Kyuubi

Release 1.7.0

Apache Kyuubi Community

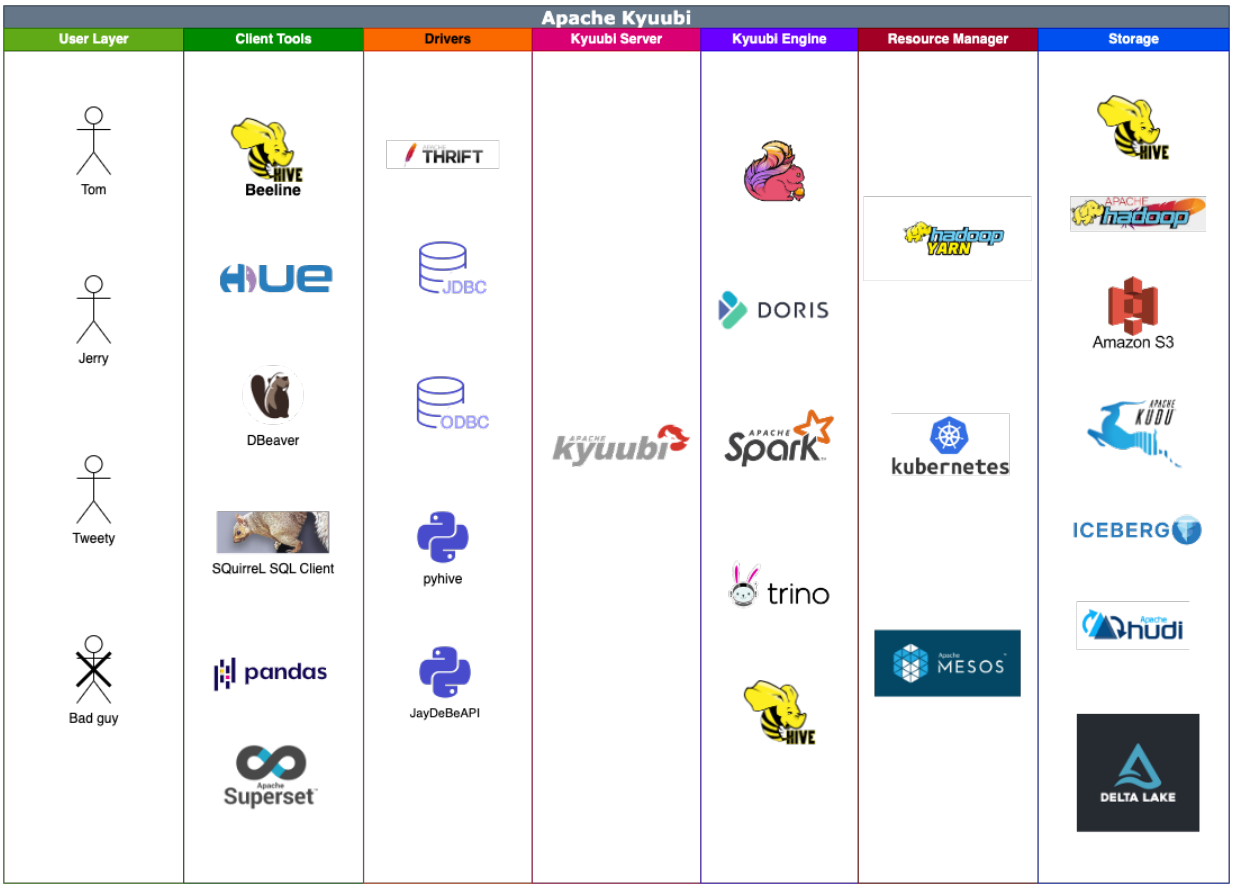
Mar 07, 2023

ADMIN GUIDE

Apache Kyuubi™ is a distributed and multi-tenant gateway to provide serverless SQL on Data Warehouses and Lakehouses.

Kyuubi builds distributed SQL query engines on top of various kinds of modern computing frameworks, e.g., Apache Spark, Flink, Doris, Hive, and Trino, etc, to query massive datasets distributed over fleets of machines from heterogeneous data sources.

The Kyuubi Server lane of the below swimlane divides our prospective users into end users and administrators. On the one hand, it hides the technical details of computing and storage from the end users. Thus, they can focus on their business and data with familiar tools. On the other hand, it hides the complexity of business logic from the administrators. Therefore, they can upgrade components on the server side with zero maintenance downtime, optimize workloads with a clear view of what end users are doing, ensure authentication, authorization, and auditing for both cluster and data security, and so forth.



In general, the complete ecosystem of Kyuubi falls into the hierarchies shown in the above figure, with each layer loosely coupled to the other. It's a child's play to combine some of the components above to build a modern data stack. For example, you can use Kyuubi, Spark and Iceberg to build and manage Data Lakehouse with pure SQL for both data processing, e.g. ETL, and online analytics processing(OLAP), e.g. BI. All workloads can be done on one platform, using one copy of data, with one SQL interface.

CHAPTER

ONE

A UNIFIED GATEWAY

The Server module plays the role of a unified gateway. The server enables simplified, secure access to any cluster resource through an entry point to deploy different workloads for end(remote) users. Behind this single entry, administrators have a single point for configuration, security, and control of remote access to clusters. And end users have an improved experience with seamless data processing with any the Kyuubi engine they need.

1.1 Application Programming Interface

End users can use the application programming interface listed below for connectivity and interoperation between supported clients and a Kyuubi server. The current implementations are:

- **Hive Thrift Protocol**
  - A HiveServer2-compatible interface that allows end users to use a thrift client(cross-language support, both tcp and http), a Java Database Connectivity(JDBC) interface over thrift, or an Open Database Connectivity (ODBC) interface over a JDBC-to-ODBC bridge to communicate with Kyuubi.
- **RESTful APIs**
  - It provides system management APIs, including engines, sessions, operations, and miscellaneous ones.
  - It provides methods that allow clients to submit SQL queries and receive the query results, submit metadata requests and receive metadata results.
  - It enables easy submission of self-contained applications for batch processing, such as Spark jobs.
- **MySQL Protocol**
  - A MySQL-compatible interface that allows end users to use MySQL Connectors, such as Connector/J, to communicate with Kyuubi.
- **We've planned to add more**
  - Please join our mailing lists if you have any ideas or questions.

1.2 Multi-tenancy

Kyuubi supports the end-to-end multi-tenancy. On the control plane, the Kyuubi server provides a centralized authentication layer to reduce the risk of data and resource breaches. It supports various protocols, such as LDAP and Kerberos, for securing networking between clients and servers. On the data plane, the Kyuubi engines use the same trusted client identities to instantiate themselves. The resource acquirement and data and metadata access all happen within their own engine. Thus, cluster managers and storage providers can easily guarantee data and resource security. Besides, Kyuubi also provides engine authorization extensions to optimize the data security model to fine-grained row/column level. Please see the security page for more information.

1.3 High Availability

Kyuubi is designed with High availability (HA), ensuring it operates continuously without failure for a designated period. HA works to provide Kyuubi that meets an agreed-upon operational performance level.

- **Load balancing**
  - It becomes necessary for Kyuubi in a real-world production environment to ensure high availability because of multi-tenant access.
  - It effectively prevents single point of failures.
  - It helps achieve zero downtime for planned system maintenance
- **Failure detectability**
  - Failures and system load of kyuubi server and engines are visible via metrics, logs, and so forth.

CHAPTER

TWO

SERVERLESS SQL AND MORE

Serverless SQL on Lakehouses makes it easier for end users to gain insight from the data universe and optimize data pipelines. It enables:

- The same user experience as an RDBMS using familiar SQL for various workloads.
- Extensive and secure data access capability across diverse data sources.
- High performance on large volumes of data with scalable computing resources.

Besides, Kyuubi also supports submissions of code snippets and self-contained applications serverlessly for more advanced usage.

