to be the latest data in Neo4j.

check_point_path

If breakpoint continuation is enabled, to avoid data loss, the state of the database should not change between the breakpoint and the continuation, for example, data cannot be added or deleted, and the `partition` quantity configuration should not be changed.

STEP 4: IMPORT DATA INTO NEBULA GRAPH

Run the following command to import Neo4j data into Nebula Graph. 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-2.5.0.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.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 6: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.9 Import data from Hive

This topic provides an example of how to use Exchange to import Nebula Graph 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.

```
scala> spark.sql("describe basketball.player").show
+--------+---------+-------+
|col_name|data_type|comment|
+--------+---------+-------+
|playerid|   string|   null|
|     age|   bigint|   null|
|    name|   string|   null|
+--------+---------+-------+

scala> spark.sql("describe basketball.team").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|    teamid|   string|   null|
|      name|   string|   null|
+----------+---------+-------+

scala> spark.sql("describe basketball.follow").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|src_player|   string|   null|
|dst_player|   string|   null|
|    degree|   bigint|   null|
+----------+---------+-------+

scala> spark.sql("describe basketball.serve").show
+----------+---------+-------+
|  col_name|data_type|comment|
+----------+---------+-------+
|  playerid|   string|   null|
|    teamid|   string|   null|
|start_year|   bigint|   null|
|  end_year|   bigint|   null|
+----------+---------+-------+
```

> **Note**
>
> The Hive data type `bigint` corresponds to the Nebula Graph `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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- Hadoop has been installed and started, and the Hive Metastore database (MySQL in this example) has been started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

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|LaMarcus 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` settings Hive data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    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: {
  #  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"
  #}

  # Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
    }
    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # Processing vertex
  tags: [
    # Set information about Tag player.
    {
      # The Tag name in Nebula Graph.
      name: player
      type: {
        # Specify the data source file format, set to Hive.
        source: hive
        # Specifies how to import the data into Nebula Graph: 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 Nebula Graph.
      # 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 Nebula Graph.
      vertex:{
        field:playerid
```

```
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }
    # Set Tag Team information.
    {
      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 edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow

      type: {
        # Specify the data source file format, set to Hive.
        source: hive

        # Specifies how to import the data into Nebula Graph: Client or SST.
        sink: client
      }

      # Set the SQL statement to read the data of follow table in 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 Nebula Graph.
      # 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.
      source: {
        field: src_player
      }

      # In target, use a column in the follow table as the source of the edge's destination vertex.
      target: {
        field: dst_player
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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
      }
      batch: 256
      partition: 32
    }
  ]
}
```

STEP 4: IMPORT DATA INTO NEBULA GRAPH

Run the following command to import Hive data into Nebula Graph. 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-2.5.0.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.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 6: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.10 Import data from MaxCompute

This topic provides an example of how to use Exchange to import Nebula Graph 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
- Hadoop: 2.9.2, Pseudo-distributed deployment
- MaxCompute: Alibaba Cloud official version
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- The Hadoop service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

For more information, see Quick start workflow.

STEP 2: MODIFY CONFIGURATION FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings MaxCompute data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }

  # Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
    }
    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
```

```
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # Processing vertex
  tags: [
    # Set information about Tag player.
    {
      name: player
      type: {
        # Specify the data source file format, set to MaxCompute.
        source: maxcompute
        # Specifies how to import the data into Nebula Graph: Client or SST.
        sink: client
      }

      # Table name of MaxCompute.
      table:player

      # Project name of MaxCompute.
      project:project

      # OdpsUrl and tunnelUrl for MaxCompute service.
      # Address at 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

      # Table partition description of MaxCompute. 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 Nebula Graph.
      # 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 Nebula Graph.
      vertex:{
        field: playerid
      }

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set Tag Team information.
    {
      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"
      fields:[name]
      nebula.fields:[name]
      vertex:{
        field: teamid
      }
      batch: 256
      partition: 32
    }
  ]

  # Processing edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow

      type:{
```

```
        # Specify the data source file format, set to MaxCompute.
        source:maxcompute

        # Specifies how to import the data into Nebula Graph: Client or SST.
        sink:client
      }

      # Table name of MaxCompute.
      table:follow

      # Project name of MaxCompute.
      project:project

      # OdpsUrl and tunnelUrl for MaxCompute service.
      # Address at 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

      # Table partition description of MaxCompute. 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 Nebula Graph.
      # 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.
      source:{
        field: src_player
      }

      # In target, use a column in the follow table as the source of the edge's destination vertex.
      target:{
        field: dst_player
      }

      # Number of Spark partitions
      partition:10

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch:10
    }

    # Set information about 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
      }
      partition:10
      batch:10
    }
  ]
}
```

STEP 3: IMPORT DATA INTO NEBULA GRAPH

Run the following command to import MaxCompute data into Nebula Graph. 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-2.5.0.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.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 5: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.11 Import data from Pulsar

This topic provides an example of how to use Exchange to import Nebula Graph 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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Pulsar service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

For more information, see Quick start workflow.

STEP 2: MODIFY CONFIGURATION FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings Pulsar data source configuration. In this case, 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: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }


  # Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
    }

    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula

    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
```

```
  }
  execution {
    retry: 3
  }
  error: {
    max: 32
    output: /tmp/errors
  }
  rate: {
    limit: 1024
    timeout: 1000
  }
}
# Processing vertex
tags: [
  # Set information about Tag player.
  {
    name: player
    type: {
      # Specify the data source file format, set to Pulsar.
      source: pulsar
      # Specifies how to import the data into Nebula Graph: Client or SST.
      sink: client
    }
    # Pulsar server address.
    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 Nebula Graph.
    # 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 Nebula Graph.
    vertex:{
        field:playerid
    }


    # Number of pieces of data written to Nebula Graph in a single batch.
    batch: 10

    # Number of Spark partitions
    partition: 10
    # Read message interval. Unit: second.
    interval.seconds: 10
  }
  # Set Tag Team information.
  {
    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 edge
edges: [
  # Set information about Edge Type follow
  {
    # The corresponding Edge Type name in Nebula Graph.
    name: follow

    type: {
      # Specify the data source file format, set to Pulsar.
      source: pulsar

      # Specifies how to import the data into Nebula Graph: Client or SST.
      sink: client
    }

    # Pulsar server address.
    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 Nebula Graph.
      # 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.

      source:{
          field:src_player
      }

      # In target, use a column in the follow table as the source of the edge's destination vertex.
      target:{
          field:dst_player
      }