2.1 Ease of Use

End users could have an optimized experience exploring their data universe in a serverless way, using either JDBC + SQL or REST + code. For most scenarios, the superpower of corresponding engines, such as Spark, and Flink, is no longer necessary. That is, most work related to deployment, runtime optimization, etc., should be done by professionals on the Kyuubi server side. It is suitable for the following scenarios:

- **Basic discovery and exploration**
  - Quickly reason about the data in various formats (Parquet, CSV, JSON, text) in your data lake in cloud storage or an on-prem HDFS cluster.
- **Lakehouse formation and analytics**
  - Easily build an ACID table storage layer via Hudi, Iceberg, or/and Delta Lake.
- **Logical data warehouse**
  - Provide a relational abstraction on top of disparate data without ETL jobs, from collecting to connecting.

2.2 Run Anywhere at Any Scale

Most of the Kyuubi engine types have a distributed backend or can schedule distributed tasks at runtime. They can process data on single-node machines or clusters, such as YARN and Kubernetes. Besides, the Kyuubi server also supports running on bare metal or in a docker.

2.3 High Performance

Query performance is one of the critical factors in implementing Serverless SQL. Implementing serviceability on state-of-the-art query engines for bigdata lays the foundation for us to achieve this goal:

- State-of-the-art query engines
- Multiple application for high throughput
- Sharable execution runtime for low latency
- Server-side global and continuous optimization
- Auxiliary performance plugins, such as Z-Ordering, Query Optimizer, and so on

Another goal of Serverless SQL is to make end users need not or rarely care about tricky performance optimization issues.

CHAPTER

THREE

WHAT'S NEXT

3.1 Quick Start

Note: In this section, you will learn how to setup and interact with kyuubi quickly.

3.1.1 Getting Started

Note: This page covers how to start with kyuubi quickly on you laptop in about 3~5 minutes.

Requirements

For quick start deployment, we need to prepare the following stuffs:

- A **client** that connects and submits queries to the server. Here, we use the kyuubi beeline for demonstration.
- A **server** that serves clients and manages engines.
- An **engine** that is used to instantiate query execution environments. Here we use Spark for demonstration.

These essential components are JVM-based applications. So, the JRE needs to be pre-installed and the *JAVA_HOME* is correctly set to each component.

Component	Role	Version	Remarks
Java	JRE	8/11	Officially released against JDK8
Kyuubi	Gateway Engine lib Beeline	1.7.0	• Kyuubi Server • Kyuubi Engine • Kyuubi Hive Beeline
Spark	Engine	>=3.0.0	A Spark distribution
Flink	Engine	>=1.14.0	A Flink distribution
Trino	Engine	>=363	A Trino cluster
Doris	Engine	N/A	A Doris cluster
Hive	Engine Metastore	• 3.1.x • N/A	• A Hive distribution • An optional and external metadata store, whose version is decided by engines
Zookeeper	HA	>=3.4.x	
Disk	Storage	N/A	N/A

The other internal or external parts listed in the above sheet can be used individually or all together. For example, you can use Kyuubi, Spark and Flink to build a streaming data warehouse. And then, you can use Zookeeper to enable the load balancing for high availability. The data could be stored in Hive, Apache Iceberg, or other DBMSs.

In what follows, we will only use Kyuubi and Spark.

Installation

Note: This following instructions are based on binary releases. If you start with source releases, please refer to the page for building kyuubi.

Install Kyuubi

The official releases, binary- and source-, are archived on the download page. Please download the most recent stable release to start.

To install Kyuubi, you need to unpack the tarball. For example,

```
$ tar zxf apache-kyuubi-1.7.0-bin.tgz
```

```
├── LICENSE
├── NOTICE
├── RELEASE
├── beeline-jars
├── bin
├── conf
```

(continues on next page)

(continued from previous page)

```
|   ├── kyuubi-defaults.conf.template
|   ├── kyuubi-env.sh.template
|   └── log4j2.properties.template
├── docker
|   ├── Dockerfile
|   ├── helm
|   ├── kyuubi-configmap.yaml
|   ├── kyuubi-deployment.yaml
|   ├── kyuubi-pod.yaml
|   └── kyuubi-service.yaml
├── externals
|   └── engines
├── jars
├── licenses
├── logs
├── pid
└── work
```

From top to bottom are:

- LICENSE: the APACHE LICENSE, VERSION 2.0 we claim to obey.
- RELEASE: the build information of this package.
- NOTICE: the notice made by Apache Kyuubi Community about its project and dependencies.
- bin: the entry of the Kyuubi server with *kyuubi* as the startup script.
- conf: all the defaults used by Kyuubi Server itself or creating a session with engines.
- externals - engines: contains all kinds of SQL engines that we support
- licenses: a bunch of licenses included.
- jars: packages needed by the Kyuubi server.
- logs: where the logs of the Kyuubi server locates.
- pid: stores the PID file of the Kyuubi server instance.
- work: the root of the working directories of all the forked sub-processes, a.k.a. SQL engines.

Install Spark

The official releases, binary- and source-, are archived on the spark download page. Please download the most recent stable release to start.

Note: Currently, Kyuubi is compiled and pre-built against Spark 3 and Scala 2.12 You will probably meet runtime exceptions if you use Spark 2 or Spark with unsupported Scala versions.

To install Spark, you need to unpack the tarball. For example,

```
$ tar zxf spark-3.3.2-bin-hadoop3.tgz
```

Configuration

The *kyuubi-env.sh* file is used to set system environment variables to the kyuubi server process and engine processes it creates.

The *kyuubi-defaults.conf* file is used to set system properties to the kyuubi server process and engine processes it creates.

Each file has a template lays in *conf* directory for your information. The following are examples of the parameters necessary for a quick start with Spark.

- **JAVA_HOME**

```
$ echo 'export JAVA_HOME=/path/to/java' >> conf/kyuubi-env.sh
```

- **SPARK_HOME**

```
$ echo 'export SPARK_HOME=/path/to/spark' >> conf/kyuubi-env.sh
```

Start Kyuubi

```
$ bin/kyuubi start
```

If script above runs successfully, it will store the *PID* of the server instance into *pid/kyuubi-<username>-org.apache.kyuubi.server.KyuubiServer.pid*. And you are able to get the JDBC connection URL from the log file - *logs/kyuubi-<username>-org.apache.kyuubi.server.KyuubiServer-<hostname>.out*.

For example,

> Starting and exposing JDBC connection at: jdbc:hive2://localhost:10009/

If something goes wrong, you shall be able to find some clues in the log file too.

Note: Alternatively, it can run in the foreground, with the logs and other output written to stdout/stderr. Both streams should be captured if using a supervision system like *supervisord*.

```
bin/kyuubi run
```

Operate Clients

Kyuubi delivers a beeline client, enabling a similar experience to Apache Hive use cases.

Open Connections

Replace the *host* and *port* with the actual ones you've got in the step of server startup for the following JDBC URL. The case below open a session for user named *apache*.

```
$ bin/beeline -u 'jdbc:hive2://localhost:10009/' -n apache
```

Note: Use *–help* to display the usage guide for the beeline tool.

```
$ bin/beeline --help
```

Execute Statements

After successfully connected with the server, you can run sql queries in the beeline console. For instance,

```
> SHOW DATABASES;
```

You will see a wall of operation logs, and a result table in the beeline console.

```
omitted logs
+------------+
| namespace  |
+------------+
| default    |
+------------+
1 row selected (0.2 seconds)
```

Start Engines

Engines are launched by the server automatically without end users' attention.

If you use the same user in the above case to create another connection, the engine will be reused. You may notice that the time cost for connection here is much shorter than the last round.

If you use a different user to create a new connection, another engine will be started.

```
$ bin/beeline -u 'jdbc:hive2://localhost:10009/' -n kentyao
```

This may change depending on the engine share level you set.

Close Connections

Close the session between beeline and Kyuubi server by executing *!quit*, for example,

```
> !quit
Closing: 0: jdbc:hive2://localhost:10009/
```

Stop Engines

Engines are stop by the server automatically according engine lifecycle without end users' attention. Terminations of connections do not necessarily mean terminations of engines. It depends on both the engine share level and engine lifecycle.

Stop Kyuubi

Stop Kyuubi by running the following in the *$KYUUBI_HOME* directory:

```
$ bin/kyuubi stop
```

And then, you will see the Kyuubi server waving goodbye to you.

The Kyuubi server will be stopped immediately while the engine will still be alive for a while.

If you start Kyuubi again before the engine terminates itself, it will reconnect to the newly created one.

3.1.2 Getting Started With Kyuubi on Kubernetes

Running Kyuubi with Helm

Helm is the package manager for Kubernetes, it can be used to find, share, and use software built for Kubernetes.

Install Helm

Please go to Installing Helm page to get and install an appropriate release version for yourself.

Get Kyuubi Started

Install the chart

```
helm install kyuubi ${KYUUBI_HOME}/charts/kyuubi -n kyuubi --create-namespace
```

It will print release info with notes, including the ways to get Kyuubi accessed within Kubernetes cluster and exposed externally depending on the configuration provided.

```
NAME: kyuubi
LAST DEPLOYED: Sat Feb 11 20:59:00 2023
NAMESPACE: kyuubi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The chart has been installed!

In order to check the release status, use:
  helm status kyuubi -n kyuubi
    or for more detailed info
  helm get all kyuubi -n kyuubi

************************
******* Services *******
************************
THRIFT_BINARY:
- To access kyuubi-thrift-binary service within the cluster, use the following URL:
    kyuubi-thrift-binary.kyuubi.svc.cluster.local
```

(continues on next page)

(continued from previous page)

```
- To access kyuubi-thrift-binary service from outside the cluster for debugging, run the␣
↪following command:
    kubectl port-forward svc/kyuubi-thrift-binary 10009:10009 -n kyuubi
  and use 127.0.0.1:10009
```

Uninstall the chart

```
helm uninstall kyuubi -n kyuubi
```

Configure chart release

Specify configuration properties using `--set` flag. For example, to install the chart with `replicaCount` set to 1, use the following command:

```
helm install kyuubi ${KYUUBI_HOME}/charts/kyuubi -n kyuubi --create-namespace --set␣
↪replicaCount=1
```

Also, custom values file can be used to override default property values. For example, create `myvalues.yaml` to specify `replicaCount` and `resources`:

```
replicaCount: 1

resources:
  requests:
    cpu: 2
    memory: 4Gi
  limits:
    cpu: 4
    memory: 10Gi
```

and use it to override default chart values with `-f` flag:

```
helm install kyuubi ${KYUUBI_HOME}/charts/kyuubi -n kyuubi --create-namespace -f␣
↪myvalues.yaml
```

Access logs

List all pods in the release namespace:

```
kubectl get pod -n kyuubi
```

Find Kyuubi pods:

```
NAME                      READY   STATUS    RESTARTS   AGE
kyuubi-5b6d496c98-kbhws   1/1     Running   0          38m
kyuubi-5b6d496c98-lqldk   1/1     Running   0          38m
```

Then, use pod name to get logs:

```
kubectl logs kyuubi-5b6d496c98-kbhws -n kyuubi
```

3.1.3 Getting Started With Hive JDBC

How to install JDBC driver

Kyuubi JDBC driver is fully compatible with the 2.3.* version of hive JDBC driver, so we reuse hive JDBC driver to connect to Kyuubi server.

Add repository to your maven configuration file which may reside in `$MAVEN_HOME/conf/settings.xml`.

```
<repositories>
  <repository>
    <id>central maven repo</id>
    <name>central maven repo https</name>
    <url>https://repo.maven.apache.org/maven2</url>
  </repository>
</repositories>
```

You can add below dependency to your `pom.xml` file in your application.

```
<!-- https://mvnrepository.com/artifact/org.apache.hive/hive-jdbc -->
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-jdbc</artifactId>
    <version>2.3.7</version>
</dependency>
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-common</artifactId>
    <!-- keep consistent with the build hadoop version -->
    <version>2.7.4</version>
</dependency>
```

Use JDBC driver with kerberos

The below java code is using a keytab file to login and connect to Kyuubi server by JDBC.

```
package org.apache.kyuubi.examples;

import java.io.IOException;
import java.security.PrivilegedExceptionAction;
import java.sql.*;

import org.apache.hadoop.security.UserGroupInformation;

public class JDBCTest {

    private static String driverName = "org.apache.hive.jdbc.HiveDriver";
    private static String kyuubiJdbcUrl = "jdbc:hive2://localhost:10009/default;";
```

(continues on next page)

(continued from previous page)

```
    public static void main(String[] args) throws ClassNotFoundException, SQLException {
        String principal = args[0]; // kerberos principal
        String keytab = args[1]; // keytab file location
        Configuration configuration = new Configuration();
        configuration.set(HADOOP_SECURITY_AUTHENTICATION, "kerberos");
        UserGroupInformation.setConfiguration(configuration);
        UserGroupInformation ugi = UserGroupInformation.
↪loginUserFromKeytabAndReturnUGI(principal, keytab);

        Class.forName(driverName);
        Connection conn = ugi.doAs(new PrivilegedExceptionAction<Connection>(){
            public Connection run() throws SQLException {
                return DriverManager.getConnection(kyuubiJdbcUrl);
            }
        });
        Statement st = conn.createStatement();
        ResultSet res = st.executeQuery("show databases");
        while (res.next()) {
            System.out.println(res.getString(1));
        }
        res.close();
        st.close();
        conn.close();
    }
}
```

3.2 Deploying Kyuubi

In this section, you will learn how to deploy Kyuubi against different platforms.

3.2.1 Basics

Deploy Kyuubi On Kubernetes

Requirements

If you want to deploy Kyuubi on Kubernetes, you'd better get a sense of the following things.

- Use Kyuubi official docker image or build Kyuubi docker image
- An active Kubernetes cluster
- Reading About Deploy Kyuubi engines on Kubernetes
- Kubectl
- KubeConfig of the target cluster

Kyuubi Official Docker Image

You can find the official docker image at Apache Kyuubi Docker Hub.

Build Kyuubi Docker Image

You can build custom Docker images from the `${KYUUBI_HOME}/bin/docker-image-tool.sh` contained in the binary package.

Examples:

```
- Build and push image with tag "v1.4.0" to docker.io/myrepo
  $0 -r docker.io/myrepo -t v1.4.0 build
  $0 -r docker.io/myrepo -t v1.4.0 push

- Build and push with tag "v1.4.0" and Spark-3.2.1 as base image to docker.io/myrepo
  $0 -r docker.io/myrepo -t v1.4.0 -b BASE_IMAGE=repo/spark:3.2.1 build
  $0 -r docker.io/myrepo -t v1.4.0 push

- Build and push for multiple archs to docker.io/myrepo
  $0 -r docker.io/myrepo -t v1.4.0 -X build

- Build with Spark placed "/path/spark"
  $0 -s /path/spark build

- Build with Spark Image myrepo/spark:3.1.0
  $0 -S /opt/spark -b BASE_IMAGE=myrepo/spark:3.1.0 build
```

`${KYUUBI_HOME}/bin/docker-image-tool.sh` use `Kyuubi Version` as default docker tag and always build `${repo}/kyuubi:${tag}` image.

The script can also help build external Spark into a Kyuubi image that acts as a client for submitting tasks by `-s ${SPARK_HOME}`.

Of course, if you have an image that contains the Spark binary package, you don't have to copy Spark locally. Make your Spark Image as BASE_IMAGE by using the `-S ${SPARK_HOME_IN_DOCKER}` and `-b BASE_IMAGE=${SPARK_IMAGE}` arguments.

You can use `${KYUUBI_HOME}/bin/docker-image-tool.sh -h` for more parameters.

Deploy

Multiple YAML files are provided under `${KYUUBI_HOME}/docker/` to help you deploy Kyuubi.

You can deploy single-node Kyuubi through `${KYUUBI_HOME}/docker/kyuubi-pod.yaml` or `${KYUUBI_HOME}/docker/kyuubi-deployment.yaml`.

Also, you can use `${KYUUBI_HOME}/docker/kyuubi-service.yaml` to deploy Kyuubi Service.