      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 10

      # Number of Spark partitions
      partition: 10

      # Read message interval. Unit: second.
      interval.seconds: 10
    }

    # Set information about 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"
      }

      fields: [start_year,end_year]
      nebula.fields: [start_year,end_year]
      source:{
          field:playerid
      }

      target:{
          field:teamid
      }

      batch: 10
      partition: 10
      interval.seconds: 10
    }
  ]
}
```

STEP 3: IMPORT DATA INTO NEBULA GRAPH

Run the following command to import Pulsar data into Nebula Graph. 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-2.5.0.jar_path> -c <pulsar_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 5: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 1, 2021

15.4.12 Import data from Kafka

This topic provides a simple guide to importing Data stored on Kafka into Nebula Graph using Exchange.

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
- Nebula Graph: 2.5.0 (Deploy Nebula Graph with Docker Compose)

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- Learn about the Schema created in Nebula Graph, including Tag and Edge type names, properties, and more.
- The Kafka service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

For more information, see Quick start workflow.

STEP 2: MODIFY CONFIGURATION FILE

> **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` settings Kafka data source configuration. In this case, the copied file is called `kafka_application.conf`. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
    app: {
      name: Nebula Exchange 2.5.0
    }
    driver: {
      cores: 1
      maxResultSize: 1G
    }
    cores {
      max: 16
    }
  }


  # Nebula Graph 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"]
      meta:["127.0.0.1:9559"]
```

```
    }
    # The account entered must have write permission for the Nebula Graph space.
    user: root
    pswd: nebula
    # Fill in the name of the graph space you want to write data to in the Nebula Graph.
    space: basketballplayer
    connection {
      timeout: 3000
      retry: 3
    }
    execution {
      retry: 3
    }
    error: {
      max: 32
      output: /tmp/errors
    }
    rate: {
      limit: 1024
      timeout: 1000
    }
  }
  # Processing vertex
  tags: [
    # Set information about Tag player.
    {
      name: player
      type: {
        # Specify the data source file format, set to Kafka.
        source: kafka
        # Specifies how to import the data into Nebula Graph: Client or SST.
        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 key for name in Nebula 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 Nebula Graph.
      # The key is the same as the value above, indicating that key is used as both VID and attribute name.
      vertex:{
          field:key
      }


      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 10

      # Number of Spark partitions
      partition: 10
      # Read message interval. Unit: second.
      interval.seconds: 10
    }
    # Set Tag Team information.
    {
      name: team
      type: {
        source: kafka
        sink: client
      }
      service: "127.0.0.1:9092"
      topic: "topic_name2"
      fields: [key]
      nebula.fields: [name]
      vertex:{
          field:key
      }
      batch: 10
      partition: 10
      interval.seconds: 10
    }

  ]

  # Processing edge
  edges: [
    # Set information about Edge Type follow
    {
      # The corresponding Edge Type name in Nebula Graph.
      name: follow

      type: {
        # Specify the data source file format, set to Kafka.
        source: kafka

        # Specifies how to import the data into Nebula Graph: Client or SST.
        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 commas.
      # Specify the field name in fields, for example 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 starting vertex.

      source:{
          field:timestamp
      }

      # In target, use a column in the topic as the source of the edge's destination vertex.
      target:{
          field:offset
      }


      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 10

      # Number of Spark partitions
      partition: 10

      # Read message interval. Unit: second.
      interval.seconds: 10
    }

    # Set information about Edge Type serve
    {
      name: serve
      type: {
        source: kafka
        sink: client
      }
      service: "127.0.0.1:9092"
      topic: "topic_name4"

      fields: [timestamp,offset]
      nebula.fields: [start_year,end_year]
      source:{
          field:key
      }

      target:{
          field:value
      }

      batch: 10
      partition: 10
      interval.seconds: 10
    }
  ]
}
```

STEP 3: IMPORT DATA INTO NEBULA GRAPH

Run the following command to import Kafka data into Nebula Graph. 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-2.5.0.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.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 5: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 2, 2021

15.4.13 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 into Nebula Graph. The sample data source is a CSV file.

> **Note**
>
> The SST file can be imported only in Linux.

Background information

Exchange supports two data import modes:

- Import the data from the data source directly into Nebula Graph as **nGQL** statements.
- Generate the SST file from the data source, and use Console to import the SST file into Nebula Graph.

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 and fast import.

Implementation methods

Nebula Graph underlying uses RocksDB as the key-value storage engine. RocksDB is a hard disk based storage engine that provides a series of apis for creating and importing SST files to help quickly import massive data.

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 process is as follows:

1. Reader reads data from the data source.
2. sstProcessor generates the SST file from the Nebula Graph'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 Nebula Graph using the `IngestExternalFile()` method. Such as:

```
IngestExternalFileOptions ifo;
# Import two SST files
Status s = db_->IngestExternalFile({"/home/usr/file1.sst", "/home/usr/file2.sst"}, ifo);
if (!s.ok()) {
  printf("Error while adding file %s and %s, Error %s\n",
         file_path1.c_str(), file_path2.c_str(), s.ToString().c_str());
  return 1;
}
```

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 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
- Nebula Graph: 2.5.0

Prerequisites

Before importing data, you need to confirm the following information:

- Nebula Graph has been installed and deployed with the following information:
  - IP address and port of Graph and Meta services.
  - User name and password with Nebula Graph write permission.
  - `--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, `1977`.
  - `--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`.
  - Schema information, including Tag and Edge type names, properties, and more.
- Exchange has been compiled, or download the compiled `.jar` file directly.
- Spark has been installed.
- JDK 1.8 or later has been installed and the environment variable `JAVA_HOME` has been configured.
- The Hadoop service has been installed and started.

Steps

STEP 1: CREATE THE SCHEMA IN NEBULA GRAPH

Analyze the data to create a Schema in Nebula Graph by following these steps:

1. Identify the Schema elements. The Schema elements in the Nebula Graph 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 Nebula Graph and create a Schema as shown below.

```
## create 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 Tag player
nebula> CREATE TAG player(name string, age int);

## create Tag team
nebula> CREATE TAG team(name string);

## create Edge type follow
nebula> CREATE EDGE follow(degree int);

## create Edge type serve
nebula> CREATE EDGE serve(start_year int, end_year int);
```