[Optional] ServiceAccount

According to Kubernetes RBAC, we need to give kyuubi server the corresponding kubernetes privileges for `created/list/delete` engine pods in kubernetes.

You should create your serviceAccount (or reuse account with the appropriate privileges) and set your serviceAccountName for kyuubi pod, which you can find template in `${KYUUBI_HOME}/docker/kyuubi-deployment.yaml` or `${KYUUBI_HOME}/docker/kyuubi-pod.yaml`.

For example, you can create serviceAccount by following command:

```
kubectl create serviceAccount kyuubi -n <your namespace>

kubectl create rolebinding kyuubi-role --role=edit --serviceAccount=<your namespace>
↪:kyuubi --namespace=<your namespace>
```

See more related details in Using RBAC Authorization and Configure Service Accounts for Pods.

Config

You can configure Kyuubi the old-fashioned way by placing kyuubi-default.conf inside the image. Kyuubi do not recommend using this way on Kubernetes.

Kyuubi provide `${KYUUBI_HOME}/docker/kyuubi-configmap.yaml` to build Configmap for Kyuubi.

You can find out how to use it in the comments inside the above file.

If you want to know kyuubi engine on kubernetes configurations, you can refer to Deploy Kyuubi engines on Kubernetes

Connect

If you do not use Service or HostNetwork to get the IP address of the node where Kyuubi deployed. You should connect like:

```
kubectl exec -it kyuubi-example -- /bin/bash
${KYUUBI_HOME}/bin/beeline -u 'jdbc:hive2://localhost:10009'
```

Or you can submit tasks directly through local beeline:

```
${KYUUBI_HOME}/bin/beeline -u 'jdbc:hive2://${hostname}:${port}'
```

As using service nodePort, port means nodePort and hostname means any hostname of kubernetes node.

As using HostNetwork, port means kyuubi containerPort and hostname means hostname of node where Kyuubi deployed.

TODO

Kyuubi will provide other connection methods in the future, like `Ingress`, `Load Balance`.

Integration with Hive Metastore

In this section, you will learn how to configure Kyuubi to interact with Hive Metastore.

- A common Hive metastore server could be set at Kyuubi server side
- Individual Hive metastore servers could be used for end users to set

Requirements

- A running Hive metastore server
  - Hive Metastore Administration
  - Configuring the Hive Metastore for CDH
- A Spark binary distribution built with `-Phive` support
  - Use the built-in one in the Kyuubi distribution
  - Download from Spark official website
  - Build from Spark source, Building With Hive and JDBC Support
- A copy of Hive client configuration

So the whole thing here is to let Spark applications use this copy of Hive configuration to start a Hive metastore client for their own to talk to the Hive metastore server.

Default Behavior

By default, Kyuubi launches Spark SQL engines pointing to a dummy embedded Apache Derby-based metastore for each application, and this metadata can only be seen by one user at a time, e.g.

```
bin/beeline -u 'jdbc:hive2://localhost:10009/' -n kentyao
Connecting to jdbc:hive2://localhost:10009/
Connected to: Spark SQL (version 1.0.0-SNAPSHOT)
Driver: Hive JDBC (version 2.3.7)
Transaction isolation: TRANSACTION_REPEATABLE_READ
Beeline version 2.3.7 by Apache Hive
0: jdbc:hive2://localhost:10009/> show databases;
2020-11-16 23:50:50.388 INFO operation.ExecuteStatement:
           Spark application name: kyuubi_kentyao_spark_2020-11-16T15:50:08.968Z
                 application ID:  local-1605541809797
                 application web UI: http://192.168.1.14:60165
                 master: local[*]
                 deploy mode: client
                 version: 3.0.1
           Start time: 2020-11-16T15:50:09.123Z
           User: kentyao
2020-11-16 23:50:50.404 INFO metastore.HiveMetaStore: 2: get_databases: *
```

(continues on next page)

(continued from previous page)

```
2020-11-16 23:50:50.404 INFO HiveMetaStore.audit: ugi=kentyao        ip=unknown-ip-
↪addr        cmd=get_databases: *
2020-11-16 23:50:50.423 INFO operation.ExecuteStatement: Processing kentyao's␣
↪query[8453e657-c1c4-4391-8406-ab4747a66c45]: RUNNING_STATE -> FINISHED_STATE,␣
↪statement: show databases, time taken: 0.035 seconds
+------------+
| namespace  |
+------------+
| default    |
+------------+
1 row selected (0.122 seconds)
0: jdbc:hive2://localhost:10009/> show tables;
2020-11-16 23:50:52.957 INFO operation.ExecuteStatement:
           Spark application name: kyuubi_kentyao_spark_2020-11-16T15:50:08.968Z
                 application ID:  local-1605541809797
                 application web UI: http://192.168.1.14:60165
                 master: local[*]
                 deploy mode: client
                 version: 3.0.1
           Start time: 2020-11-16T15:50:09.123Z
           User: kentyao
2020-11-16 23:50:52.968 INFO metastore.HiveMetaStore: 2: get_database: default
2020-11-16 23:50:52.968 INFO HiveMetaStore.audit: ugi=kentyao        ip=unknown-ip-
↪addr        cmd=get_database: default
2020-11-16 23:50:52.970 INFO metastore.HiveMetaStore: 2: get_database: default
2020-11-16 23:50:52.970 INFO HiveMetaStore.audit: ugi=kentyao        ip=unknown-ip-
↪addr        cmd=get_database: default
2020-11-16 23:50:52.972 INFO metastore.HiveMetaStore: 2: get_tables: db=default pat=*
2020-11-16 23:50:52.972 INFO HiveMetaStore.audit: ugi=kentyao        ip=unknown-ip-
↪addr        cmd=get_tables: db=default pat=*
2020-11-16 23:50:52.986 INFO operation.ExecuteStatement: Processing kentyao's␣
↪query[ff902582-ba29-433b-b70a-c25ead1353a8]: RUNNING_STATE -> FINISHED_STATE,␣
↪statement: show tables, time taken: 0.03 seconds
+-----------+------------+--------------+
| database  | tableName  | isTemporary  |
+-----------+------------+--------------+
+-----------+------------+--------------+
No rows selected (0.04 seconds)
```

Using this mode for experimental purposes only.

In a real production environment, we always have a communal standalone metadata store, to manage the metadata of persistent relational entities, e.g. databases, tables, columns, partitions, for fast access. Usually, Hive metastore as the de facto.

Related Configurations

These are the basic needs for a Hive metastore client to communicate with the remote Hive Metastore server.

Use remote metastore database or server mode depends on the server-side configuration.

Remote Metastore Database

Remote Metastore Server

Activate Configurations

Via kyuubi-defaults.conf

In `$KYUUBI_HOME/conf/kyuubi-defaults.conf`, all ***Hive primitive configurations***, e.g. `hive.metastore.uris`, and the ***Spark derivatives***, which are prefixed with `spark.hive.` or `spark.hadoop.`, e.g `spark.hive.metastore.uris` or `spark.hadoop.hive.metastore.uris`, will be loaded as Hive primitives by the Hive client inside the Spark application.

Kyuubi will take these configurations as system wide defaults for all applications it launches.

Via hive-site.xml

Place your copy of `hive-site.xml` into `$SPARK_HOME/conf`, every single Spark application will automatically load this config file to its classpath.

This version of configuration has lower priority than those in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`.

Via JDBC Connection URL

We can pass ***Hive primitives*** or ***Spark derivatives*** directly in the JDBC connection URL, e.g.

```
jdbc:hive2://localhost:10009/;#hive.metastore.uris=thrift://localhost:9083
```

This will override the defaults in `$SPARK_HOME/conf/hive-site.xml` and `$KYUUBI_HOME/conf/kyuubi-defaults.conf` for each ***user account***.

With this feature, end users are possible to visit different Hive metastore server instance. Similarly, this works for other services like HDFS, YARN too.

Limitation: As most Hive configurations are final and unmodifiable in Spark at runtime, this only takes effect during instantiating the Spark applications and will be ignored when reusing an existing application. So, keep this in our mind.

!!!THIS WORKS ONLY ONCE!!!