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 FILE

After Exchange is compiled, copy the conf file `target/classes/application.conf` settings SST data source configuration. In this case, the copied file is called `sst_application.conf`. For details on each configuration item, see Parameters in the configuration file.

```
{
  # Spark configuration
  spark: {
    app: {
      name: Nebula Exchange 2.0
    }

    master:local

    driver: {
      cores: 1
      maxResultSize: 1G
    }

    executor: {
        memory:1G
    }

    cores:{
      max: 16
    }
  }

  # Nebula Graph configuration
  nebula: {
    address:{
      graph:["127.0.0.1:9669"]
      meta:["127.0.0.1:9559"]
    }
    user: root
    pswd: nebula
    space: basketballplayer

    # SST file configuration
    path:{
        # Local directory that temporarily stores generated SST files
        local:"/tmp"

        # Path for storing the SST file in the HDFS
        remote:"/sst"

        # NameNode address of HDFS
        hdfs.namenode: "hdfs://*.*.*.*:9000"
    }

    # Client connection parameters
    connection {
      # Timeout duration of socket connection and execution, in milliseconds.
      timeout: 30000
    }

    error: {
      # 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 calls to NebulaGraph.
    rate: {
      # Steady throughput of RateLimiter.
      limit: 1024

      # Get the allowed timeout from RateLimiter, in milliseconds
      timeout: 1000
    }
  }
```

```
# Processing vertex
tags: [
  # Set information about Tag player.
  {
    name: player
    type: {
      # Specify the data source file format, set to CSV.
      source: csv

      # Specifies how to import the data into Nebula Graph: 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:port/xx/xx".
    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 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 Nebula Graph.
    # 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 Nebula Graph.
    # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
    vertex: {
      field:_c0
    }

    # The delimiter specified. The default value is comma.
    separator: ","

    # If the CSV file have header, set the header to true.
    # If the CSV file does not have header, set the header to false. The default value is false.
    header: false

    # Number of pieces of data written to Nebula Graph in a single batch.
    batch: 256

    # Number of Spark partitions
    partition: 32
  }

  # Set Tag Team information.
  {
    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
  }


  # If more vertexes need to be added, refer to the previous configuration to add them.
]
# Processing edge
edges: [
  # Set information about Edge Type follow
  {
    # The corresponding Edge Type name in Nebula Graph.
    name: follow
    type: {
      # Specify the data source file format, set to CSV.
      source: csv

      # Specifies how to import the data into Nebula Graph: 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:port/xx/xx".
    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 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 Nebula Graph.
    # The sequence of fields and nebula.fields must correspond to each other.
    nebula.fields: [degree]
```

```
      # Specify a column as the source for the starting and destination vertexes.
      # Currently, Nebula Graph 2.5.0 supports only strings or integers of VID.
      source: {
        field: _c0
      }
      target: {
        field: _c1
      }

      # The delimiter specified. The default value is comma.
      separator: ","

      # (optionally) Specify a column as the source of the rank.

      #ranking: rank

      # If the CSV file have header, set the header to true.
      # If the CSV file does not have header, set the header to false. The default value is false.
      header: false

      # Number of pieces of data written to Nebula Graph in a single batch.
      batch: 256

      # Number of Spark partitions
      partition: 32
    }

    # Set information about 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
    }

  ]
  # 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" --class com.vesoft.nebula.exchange.Exchange <nebula-exchange-2.5.0.jar_path> -c <sst_application.conf_path>
```

> **Note**
>
> JAR packages are available in two ways: compiled them yourself, or download the compiled `.jar` file directly.

Example:

```
${SPARK_HOME}/bin/spark-submit  --master "local" --class com.vesoft.nebula.exchange.Exchange  /root/nebula-spark-utils/nebula-exchange/target/nebula-exchange-2.5.0.jar  -c /root/nebula-spark-utils/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 `nebula.path.remote` parameter) on HDFS.

> **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

> **Caution**
>
> If you need to import SST files to Nebula Graph 2.5.x, please add `--enable_vertex_cache =false` in the Storage configuration file and restart the Storage service to import SST files normally. Otherwise, the old data may not be overwritten.

Connect to the Nebula Graph database using the client tool and import the SST file as follows:

1. 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> DOWNLOAD HDFS "hdfs://<hadoop_address>:<hadoop_port>/<sst_file_path>";
```

Example:

```
nebula> DOWNLOAD HDFS "hdfs://*.*.*.*:9000/sst";
```

3. Run the following command to import the SST file:

```
nebula> INGEST;
```

> **Note**
>
> - To download the SST file again, delete the `download` folder in the space ID in the `data/storage/nebula` directory in the Nebula Graph installation path, and then download the SST file again. If the space is 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 you need to re-import, re-execute `INGEST;`.

STEP 6: (OPTIONAL) VALIDATION DATA

Users can verify that data has been imported by executing a query in the Nebula Graph client (for example, Nebula Graph Studio). Such as:

```
GO FROM "player100" OVER follow;
```

Users can also run the SHOW STATS command to view statistics.

STEP 7: (OPTIONAL) REBUILD INDEXES IN NEBULA GRAPH

With the data imported, users can recreate and rebuild indexes in Nebula Graph. For details, see Index overview.

Last update: September 23, 2021

15.5 Exchange FAQ

15.5.1 Compilation

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`:

```
<mirror>
    <id>alimaven</id>
    <mirrorOf>central</mirrorOf>
    <name>aliyun maven</name>
    <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
</mirror>
```

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.

15.5.2 Execution

How to submit in Yarn-Cluster mode?

To submit a task in Yarn-Cluster mode, run the following command:

```
$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-2.0.0.jar \
-c application.conf
```

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.

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.

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, check 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.

Run error: `com.facebook.thrift.protocol.TProtocolException: Expected protocol id xxx`

Check that the Nebula Graph 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-metad --flagf ...   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-metad --flagf ...   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-metad --flagf ...   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-storaged --fl ...   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-storaged --fl ...   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-storaged --fl ...   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.

15.5.3 Configuration

Which configuration items affect import performance?

- batch: The number of pieces of data contained in each nGQL statement sent to the Nebula Graph service.
- partition: Number of Spark data partitions, indicating the number of concurrent data import.
- nebula.rate: Go to the token bucket to get a token before sending a request to Nebula Graph.

  - 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.

15.5.4 Others

Which versions of Nebula Graph are supported by Exchange?

See Limitations.

What is the relationship between Exchange and Spark Writer?

Exchange is the Spark application developed on the basis of Spark Writer. Both are suitable for bulk migration of cluster data to Nebula Graph in a distributed environment, but later maintenance work will be focused on Exchange. Compared with Spark Writer, Exchange has the following improvements:

- Supports more abundant data sources, such as MySQL, Neo4j, Hive, HBase, Kafka, Pulsar, etc.
- 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 Nebula Graph's Schema, so Exchange adds automatic data type matching and type conversion. When the data type in the Nebula Graph's Schema is non-String, Exchange converts the source data of String type to the corresponding type.

Last update: September 2, 2021

16. Nebula Algorithm

Nebula Algorithm (Algorithm) is a Spark application based on GraphX. It uses a complete algorithm tool to perform graph computing on the data in the Nebula Graph database by submitting a Spark task. You can also programmatically use the algorithm under the lib repository to perform graph computing on DataFrame.

16.1 Prerequisites

Before using the Nebula Algorithm, users need to confirm the following information:

- The Nebula Graph services have been deployed and started. For details, see Nebula Installation.
- The Spark version is 2.4.x.
- (Optional) If users need to clone, compile, and package the latest Algorithm in Github, install Maven.

16.2 Limitations

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.

For non-integer String data, it is recommended to use the algorithm interface. You can use the `dense_rank` function of SparkSQL to encode the data as the Long type instead of the String type.

16.3 Supported algorithms

The graph computing algorithms supported by Nebula Algorithm are as follows.

Algorithm	Description	Scenario
PageRank	The rank of pages	Web page ranking, key node mining
Louvain	Community discovery	Community mining, hierarchical clustering
KCore	K core	Community discovery, financial risk control
LabelPropagation	Label propagation	Information spreading, advertising, and community discovery
ConnectedComponent	Connected component	Community discovery, island discovery
StronglyConnectedComponent	Strongly connected component	Community discovery
ShortestPath	The shortest path	Path planning, network planning
TriangleCount	Triangle counting	Network structure analysis
BetweennessCentrality	Intermediate centrality	Key node mining, node influence computing
DegreeStatic	Degree of statistical	Graph structure analysis

16.4 Implementation methods

Nebula Algorithm implements the graph calculating as follows:

1. Read the graph data of DataFrame from the Nebula Graph database using the Nebula 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.

16.5 Get Nebula Algorithm

16.5.1 Compile and package

1. Clone the repository `nebula-spark-utils`.

```
$ git clone -b v2.5 https://github.com/vesoft-inc/nebula-spark-utils.git
```

2. Enter the directory `nebula-algorithm`.

```
$ cd nebula-spark-utils/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-2.5.0.jar` is generated in the directory `nebula-algorithm/target`.

16.5.2 Download maven from the remote repository

Download address

16.6 How to use

16.6.1 Use algorithm interface (recommended)

The `lib` repository provides 10 common graph algorithms.

1. Add dependencies to the file `pom.xml`.

```
<dependency>
<groupId>com.vesoft</groupId>
<artifactId>nebula-algorithm</artifactId>
<version>2.5.0</version>
</dependency>
```

2. Use the algorithm (take PageRank as an example) by filling in parameters. For more algorithms, see Test cases.

> **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 Nebula Graph).

```
val prConfig = new PRConfig(5, 1.0)
val louvainResult = PageRankAlgo.apply(spark, data, prConfig, false)
```

16.6.2 Submit the algorithm package directly

> **Note**
>
> There are limitations to use sealed packages. For example, when sinking a repository into Nebula Graph, the property name of the tag created in the sunk graph space must match the preset name in the code. The first method is recommended if the user has development skills.

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 Nebula Graph
  nebula: {
    # Data source. When Nebula Graph 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 Nebula Graph 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 Nebula Graph.
        space: basketballplayer
        # Edge types in Nebula Graph. When there are multiple labels, the data of multiple edges will be merged.
        labels: ["serve"]
        # The property name of each edge type in Nebula Graph. 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 Nebula Graph, 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 Nebula Graph 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 Nebula Graph.
        space:nb
        # The name of the tag in Nebula Graph. 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 pagerank, louvain, connectedcomponent,
    # labelpropagation, shortestpaths, degreestatic, kcore,
    # stronglyconnectedcomponent, trianglecount, betweenness,
    executeAlgo: pagerank

    # PageRank
    pagerank: {
        maxIter: 10
        resetProb: 0.15  # The default value is 0.15
    }

    # Louvain
    louvain: {
        maxIter: 20
        internalIter: 10
        tol: 0.5
    }

   # ConnectedComponent/StronglyConnectedComponent
   connectedcomponent: {
       maxIter: 20
   }

   # LabelPropagation
   labelpropagation: {
       maxIter: 20
   }

    # ShortestPath
    shortestpaths: {
        # several vertices to compute the shortest path to all vertices.
        landmarks: "1"
    }

    # DegreeStatic
    degreestatic: {}

    # KCore
    kcore:{
        maxIter:10
        degree:1
    }

    # TriangleCount
    trianglecount:{}

    # BetweennessCentrality
    betweenness:{
        maxIter:5
    }
}
}
```

2. Submit the graph computing task.

```
${SPARK_HOME}/bin/spark-submit --master <mode> --class com.vesoft.nebula.algorithm.Main <nebula-algorithm-2.0.0.jar_path> -p <application.conf_path>
```

Example:

```
${SPARK_HOME}/bin/spark-submit --master "local" --class com.vesoft.nebula.algorithm.Main /root/nebula-spark-utils/nebula-algorithm/target/nebula-
algorithm-2.0.0.jar -p /root/nebula-spark-utils/nebula-algorithm/src/main/resources/application.conf
```

Last update: September 2, 2021

17. Nebula Spark Connector

Nebula Spark Connector is a Spark connector application for reading and writing Nebula Graph data in Spark standard format. Nebula Spark Connector consists of two parts: Reader and Writer.

- Reader

  Provides a Spark SQL interface. This interface can be used to read Nebula Graph 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 Nebula Graph in a row-by-row or batch-import way.

For more information, see Nebula Spark Connector.

17.1 Use cases

Nebula Spark Connector applies to the following scenarios:

- Migrate data between different Nebula Graph clusters.
- Migrate data between different graph spaces in the same Nebula Graph cluster.
- Migrate data between Nebula Graph and other data sources.
- Graph computing with Nebula Algorithm.

17.2 Benefits

The features of Nebula Spark Connector 2.5.0 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 Nebula Graph data into VertexRDD and EdgeRDD, and supports non-Long vertex IDs.
- Unifies the extended data source of SparkSQL, and uses DataSourceV2 to extend Nebula Graph data.
- Two write modes, `insert` and `update`, are supported. `insert` mode will insert (overwrite) data, and `update` mode will only update existing data.

17.3 Get Nebula Spark Connector

17.3.1 Compile package

1. Clone repository `nebula-spark-utils`.

```
$ git clone -b v2.5 https://github.com/vesoft-inc/nebula-spark-utils.git
```

2. Make the `nebula-spark-connector` directory the current working directory.

```
$ cd nebula-spark-utils/nebula-spark-connector
```

3. Compile package.

```
$ mvn clean package -Dmaven.test.skip=true -Dgpg.skip -Dmaven.javadoc.skip=true
```

After compilation, a similar file `nebula-spark-connector-2.5.0-SHANPSHOT.jar` is generated in the directory `nebula-spark-connector/target`.

17.3.2 Download maven remote repository

Download

17.4 How to use

When using Nebula Spark Connector to reading and writing Nebula Graph data, You can refer to the following code.

```
# Read vertex and edge data from Nebula Graph.
spark.read.nebula().loadVerticesToDF()
spark.read.nebula().loadEdgesToDF()

# Write dataframe data into Nebula Graph as vertex and edges.
dataframe.write.nebula().writeVertices()
dataframe.write.nebula().writeEdges()
```

`nebula()` receives two configuration parameters, including connection configuration and read-write configuration.