!!!THIS WORKS ONLY ONCE!!!

!!!THIS WORKS ONLY ONCE!!!

Via SET syntax

Most Hive configurations are final and unmodifiable in Spark at runtime, so keep this in our mind.

!!!THIS WON'T WORK!!!

!!!THIS WON'T WORK!!!

!!!THIS WON'T WORK!!!

Version Compatibility

If backward compatibility is guaranteed by Hive versioning, we can always use a lower version Hive metastore client to communicate with the higher version Hive metastore server.

For example, Spark 3.0 was released with a built-in Hive client (2.3.7), so, ideally, the version of server should >= 2.3.x.

If you do have a legacy Hive metastore server that cannot be easily upgraded, and you may face the issue by default like this,

```
Caused by: org.apache.thrift.TApplicationException: Invalid method name: 'get_table_req'
        at org.apache.thrift.TServiceClient.receiveBase(TServiceClient.java:79)
        at org.apache.hadoop.hive.metastore.api.ThriftHiveMetastore$Client.recv_get_
↪table_req(ThriftHiveMetastore.java:1567)
        at org.apache.hadoop.hive.metastore.api.ThriftHiveMetastore$Client.get_table_
↪req(ThriftHiveMetastore.java:1554)
        at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.
↪getTable(HiveMetaStoreClient.java:1350)
        at org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient.
↪getTable(SessionHiveMetaStoreClient.java:127)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
↪java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.
↪invoke(RetryingMetaStoreClient.java:173)
        at com.sun.proxy.$Proxy37.getTable(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
↪java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.hive.metastore.HiveMetaStoreClient$SynchronizedHandler.
↪invoke(HiveMetaStoreClient.java:2336)
        at com.sun.proxy.$Proxy37.getTable(Unknown Source)
        at org.apache.hadoop.hive.ql.metadata.Hive.getTable(Hive.java:1274)
        ... 93 more
```

To prevent this problem, we can use Spark's Interacting with Different Versions of Hive Metastore.

Further Readings

- Hive Wiki
  - Hive Metastore Administration
- Spark Online Documentation
  - Custom Hadoop/Hive Configuration
  - Hive Tables

Kyuubi High Availability Guide

As an enterprise-class ad-hoc SQL query service built on top of Apache Spark, Kyuubi takes high availability (HA) as a major characteristic, aiming to ensure an agreed level of service availability, such as a higher than normal period of uptime.

Running Kyuubi in HA mode is to use groups of computers or containers that support SQL query service on Kyuubi that can be reliably utilized with a minimum amount of down-time. Kyuubi operates by using Apache ZooKeeper to harness redundant service instances in groups that provide continuous service when one or more components fail.

Without HA, if a server crashes, Kyuubi will be unavailable until the crashed server is fixed. With HA, this situation will be remedied by hardware/software faults auto-detecting, and immediately another Kyuubi service instance will be ready to serve without requiring human intervention.

HA Architecture

Currently, Kyuubi supports load balancing to make the whole system highly available.

Load balancing aims to optimize all Kyuubi service unit's usage, maximize throughput, minimize response time, and avoid overload of a single unit. Using multiple Kyuubi service units with load balancing instead of a single unit may increase reliability and availability through redundancy.

Key Benefits

- High concurrency
  - By adding or removing Kyuubi server instances can easily scale up or down to meet the need of client requests.
- Upgrade smoothly
  - Kyuubi server supports stop gracefully. We could delete a `k.i.` but not stop it immediately. In this case, the `k.i.` will not take any new connection request but only operation requests from existing connections. After all connection are released, it stops then.
  - The dependencies of Kyuubi engines are free to change, such as bump up versions, modify configurations, add external jars, relocate to another engine home. Everything will be reloaded during start and stop.

System-side Deployment

When applying HA to Kyuubi deployment, we need to be aware of the below two thing basically,

- `kyuubi.ha.zookeeper.quorum` - the external zookeeper cluster address for deploy a `k.i.`
- `kyuubi.ha.zookeeper.namespace` - the root directory, a.k.a. the ServerSpace for deploy a `k.i.`

For more configurations, please see the HA section of Introduction to the Kyuubi Configurations System

Pseudo mode

When `kyuubi.ha.zookeeper.quorum` is not configured, a `k.i.` will start an embedded zookeeper service and expose the address of itself there. In this pseduo mode, the `k.i.` can be connected by clients through both raw ip address and zk quorum + namespace. But it doesn't have any availability to being highly available.

Production mode

For production deployment purpose, an external zookeeper cluster is required for `kyuubi.ha.zookeeper.quorum`. In this mode, multiple `k.i.`s can be registered to the same ServerSpace configured by `kyuubi.ha.zookeeper.namespace` and serve together.

Client-side Usage

With Kyuubi Hive JDBC Driver or vanilla Hive JDBC Driver, a client can specify service discovery mode in JDBC connection string, i.e. `serviceDiscoveryMode=zooKeeper;` and set `zooKeeperNamespace=kyuubi;`, then it can randomly pick one of the Kyuubi service uris from the specified ZooKeeper addresses in the `/kyuubi` path.

For example,

```
bin/beeline -u 'jdbc:hive2://10.242.189.214:2181/;serviceDiscoveryMode=zooKeeper;
↪zooKeeperNamespace=kyuubi' -n kentyao
```

How to Hot Upgrade Kyuubi Server

Kyuubi supports hot upgrade one of server in a HA cluster which is transparent to users.

- If you have specified a custom port for Kyuubi server

  For example, the Kyuubi server started at host `kyuubi.host` with port `10009`, you can run the following cmd using `bin/kyuubi-ctl`:

  ```
  ./bin/kyuubi-ctl delete server --host "kyuubi.host" --port "10009"
  ```

  Kyuubi server will stop until all session closed, and then you can start a new Kyuubi server.

- If you use a random port for Kyuubi server

  You can just start the new Kyuubi Server, and then run cmd using `bin/kyuubi-ctl`:

  ```
  ./bin/kyuubi-ctl delete server --host "kyuubi.host" --port "${PORT_FPR_OLD_KYUUBI_
  ↪SERVER}"
  ```

  The `${PORT_FPR_OLD_KYUUBI_SERVER}` can be found by:

```
grep "server.KyuubiThriftBinaryFrontendService: Starting and exposing JDBC␣
↪connection at" logs/kyuubi-*.out
```

Note that, you do not need to care when the old Kyuubi server actually stopped since the new coming session are routed to the new Kyuubi server and others.

Kyuubi Migration Guide

Upgrading from Kyuubi 1.6 to 1.7

- In Kyuubi 1.7, `kyuubi.ha.zookeeper.engine.auth.type` does not fallback to `kyuubi.ha.zookeeper.auth.type`.When Kyuubi engine does Kerberos authentication with Zookeeper, user needs to explicitly set `kyuubi.ha.zookeeper.engine.auth.type` to `KERBEROS`.
- Since Kyuubi 1.7, Kyuubi returns engine's information for `GetInfo` request instead of server. To restore the previous behavior, set `kyuubi.server.info.provider` to `SERVER`.
- Since Kyuubi 1.7, Kyuubi session type `SQL` is refactored to `INTERACTIVE`, because Kyuubi supports not only `SQL` session, but also `SCALA` and `PYTHON` sessions. User need to use `INTERACTIVE` sessionType to look up the session event.
- Since Kyuubi 1.7, the REST API of `Open(create) a session` will not contains parameters `user` `password` and `IpAddr`. User and password should be set in `Authorization` of http request if needed.

Upgrading from Kyuubi 1.6.0 to 1.6.1

- Since Kyuubi 1.6.1, `kyuubi.ha.zookeeper.engine.auth.type` does not fallback to `kyuubi.ha.zookeeper.auth.type`.When Kyuubi engine does Kerberos authentication with Zookeeper, user needs to explicitly set `kyuubi.ha.zookeeper.engine.auth.type` to `KERBEROS`.