17.4.1 Reading data from Nebula Graph

```
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 the nebula graph, 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 `ip1:port1,ip2:port2,...`. Read data is no need to configure `withGraphAddress`.
`withConnectionRetry`	No	The number of retries that the Nebula Java Client connected to the Nebula Graph. The default value is `1`.
`withExecuteRetry`	No	The number of retries that the Nebula Java Client executed query statements. The default value is `1`.
`withTimeout`	No	The timeout for the Nebula Java Client request response. The default value is `6000`, Unit: ms.

- `ReadNebulaConfig` is the configuration to read Nebula Graph data, as described below.

Parameter	Required	Description
`withSpace`	Yes	Nebula Graph space name.
`withLabel`	Yes	The Tag or Edge type name within the Nebula Graph space.
`withNoColumn`	No	Whether the property is not read. The default value is `false`, read property. If the value is `true`, the property is not read, the `withReturnCols` 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 Nebula Java Storage Client at a time. The default value is `1000`.
`withPartitionNum`	No	Configures the number of Spark partitions to read the Nebula Graph data. The default value is `100`. This value should not exceed the number of slices in the graph space (partition_num).

17.4.2 Write data into Nebula Graph

```
val config = NebulaConnectionConfig
  .builder()
  .withMetaAddress("127.0.0.1:9559")
  .withGraphAddress("127.0.0.1:9669")
  .withConenctionRetry(2)
  .build()

val nebulaWriteVertexConfig: WriteNebulaVertexConfig = WriteNebulaVertexConfig
  .builder()
  .withSpace("test")
  .withTag("person")
  .withVidField("id")
  .withVidPolicy("hash")
  .withVidAsProp(true)
  .withUser("root")
  .withPasswd("nebula")
  .withBatch(1000)
  .build()
df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()

val nebulaWriteEdgeConfig: WriteNebulaEdgeConfig = WriteNebulaEdgeConfig
  .builder()
  .withSpace("test")
  .withEdge("friend")
  .withSrcIdField("src")
  .withSrcPolicy(null)
  .withDstIdField("dst")
  .withDstPolicy(null)
  .withRankField("degree")
  .withSrcAsProperty(true)
```

```
  .withDstAsProperty(true)
  .withRankAsProperty(true)
  .withUser("root")
  .withPasswd("nebula")
  .withBatch(1000)
  .build()
df.write.nebula(config, nebulaWriteEdgeConfig).writeEdges()
```

The default write mode is `insert`, which can be changed to `update` 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")
  .withVidField("id")
  .withVidAsProp(true)
  .withBatch(1000)
  .withWriteMode(WriteMode.UPDATE)
```

```
  .build()
df.write.nebula(config, nebulaWriteVertexConfig).writeVertices()
```

- `NebulaConnectionConfig` is the configuration for connecting to the nebula graph, 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 `ip1:port1,ip2:port2,...`.
`withGraphAddress`	Yes	Specifies the IP addresses and ports of Graph Services. Separate multiple addresses with commas. The format is `ip1:port1,ip2:port2,...`.
`withConnectionRetry`	No	Number of retries that the Nebula Java Client connected to the Nebula Graph. The default value is `1`.

- `WriteNebulaVertexConfig` is the configuration of the write vertex, as described below.

Parameter	Required	Description
`withSpace`	Yes	Nebula Graph 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, Nebula Graph 2.x use mapping function, supports HASH only. 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	Nebula Graph user name. If authentication is disabled, you do not need to configure the user name and password.
`withPasswd`	No	The password for the Nebula Graph user name.
`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 `insert` and `update`. The default value is `insert`.

- `WriteNebulaEdgeConfig` is the configuration of the write edge, as described below.

Parameter	Required	Description
`withSpace`	Yes	Nebula Graph 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, Nebula Graph 2.x 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, Nebula Graph 2.x 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 `false`. If set to `true`, make sure Edge type has the same property name as `DstIdField`.
`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	Nebula Graph user name. If authentication is disabled, you do not need to configure the user name and password.
`withPasswd`	No	The password for the Nebula Graph user name.
`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 `insert` and `update`. The default value is `insert`.

Last update: September 2, 2021

18. Nebula Flink Connector

Nebula Flink Connector is a connector that helps Flink users quickly access Nebula Graph. Nebula Flink Connector supports reading data from the Nebula Graph database or writing other external data to the Nebula Graph database.

For more information, see Nebula Flink Connector.

18.1 Use cases

Nebula Flink Connector applies to the following scenarios:

- Migrate data between different Nebula Graph clusters.
- Migrate data between different graph spaces in the same Nebula Graph cluster.
- Migrate data between Nebula Graph and other data sources.

Last update: May 20, 2021

19. Nebula Bench

Nebula Bench is a performance test tool for Nebula Graph using the LDBC data set.

19.1 Scenario

- Generate test data and import Nebula Graph.
- Performance testing in the Nebula Graph cluster.

19.2 Test process

1. Generate test data by using ldbc_snb_datagen.
2. Import data to Nebula Graph by using the Importer.
3. Performance testing by using K6 with the XK6-Nebula plug-in.

For detailed usage instructions, see Nebula Bench.

Last update: September 2, 2021

20. Appendix

20.1 Nebula Graph 2.5.0 release notes

20.1.1 Feature

- Support management of session. #280
- Support terminate the slow queries, know issue: there is a delay in querying and terminating the query due to the implementation. #1152
- Enhance the ability to extract the indices from expressions for the `LOOKUP` statement. #1188
- Supports configuring machine memory watermarks to alleviate OOM issues to some extent. 1067
- Support filter the edges in the `FindPath` statement. #1091
- Support return structure of a graph without properties in the `Subgraph` statement.#1134
- Improve the usage of the `timestamp` function. #515
- Support for querying the version of each service. #944
- `Index` and `TTL` can be supported together. #382
- Support the creation of full-text indexes on specified properties. #460
- Support make comment when create space or schema. #895
- Support for full-text index rebuild. #1123

20.1.2 Enhancement

- The Listener interface is optimized to support full data acquisition. #465, #484
- The leader table of the meta is reorganized. #439
- Add a DiskManager to check disk capacity. #461
- Improve heartbeat of raft to avoid leader change. #438
- Support concurrently go/fetch/lookup in storage. #503
- Enhanced for the `EXISTS` function to the `MAP`. #973
- Enforce the use of aggregate functions, such as `COUNT(v)+AVG(v)`. #968

20.1.3 Bug fix

- Fixed multiple statement execution problems caused by permissions. #1165
- Fixed unwinding causing no results. #1018
- Fixed crash problems caused by aggregation functions in some scenarios. #1015
- Fixed index matching problems with `OR` expressions. #1005
- Fixed case sensitivity of functions. #927
- Fixed issue where query index creation information was not checked for Tag/Edge type. #933
- Fixed a bug in the `Substring` function. #491
- Fixed meta not returning leader change correctly. #423
- Fixed an issue with `LIMIT`, `ORDER`, `GROUP` statements using variables. #1314
- Fixed issue with the `db_dump` tool printing VID of the `int` type. #533
- Fixed the issue that `FAILE` is still displayed after the `Balance` task is recovered. #528

20.1.4 Changes & Known issues

- A little bit grammar change of Subgraph.

```
# Add the WITH PROP keyword to the output property in 2.5.0.
GET SUBGRAPH WITH PROP FROM <vids>

# The original syntax will only output the graph structure without properties.
GET SUBGRAPH FROM <vids>#
```

- We must use the symbol `$-.` in `ORDER BY`. But in earlier releases, there is no need.

```
# We must use the symbol `$-.` in `ORDER BY` in 2.5.0.
nebula> LOOKUP ON player \
        YIELD player.age As playerage \
        | GROUP BY $-.playerage \
        YIELD $-.playerage as age, count(*) AS number \
        | ORDER BY $-.number DESC, $-.age DESC;

# There is no need to use the symbol `$-.` in earlier releases.
nebula> LOOKUP ON player \
        YIELD player.age As playerage \
        | GROUP BY $-.playerage \
        YIELD $-.playerage as age, count(*) AS number \
        | ORDER BY number DESC, age DESC;
```

For the known bug/issue in 2.5.0, see issues.

Last update: September 2, 2021

20.2 FAQ

This topic lists the frequently asked questions for using Nebula Graph 2.5.0. 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 Nebula Graph forum or submit an issue on GitHub issue.

20.2.1 About manual updates

"Why is the behavior in the manual not consistent with the system?"

Nebula Graph 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:
>
> 1. 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.

20.2.2 About legacy version compatibility

> **`X` version compatibility**
>
> Neubla Graph 2.5.0 is **not compatible** with Nebula Graph 1.x nor 2.0-RC in both data formats and RPC-protocols, and **vice versa**. To upgrade data formats, see Upgrade Nebula Graph to v2.0.0. Users must upgrade all clients.

> **`Y` version compatibility**
>
> Data formats of Neubla Graph 2.5.0 are compatible with Nebula Graph 2.0, while their clients are incompatible.

20.2.3 About executions

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.

Nebula Graph 2.5.0 allows dangling edges. And there is no `MERGE` 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.

"How to resolve `[ERROR (-1005)]: Used memory hits the high watermark(0.800000) of total system memory.`?"

The reason for this error may be that `system_memory_high_watermark_ratio` specifies the trigger threshold of the memory high watermark alarm mechanism. The default value is `0.8`. If the system memory usage is higher than this value, an alarm mechanism will be triggered, and Nebula Graph will stop querying.

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`.

> **Note**
>
> Only the Graph service supports `system_memory_high_watermark_ratio`, while the Storage and Meta services do not.

"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 `GO` 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.

"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.

"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             |
+------------------+
| basketballplayer |
+------------------+
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.

"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.

"Is stopping or killing slow queries supported?"

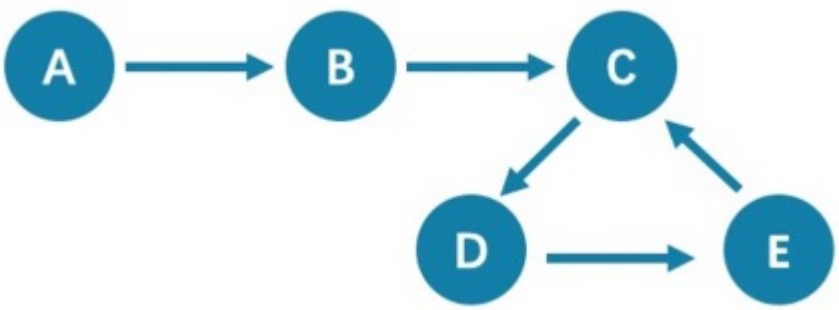
Yes. For more information, see Kill query.

"Why are the query results different when using `GO` and `MATCH` to execute the same semantic query?"

Using different types of paths may cause different query results.

- `GO` statements use `walk`. Both vertices and edges can be repeatedly visited in graph traversal.
- `MATCH` statements are compatible with openCypher and use `trail`. 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->D`), because the edge `C->D` can be visited repeatedly. However, the `MATCH` statement returns empty, because edges cannot be visited repeatedly.

Therefore, using `GO` and `MATCH` to execute the same semantic query may cause different query results.

For more information, see Wikipedia.

"How to resolve `[ERROR (-7)]: 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.

"How to count the vertices/edges number of each tag/edge type?"

See show-stats.

"How to get all the vertices/edge of each tag/edge type?"

1. Create and rebuild the index.

```
> CREATE TAG INDEX i_player ON player();
> REBUILD TAG INDEX 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`.

"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:

```
> GO 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 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.

"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.

"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.

"How to resolve the error `edge conflict` or `vertex conflict`?"

Nebula Graph 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.

"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.

"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-storaged.conf` file. This configuration is measured in milliseconds (ms). The default value is 60000ms.

"How to resolve the error `MetaClient.cpp:65] Heartbeat failed, status:Wrong cluster!` in `nebula-storaged.INFO`, or `HBProcessor.cpp:54] Reject wrong cluster host "x.x.x.x":9771!` in `nebula-metad.INFO`?

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.

Can non-English characters be used as identifiers, such as the names of graph spaces, tags, edge types, properties, and indexes?

No.

The names of graph spaces, tags, edge types, properties, and indexes must use English letters, numbers, or underlines. Non-English characters are not currently supported.

Meanwhile, the above identifiers are case-sensitive and cannot use Keywords and reserved words.

"How to get the out-degree/the in-degree of a vertex with a given name"?

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.

```
nebula > MATCH (s)-[e]->() WHERE id(s) == "given" RETURN count(e); #Out-degree
nebula > MATCH (s)<-[e]-() WHERE id(s) == "given" RETURN count(e); #In-degree
```

"How to quickly get the out-degree and in-degree of all vertices?"

There is no such command.

You can use Nebula Algorithm.

"How to resolve `[ERROR (-1005)]: 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.

20.2.4 About operation and maintenance

"The log files are too large. How to recycle the logs?"

By default, the logs of Nebula Graph are stored in `/usr/local/nebula/logs/`. The INFO level log files are `nebula-graphd.INFO, nebula-storaged.INFO, nebula-metad.INFO`. If an alarm or error occurs, the suffixes are modified as `.WARNING` or `.ERROR`.

Nebula Graph uses glog to print logs. `glog` cannot recycle the outdated files. You can use crontab to delete them by yourself. For more information, see `Glog should delete old log files automatically`.

"How to check the Nebula Graph 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 you install Nebula Graph by compiling the source code

  a. Use the `./<binary_name> --version` command to get the Git commit IDs of the Nebula Graph binary files. For example:

  ```
  $ ./nebula-graphd --version
  nebula-graphd version Git: ab4f683, Build Time: Mar 24 2021 02:17:30
  ```

  b. Search for the commit ID on the GitHub commits page. Check the commit time.

  c. Compare the commit time with the Release timeof Nebula Graph versions to find out the version of the Nebula Graph services.

- If you deploy Nebula Graph with Docker Compose

  Check the version of Nebula Graph 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 Nebula Graph with RPM/DEB package

  Run `rpm -qa |grep nebula` to check the version of Nebula Graph.

"How to scale out or scale in?"

Nebula Graph 2.5.0 does not provide any commands or tools to support automatic scale out/in. You can refer to the following steps:

1. 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.
2. Scale in graphd: Remove the IP of the graphd process from the code in the client. Close this graphd process.
3. 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.
4. Scale in storaged: (The number of replicas must be greater than 1) See Balance remove command. After the command is finished, stop this storaged process.
5. Scale out storaged: (The number of replicas must be greater than 1) 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 start the new storaged process.

You also need to run Balance Data and Balance leader after scaling in/out storaged.