Upgrading from Kyuubi 1.5 to 1.6

- Kyuubi engine gets Zookeeper principal & keytab from `kyuubi.ha.zookeeper.auth.principal` & `kyuubi.ha.zookeeper.auth.keytab`.`kyuubi.ha.zookeeper.auth.principal` & `kyuubi.ha.zookeeper.auth.keytab` fallback to `kyuubi.kinit.principal` & `kyuubi.kinit.keytab` when not set.Since Kyuubi 1.6, `kyuubi.kinit.principal` & `kyuubi.kinit.keytab` are filtered out from Kyuubi engine's conf for better security.When Kyuubi engine does Kerberos authentication with Zookeeper, user needs to explicitly set `kyuubi.ha.zookeeper.auth.principal` & `kyuubi.ha.zookeeper.auth.keytab`.

3.2.2 Configurations

Introduction to the Kyuubi Configurations System

Kyuubi provides several ways to configure the system and corresponding engines.

Environments

You can configure the environment variables in `$KYUUBI_HOME/conf/kyuubi-env.sh`, e.g, `JAVA_HOME`, then this java runtime will be used both for Kyuubi server instance and the applications it launches. You can also change the variable in the subprocess's env configuration file, e.g.`$SPARK_HOME/conf/spark-env.sh` to use more specific ENV for SQL engine applications. see `$KYUUBI_HOME/conf/kyuubi-env.sh.template` as an example. For the environment variables that only needed to be transferred into engine side, you can set it with a Kyuubi configuration item formatted `kyuubi.engineEnv.VAR_NAME`. For example, with `kyuubi.engineEnv.SPARK_DRIVER_MEMORY=4g`, the environment variable `SPARK_DRIVER_MEMORY` with value `4g` would be transferred into engine side. With `kyuubi.engineEnv.SPARK_CONF_DIR=/apache/confs/spark/conf`, the value of `SPARK_CONF_DIR` on the engine side is set to `/apache/confs/spark/conf`.

Kyuubi Configurations

You can configure the Kyuubi properties in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`, see `$KYUUBI_HOME/conf/kyuubi-defaults.conf.template` as an example.

Authentication

Backend

Batch

Credentials

Ctl

Delegation

Engine

Event

Frontend

Ha

Kinit

Kubernetes

Metadata

Metrics

Operation

Server

Session

Spnego

Zookeeper

Spark Configurations

Via spark-defaults.conf

Setting them in `$SPARK_HOME/conf/spark-defaults.conf` supplies with default values for SQL engine application. Available properties can be found at Spark official online documentation for Spark Configurations

Via kyuubi-defaults.conf

Setting them in `$KYUUBI_HOME/conf/kyuubi-defaults.conf` supplies with default values for SQL engine application too. These properties will override all settings in `$SPARK_HOME/conf/spark-defaults.conf`

Via JDBC Connection URL

Setting them in the JDBC Connection URL supplies session-specific for each SQL engine. For example: `jdbc:hive2://localhost:10009/default;#spark.sql.shuffle.partitions=2;spark.executor.memory=5g`

- **Runtime SQL Configuration**
  - For Runtime SQL Configurations, they will take affect every time
- **Static SQL and Spark Core Configuration**
  - For Static SQL Configurations and other spark core configs, e.g. `spark.executor.memory`, they will take effect if there is no existing SQL engine application. Otherwise, they will just be ignored

Via SET Syntax

Please refer to the Spark official online documentation for SET Command

Flink Configurations

Via flink-conf.yaml

Setting them in `$FLINK_HOME/conf/flink-conf.yaml` supplies with default values for SQL engine application. Available properties can be found at Flink official online documentation for Flink Configurations

Via kyuubi-defaults.conf

Setting them in `$KYUUBI_HOME/conf/kyuubi-defaults.conf` supplies with default values for SQL engine application too. You can use properties with the additional prefix `flink.` to override settings in `$FLINK_HOME/conf/flink-conf.yaml`.

For example:

```
flink.parallelism.default 2
flink.taskmanager.memory.process.size 5g
```

The below options in `kyuubi-defaults.conf` will set `parallelism.default: 2` and `taskmanager.memory.process.size: 5g` into flink configurations.

Via JDBC Connection URL

Setting them in the JDBC Connection URL supplies session-specific for each SQL engine. For example: `jdbc:hive2://localhost:10009/default;#parallelism.default=2;taskmanager.memory.process.size=5g`

Via SET Statements

Please refer to the Flink official online documentation for SET Statements

Logging

Kyuubi uses log4j for logging. You can configure it using `$KYUUBI_HOME/conf/log4j2.xml`, see `$KYUUBI_HOME/conf/log4j2.xml.template` as an example.

Other Configurations

Hadoop Configurations

Specifying `HADOOP_CONF_DIR` to the directory containing Hadoop configuration files or treating them as Spark properties with a `spark.hadoop.` prefix. Please refer to the Spark official online documentation for Inheriting Hadoop Cluster Configuration. Also, please refer to the Apache Hadoop's online documentation for an overview on how to configure Hadoop.

Hive Configurations

These configurations are used for SQL engine application to talk to Hive MetaStore and could be configured in a `hive-site.xml`. Placed it in `$SPARK_HOME/conf` directory, or treat them as Spark properties with a `spark.hadoop.` prefix.

User Defaults

In Kyuubi, we can configure user default settings to meet separate needs. These user defaults override system defaults, but will be overridden by those from *JDBC Connection URL* or *Set Command* if could be. They will take effect when creating the SQL engine application ONLY. User default settings are in the form of `___{username}___.{config key}`. There are three continuous underscores(_) at both sides of the `username` and a dot(`.`) that separates the config key and the prefix. For example:

```
# For system defaults
spark.master=local
spark.sql.adaptive.enabled=true
# For a user named kent
___kent___.spark.master=yarn
___kent___.spark.sql.adaptive.enabled=false
# For a user named bob
___bob___.spark.master=spark://master:7077
___bob___.spark.executor.memory=8g
```

In the above case, if there are related configurations from *JDBC Connection URL*, `kent` will run his SQL engine application on YARN and prefer the Spark AQE to be off, while `bob` will activate his SQL engine application on a Spark standalone cluster with 8g heap memory for each executor and obey the Spark AQE behavior of Kyuubi system default. On the other hand, for those users who do not have custom configurations will use system defaults.

3.2.3 Engines

Deploy Kyuubi engines on Yarn

Deploy Kyuubi Spark Engine on Yarn

Requirements

When you want to deploy Kyuubi's Spark SQL engines on YARN, you'd better have cognition upon the following things.

- Knowing the basics about Running Spark on YARN
- A binary distribution of Spark which is built with YARN support
  - You can use the built-in Spark distribution
  - You can get it from Spark official website directly
  - You can Build Spark with `-Pyarn` maven option
- An active Apache Hadoop YARN cluster
- An active Apache Hadoop HDFS cluster
- Setup Hadoop client configurations at the machine the Kyuubi server locates

Configurations

Environment

Either `HADOOP_CONF_DIR` or `YARN_CONF_DIR` is configured and points to the Hadoop client configurations directory, usually, `$HADOOP_HOME/etc/hadoop`.

If the `HADOOP_CONF_DIR` points the YARN and HDFS cluster correctly, you should be able to run the `SparkPi` example on YARN.

```
$ HADOOP_CONF_DIR=/path/to/hadoop/conf $SPARK_HOME/bin/spark-submit \
    --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --queue thequeue \
    $SPARK_HOME/examples/jars/spark-examples*.jar \
    10
```

If the `SparkPi` passes, configure it in `$KYUUBI_HOME/conf/kyuubi-env.sh` or `$SPARK_HOME/conf/spark-env.sh`, e.g.

```
$ echo "export HADOOP_CONF_DIR=/path/to/hadoop/conf" >> $KYUUBI_HOME/conf/kyuubi-env.sh
```

Spark Properties

These properties are defined by Spark and Kyuubi will pass them to `spark-submit` to create Spark applications.

Note: None of these would take effect if the application for a particular user already exists.

- Specify it in the JDBC connection URL, e.g. `jdbc:hive2://localhost:10009/;#spark.master=yarn;spark.yarn.queue=thequeue`
- Specify it in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`
- Specify it in `$SPARK_HOME/conf/spark-defaults.conf`

Note: The priority goes down from top to bottom.