"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.

20.2.5 About connections

"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 Nebula Graph services:

Service	Port
Meta	9559, 9560, 19559, 19560
Graph	9669, 19669, 19670
Storage	9777 ~ 9780, 19779, 19780

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 those eco-tools, see the corresponding document.

"How to test whether a port is open or closed?"

You can use telnet as follows to check for port status.

```
telnet <ip> <port>
```

> **Note**
>
> If you cannot use the telnet command, check if telnet is installed or enabled on your host.

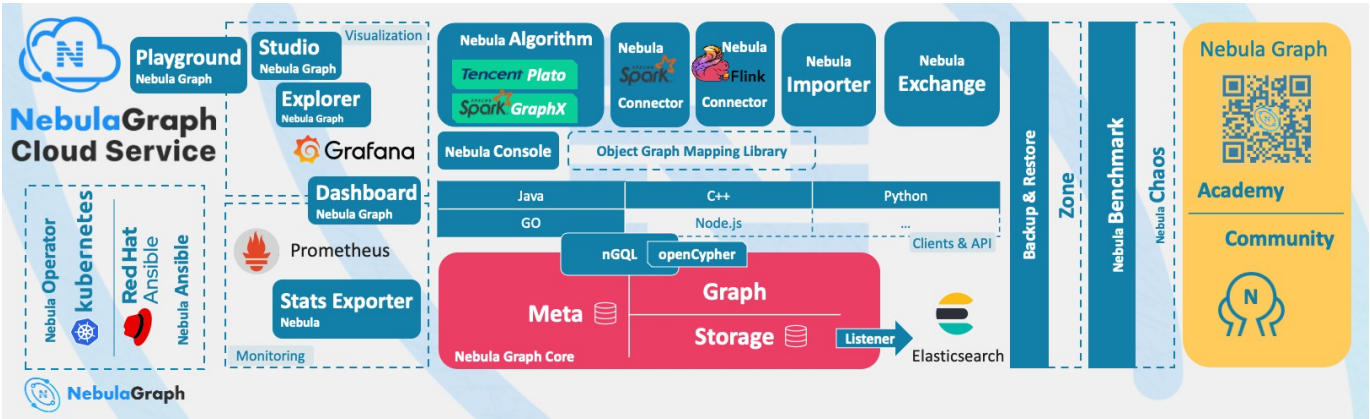
For example:

```
// 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: November 3, 2021

20.3 Ecosystem tools overview



Compatibility

The core release number naming rule is `X.Y.Z`, which means `Major version X`, `Medium version Y`, and `Minor version Z`. The upgrade requirements for the client are:

- Upgrade the core from `X.Y.Z1` to `X.Y.Z2`: It means that the core is fully forward compatible and is usually used for bugfixes. It is recommended to upgrade the minor version of the core as soon as possible. At this time, the client can stay **not upgraded**.
- Upgrade the core from `X.Y1.*` to `X.Y2.*`: It means that there is some incompatibility of API, syntax, and return value. It is usually used to add functions, improve performance, and optimize code. The client needs to be upgraded to `X.Y2.*`.
- Upgrade the core from `X1.*.*` to `X2.*.*`: It means that there is a major incompatibility in storage formats, API, syntax, etc. You need to use tools to upgrade the core data. The client must be upgraded.
- The default core and client do not support downgrade: You cannot downgrade from `X.Y.Z2` to `X.Y.Z1`.
- The release cycle of a `Y` version is about 6 months, and its maintenance and support cycle is 6 months.
- The version released at the beginning of the year is usually named `X.0.0`, and in the middle of the year, it is named `X.5.0`.
- The file name contains `RC` to indicate an unofficial version (`Release Candidate`) that is only used for preview. Its maintenance period is only until the next RC or official version is released. Its client, data compatibility, etc. are not guaranteed.
- The files with `nightly`, `SNAPSHOT`, or date are the nightly versions. There is no quality assurance and maintenance period.

Compatibility

All ecosystem tools of 1.x did not support Nebula Graph 2.x core.

20.3.1 Nebula Graph Studio

Nebula Graph Studio (Studio for short) is a graph database visualization tool that can be accessed through the Web. It can be used with Nebula Graph DBMS to provide one-stop services such as composition, data import, writing nGQL queries, and graph exploration. For details, see What is Nebula Graph Studio.

> **Note**
>
> The release of the Studio is independent of Nebula Graph core, and its naming method is also not the same as the core naming rules. The compatible relationship between them is as follows.

Nebula Graph	Studio (commit id)
2.5.0	3.0.0(9e2a120)

20.3.2 Nebula Dashboard

Nebula Dashboard (Dashboard for short) is a visualization tool for monitoring the status of machines and services in the Nebula Graph cluster. For details, see What is Nebula Dashboard.

Nebula Graph version	Dashboard version (commit id)
2.5.0	1.0.1(49ab1bc)

20.3.3 Nebula Explorer

Nebula Explorer (Explorer for short) is a graph exploration visualization tool that can be accessed through the Web. It is used with the Nebula Graph core to visualize interaction with graph data. Users can quickly become map experts, even without experience in map data manipulation. For details, see What is Nebula Explorer.

Nebula Graph version	Explorer version (commit id)
2.5.0	1.0.0(3b82142)

20.3.4 Nebula Exchange

Nebula Exchange (Exchange for short) is an Apache Spark&trade application for batch migration of data in a cluster to Nebula Graph 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 Nebula Exchange.

Nebula Graph version	Exchange version (commit id)
2.5.0	2.5.0(3c0f4c6)

20.3.5 Nebula Importer

Nebula Importer (Importer for short) is a CSV file import tool for Nebula Graph. The Importer can read the local CSV file, and then import the data into the Nebula Graph database. For details, see What is Nebula Importer.

Nebula Graph version	Importer version(commit id)
2.5.0	2.5.0(5c7417d)

20.3.6 Nebula Spark Connector

Nebula Spark Connector is a Spark connector that provides the ability to read and write Nebula Graph data in the Spark standard format. Nebula Spark Connector consists of two parts, Reader and Writer. For details, see What is Nebula Spark Connector.

Nebula Graph version	Spark Connector version (commit id)
2.5.0	2.5.0(3c0f4c6)

20.3.7 Nebula Flink Connector

Nebula Flink Connector is a connector that helps Flink users quickly access Nebula Graph. It supports reading data from the Nebula Graph database or writing data read from other external data sources to the Nebula Graph database. For details, see What is Nebula Flink Connector.

Nebula Graph version	Flink Connector version (commit id)
2.5.0	2.5.0(49b8f3d)

20.3.8 Nebula Algorithm

Nebula Algorithm (Algorithm for short) is a Spark application based on GraphX, which uses a complete algorithm tool to analyze data in the Nebula Graph 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 Nebula Algorithm.

Nebula Graph version	Algorithm version (commit id)
2.5.0	2.5.0(3c0f4c6)

20.3.9 Nebula Console

Nebula Console is the native CLI client of Nebula Graph. For how to use it, see Connect Nebula Graph.

Nebula Graph version	Console version (commit id)
2.5.0	2.5.0(3ce5151)

20.3.10 Nebula Docker Compose

Docker Compose can quickly deploy Nebula Graph clusters. For how to use it, please refer to Docker Compose Deployment Nebula Graph.

Nebula Graph version	Docker Compose version (commit id)
2.5.0	2.5.0(d42231f)

20.3.11 Nebula Bench

Nebula Bench is used to test the baseline performance data of Nebula Graph. It uses the standard data set of LDBC v0.3.3.

Nebula Graph version	Nebula Bench version (commit id)
2.5.0	1.0.0(661f871)

20.3.12 API, SDK

> **Compatibility**
>
> Select the latest version of `X.Y.*` which is the same as the core version.

Nebula Graph version	Language (commit id)
2.5.0	C++(00e2625)
2.5.0	Go(8a1495a)
2.5.0	Python(98e08e4)
2.5.0	Java Client(0fbc3c6)

20.3.13 Not Released

- API
  - Rust Client
  - Node.js Client
  - HTTP Client
  - [Object Graph Mapping Library (OGM, or ORM)] Java, Python (TODO: in design)
- Monitoring
  - Promethus connector
  - [Graph Computing] (TODO: in coding)
- Test
  - Chaos Test
- Backup&Restore

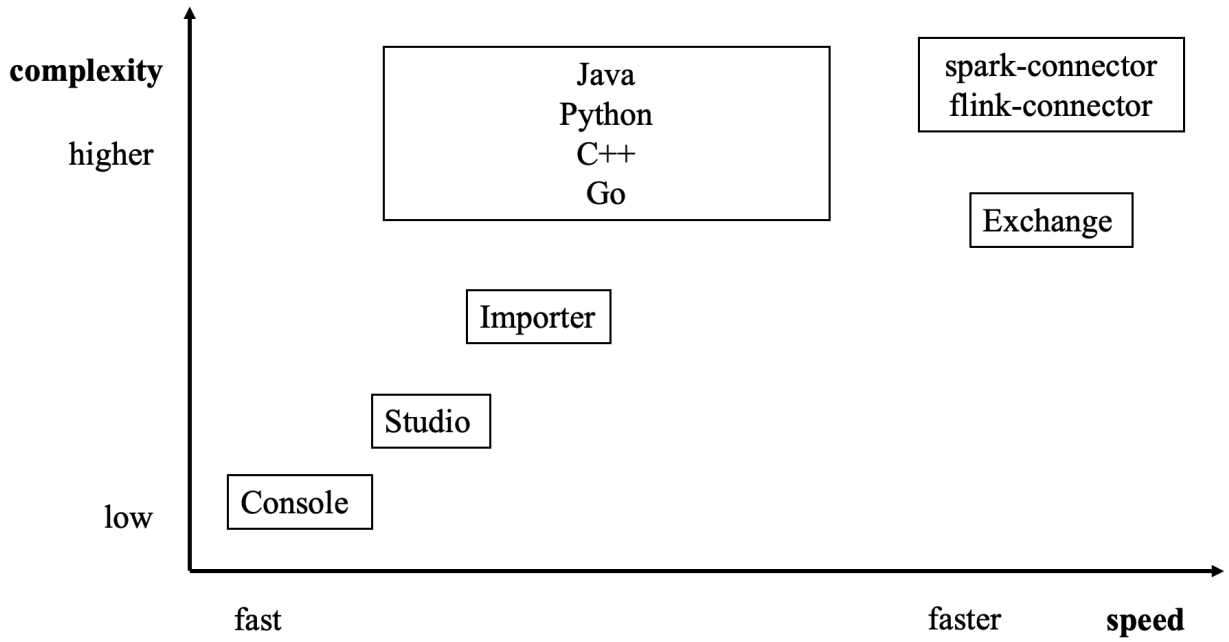
Last update: September 24, 2021

20.4 Import tools

There are many ways to write Nebula Graph 2.5.0:

- 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.
- Import with Exchange: This method imports from various distribution sources, such as Neo4j, Hive, MySQL, etc., which requires a Spark cluster.
- Import with Spark-connector/Flink-connector: This method has corresponding components (Spark/Flink) and writes a small amount of code.
- 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:



Last update: September 2, 2021

20.5 How to Contribute

20.5.1 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)

What is CLA?

Here is the vesoft inc. Contributor License Agreement.

Click the **Sign in with GitHub to agree** button to sign the CLA.

If you have any questions, send an email to `info@vesoft.com`.

20.5.2 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 modify a single document only.

20.5.3 Batch modify or add files

This method applies to contribute codes, modify multiple documents in batches, or add new documents.

20.5.4 Step 1: Fork in the github.com

The Nebula Graph project has many repositories. Take the nebula-graph repository for example:

1. Visit https://github.com/vesoft-inc/nebula.
2. Click the `Fork` button to establish an online fork.

20.5.5 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-graph.git
# or: git clone git@github.com:$user/nebula-graph.git

cd $working_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.
# The correct format is:
```

```
# origin    git@github.com:$(user)/nebula-graph.git (fetch)
# origin    git@github.com:$(user)/nebula-graph.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 Nebula Graph pre-commit hook into the `.git` directory.

   This hook checks the commits for formatting, building, doc generation, etc.

```
cd $working_dir/nebula-graph/.git/hooks
ln -s $working_dir/nebula-graph/.linters/cpp/hooks/pre-commit.sh .
```

   Sometimes, the pre-commit hook cannot be executed. You have to execute it manually.

```
cd $working_dir/nebula-graph/.git/hooks
chmod +x pre-commit
```

20.5.6 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
> ```

20.5.7 Step 4: Develop

- Code style

Nebula Graph 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 Nebula Graph by compiling the source code.

  > **Note**
  >
  > Make sure you have enabled the building of unit tests by setting `-DENABLE_TESTING=ON`.

- Run tests

  In the root directory of `nebula`, run the following command:

  ```
  cd nebula/build
  ctest -j$(nproc)
  ```

20.5.8 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.

20.5.9 Step 6: Commit

Commit your changes.

```
git commit -a
```

Users can use the command `--amend` to re-edit the previous code.

20.5.10 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
```

20.5.11 Step 8: Create a Pull Request

1. Visit your fork at `https://github.com/$user/nebula-graph` (replace `$user` here).
2. Click the `Compare & pull request` button next to your `myfeature` branch.

20.5.12 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.

20.5.13 Add test cases

For detailed methods, see How to add test cases.

20.5.14 Donation

Step 1: Confirm the project donation

Contact the official Nebula Graph staff via email, WeChat, Slack, etc. to confirm the donation project. The project will be donated to the Nebula Contrib organization.

Email address: info@vesoft.com

WeChat: NebulaGraphbot

Slack: Join Slack

Step 2: Get the information of the project recipient

The Nebula Graph official staff will give the recipient ID of the Nebula 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 Nebula 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: September 2, 2021

https://docs.nebula-graph.io/2.5.0