Master

Setting `spark.master=yarn` tells Kyuubi to submit Spark SQL engine applications to the YARN cluster manager.

Queue

Set `spark.yarn.queue=thequeue` in the JDBC connection string to tell Kyuubi to use the QUEUE in the YARN cluster, otherwise, the QUEUE configured at Kyuubi server side will be used as default.

Sizing

Pass the configurations below through the JDBC connection string to set how many instances of Spark executor will be used and how many cpus and memory will Spark driver, ApplicationMaster and each executor take.

It is recommended to use Dynamic Allocation with Kyuubi, since the SQL engine will be long-running for a period, execute user's queries from clients periodically, and the demand for computing resources is not the same for those queries. It is better for Spark to release some executors when either the query is lightweight, or the SQL engine is being idled.

Tuning

You can specify `spark.yarn.archive` or `spark.yarn.jars` to point to a world-readable location that contains Spark jars on HDFS, which allows YARN to cache it on nodes so that it doesn't need to be distributed each time an application runs.

Others

Please refer to Spark properties to check other acceptable configs.

Kerberos

Kyuubi currently does not support Spark's YARN-specific Kerberos Configuration, so `spark.kerberos.keytab` and `spark.kerberos.principal` should not use now.

Instead, you can schedule a periodically `kinit` process via `crontab` task on the local machine that hosts Kyuubi server or simply use Kyuubi Kinit.

Deploy Kyuubi Flink Engine on Yarn

Requirements

When you want to deploy Kyuubi's Flink SQL engines on YARN, you'd better have cognition upon the following things.

- Knowing the basics about Running Flink on YARN
- A binary distribution of Flink which is built with YARN support
  - Download a recent Flink distribution from the Flink official website and unpack it
- An active Apache Hadoop YARN cluster
  - Make sure your YARN cluster is ready for accepting Flink applications by running yarn top. It should show no error messages
- An active Object Storage cluster, e.g. HDFS, S3 and Minio etc.
- Setup Hadoop client configurations at the machine the Kyuubi server locates

Yarn Session Mode

Flink Configurations

```
execution.target: yarn-session
# Yarn Session Cluster application id.
yarn.application.id: application_00000000XX_00XX
```

Environment

Either HADOOP_CONF_DIR or YARN_CONF_DIR is configured and points to the Hadoop client configurations directory, usually, $HADOOP_HOME/etc/hadoop.

If the HADOOP_CONF_DIR points to the YARN and HDFS cluster correctly, and the HADOOP_CLASSPATH environment variable is set, you can launch a Flink on YARN session, and submit an example job:

```
# we assume to be in the root directory of
# the unzipped Flink distribution

# (0) export HADOOP_CLASSPATH
export HADOOP_CLASSPATH=`hadoop classpath`

# (1) Start YARN Session
./bin/yarn-session.sh --detached

# (2) You can now access the Flink Web Interface through the
# URL printed in the last lines of the command output, or through
# the YARN ResourceManager web UI.

# (3) Submit example job
./bin/flink run ./examples/streaming/TopSpeedWindowing.jar

# (4) Stop YARN session (replace the application id based
# on the output of the yarn-session.sh command)
echo "stop" | ./bin/yarn-session.sh -id application_XXXXX_XXX
```

If the TopSpeedWindowing passes, configure it in $KYUUBI_HOME/conf/kyuubi-env.sh

```
$ echo "export HADOOP_CONF_DIR=/path/to/hadoop/conf" >> $KYUUBI_HOME/conf/kyuubi-env.sh
```

Required Environment Variable

The FLINK_HADOOP_CLASSPATH is required, too.

For users who are using Hadoop 3.x, Hadoop shaded client is recommended instead of Hadoop vanilla jars. For users who are using Hadoop 2.x, FLINK_HADOOP_CLASSPATH should be set to hadoop classpath to use Hadoop vanilla jars. For users which does not use Hadoop services, e.g. HDFS, YARN at all, Hadoop client jars is also required, and recommend to use Hadoop shaded client as Hadoop 3.x's users do.

See HADOOP-11656 for details of Hadoop shaded client.

To use Hadoop shaded client, please configure $KYUUBI_HOME/conf/kyuubi-env.sh as follows:

```
$ echo "export FLINK_HADOOP_CLASSPATH=/path/to/hadoop-client-runtime-3.3.2.jar:/path/to/
↪hadoop-client-api-3.3.2.jar" >> $KYUUBI_HOME/conf/kyuubi-env.sh
```

To use Hadoop vanilla jars, please configure $KYUUBI_HOME/conf/kyuubi-env.sh as follows:

```
$ echo "export FLINK_HADOOP_CLASSPATH=`hadoop classpath`" >> $KYUUBI_HOME/conf/kyuubi-
↪env.sh
```

Deployment Modes Supported by Flink on YARN

For experiment use, we recommend deploying Kyuubi Flink SQL engine in Session Mode. At present, Application Mode and Per-Job Mode (deprecated) are not supported for Flink engine.

Kerberos

As Kyuubi Flink SQL engine wraps the Flink SQL client that currently does not support Flink Kerberos Configuration, so `security.kerberos.login.keytab` and `security.kerberos.login.principal` should not use now.

Instead, you can schedule a periodically `kinit` process via `crontab` task on the local machine that hosts Kyuubi server or simply use Kyuubi Kinit.

Deploy Kyuubi Hive Engine on Yarn

Requirements

When you want to deploy Kyuubi's Hive SQL engines on YARN, you'd better have cognition upon the following things.

- Knowing the basics about Running Hive on YARN
- A binary distribution of Hive
  - You can use the built-in Hive distribution
  - Download a recent Hive distribution from the Hive official website and unpack it
  - You can Build Hive
- An active Apache Hadoop YARN cluster
  - Make sure your YARN cluster is ready for accepting Hive applications by running yarn top. It should show no error messages
- An active Apache Hadoop HDFS cluster
- Setup Hadoop client configurations at the machine the Kyuubi server locates
- An active Hive Metastore Service

Configurations

Environment

Either `HADOOP_CONF_DIR` or `YARN_CONF_DIR` is configured and points to the Hadoop client configurations directory, usually, `$HADOOP_HOME/etc/hadoop`.

If the `HADOOP_CONF_DIR` points to the YARN and HDFS cluster correctly, you should be able to run the `Hive SQL` example on YARN.

```
$ $HIVE_HOME/bin/hiveserver2
# In another terminal
$ $HIVE_HOME/bin/beeline -u 'jdbc:hive2://localhost:10000/default'
0: jdbc:hive2://localhost:10000/default> CREATE TABLE pokes (foo INT, bar STRING);
0: jdbc:hive2://localhost:10000/default> INSERT INTO TABLE pokes VALUES (1, 'hello');
```

If the `Hive SQL` passes and there is a job in Yarn Web UI, It indicates the hive environment is normal.

Required Environment Variable

The `HIVE_HADOOP_CLASSPATH` is required, too. It should contain `commons-collections-*.jar`, `hadoop-client-runtime-*.jar`, `hadoop-client-api-*.jar` and `htrace-core4-*.jar`. All four jars are in the `HADOOP_HOME`.

For example, in Hadoop 3.1.0 version, the following is their location.

- `${HADOOP_HOME}/share/hadoop/common/lib/commons-collections-3.2.2.jar`
- `${HADOOP_HOME}/share/hadoop/client/hadoop-client-runtime-3.1.0.jar`
- `${HADOOP_HOME}/share/hadoop/client/hadoop-client-api-3.1.0.jar`
- `${HADOOP_HOME}/share/hadoop/common/lib/htrace-core4-4.1.0-incubating.jar`

Configure them in `$KYUUBI_HOME/conf/kyuubi-env.sh` or `$HIVE_HOME/conf/hive-env.sh`, e.g.

```
$ echo "export HADOOP_CONF_DIR=/path/to/hadoop/conf" >> $KYUUBI_HOME/conf/kyuubi-env.sh
$ echo "export HIVE_HADOOP_CLASSPATH=${HADOOP_HOME}/share/hadoop/common/lib/commons-
↪collections-3.2.2.jar:${HADOOP_HOME}/share/hadoop/client/hadoop-client-runtime-3.1.0.
↪jar:${HADOOP_HOME}/share/hadoop/client/hadoop-client-api-3.1.0.jar:${HADOOP_HOME}/
↪share/hadoop/common/lib/htrace-core4-4.1.0-incubating.jar" >> $KYUUBI_HOME/conf/kyuubi-
↪env.sh
```

Deploy Kyuubi engines on Kubernetes

Requirements

When you want to run Kyuubi's Spark SQL engines on Kubernetes, you'd better have cognition upon the following things.

- Read about Running Spark On Kubernetes
- An active Kubernetes cluster
- Kubectl
- KubeConfig of the target cluster

Configurations

Master

Spark on Kubernetes config master by using a special format.

`spark.master=k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port>`

You can use cmd `kubectl cluster-info` to get api-server host and port.

Docker Image

Spark ships a `./bin/docker-image-tool.sh` script to build and publish the Docker images for running Spark applications on Kubernetes.

When deploying Kyuubi engines against a Kubernetes cluster, we need to set up the docker images in the Docker registry first.

Example usage is:

```
./bin/docker-image-tool.sh -r <repo> -t my-tag build
./bin/docker-image-tool.sh -r <repo> -t my-tag push
# To build docker image with specify openJdk
./bin/docker-image-tool.sh -r <repo> -t my-tag -b java_image_tag=<openjdk:${java_image_
↪tag}> build
# To build additional PySpark docker image
./bin/docker-image-tool.sh -r <repo> -t my-tag -p ./kubernetes/dockerfiles/spark/
↪bindings/python/Dockerfile build
# To build additional SparkR docker image
./bin/docker-image-tool.sh -r <repo> -t my-tag -R ./kubernetes/dockerfiles/spark/
↪bindings/R/Dockerfile build
```

Test Cluster

You can use the shell code to test your cluster whether it is normal or not.

```
$SPARK_HOME/bin/spark-submit \
 --master k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> \
 --class org.apache.spark.examples.SparkPi \
 --conf spark.executor.instances=5 \
 --conf spark.dynamicAllocation.enabled=false \
 --conf spark.shuffle.service.enabled=false \
 --conf spark.kubernetes.container.image=<spark-image> \
 local://<path_to_examples.jar>
```

When running shell, you can use cmd `kubectl describe pod <podName>` to check if the information meets expectations.

ServiceAccount

When use Client mode to submit application, spark driver use the kubeconfig to access api-service to create and watch executor pods.

When use Cluster mode to submit application, spark driver pod use serviceAccount to access api-service to create and watch executor pods.

In both cases, you need to figure out whether you have the permissions under the corresponding namespace. You can use following cmd to create serviceAccount (You need to have the kubeconfig which have the create serviceAccount permission).

```
# create serviceAccount
kubectl create serviceaccount spark -n <namespace>
# binding role
kubectl create clusterrolebinding spark-role --clusterrole=edit --serviceaccount=
↪<namespace>:spark --namespace=<namespace>
```

Volumes

As it known to us all, Kubernetes can use configurations to mount volumes into driver and executor pods.

- hostPath: mounts a file or directory from the host node's filesystem into a pod.
- emptyDir: an initially empty volume created when a pod is assigned to a node.
- nfs: mounts an existing NFS(Network File System) into a pod.
- persistentVolumeClaim: mounts a PersistentVolume into a pod.

Note: Please see the Security section of this document for security issues related to volume mounts.

```
spark.kubernetes.driver.volumes.<type>.<name>.options.path=<dist_path>
spark.kubernetes.driver.volumes.<type>.<name>.mount.path=<container_path>

spark.kubernetes.executor.volumes.<type>.<name>.options.path=<dist_path>
spark.kubernetes.executor.volumes.<type>.<name>.mount.path=<container_path>
```

Read Using Kubernetes Volumes for more about volumes.

PodTemplateFile

Kubernetes allows defining pods from template files. Spark users can similarly use template files to define the driver or executor pod configurations that Spark configurations do not support.

To do so, specify the spark properties `spark.kubernetes.driver.podTemplateFile` and `spark.kubernetes.executor.podTemplateFile` to point to local files accessible to the spark-submit process.

Other

You can read Spark's official documentation for Running on Kubernetes for more information.

The Share Level Of Kyuubi Engines

The share level of Kyuubi engines describes the relationship between sessions and engines. It determines whether a new session can share an existing backend engine with other sessions or not. The sessions are also known as JDBC/ODBC/Thrift connections from clients that end-users create, and the engines are standalone applications with the full capabilities of Spark SQL, Flink SQL(under dev), running on single-node machines or clusters.

The share level of Kyuubi engines works the same whether in HA or single node mode. In other words, an engine is cluster widely shared by all Kyuubi server peers if could.

Why do we need this feature?

Apache Spark is a unified engine for large-scale data analytics. Using Spark to process data is like driving an all-wheel-drive hefty horsepower supercar. However,

- Cars have their limit of 0-60 times. In a similar way, all Spark applications also have to warm up before go full speed.
- Cars have a constant number of seats and are not allowed to be overloaded. Due to the master-slave architecture of Spark and the resource configured ahead, the overall workload of a single application is predictable.
- Cars have various shapes to meet our needs.

With this feature, Kyuubi give you a more flexible way to handle different big data workloads.

The current supported share levels

The current supported share levels are,

- Better isolation degree of engines gives us better stability of an engine and the query executions running on it.
- Better shareability of engines means we are more likely to reuse an engine which is already in full speed.

CONNECTION

Figure.1 CONNECTION Share Level

Each session with CONNECTION share level has a standalone engine for itself which is unreachable for anyone else. Within the session, a user or client can send multiple operation request, including metadata calls or queries, to the corresponding engine.

Although it is still an interactive form, this model does allow for more practical batch processing jobs as well.

When closing session, the corresponding engine will be shutdown at the same time.

USER(Default)

Figure.2 USER Share Level

All sessions with USER share level use the same engine if and only if the session user is the same.

Those sessions share the same engine with objects belong to the one and only `SparkContext` instance, including `Classes/Classloaders`, `SparkConf`, `Driver/Executors`, `Hive Metastore Client`, etc. But each session can still have its own `SparkSession` instance, which contains separate session state, including temporary views, SQL config, UDFs etc. Setting `kyuubi.engine.single.spark.session` to true will make `SparkSession` instance a singleton and share across sessions.

When closing session, the corresponding engine will not be shutdown. When all sessions are closed, the corresponding engine still has a time-to-live lifespan. This TTL allows new sessions to be established quickly without waiting for the engine to start.

GROUP

Figure.3 GROUP Share Level

An engine will be shared by all sessions created by all users belong to the same primary group name. The engine will be launched by the group name as the effective username, so here the group name is kind of special user who is able to visit the compute resources/data of a team. It follows the Hadoop GroupsMapping to map user to a primary group. If the primary group is not found, it falls back to the USER level.

The mechanisms of `SparkContext`, `SparkSession` and TTL works similarly to USER share level.

Tips for authorization in GROUP share level:

The session user and the primary group name(as sparkUser/execute user) will be both accessible at engine-side. By default, the sparkUser will be used to check the YARN/HDFS ACLs. If you want fine-grained access control for session user, you need to get it from `SparkContext.getLocalProperty("kyuubi.session.user")` and send it to security service, like Apache Ranger.

SERVER

Figure.4 SERVER Share Level

Literally, this model is similar to Spark Thrift Server with High availability.

Subdomain

For USER, GROUP, or SERVER share levels, you can further use `kyuubi.engine.share.level.subdomain` to isolate the engine. That is, you can also create multiple engines for a single user, group or server(cluster). For example, in USER share level, you can use `kyuubi.engine.share.level.subdomain=sd1` and `kyuubi.engine.share.level.subdomain=sd2` to create two standalone engines for user `Tom`.

The `kyuubi.engine.share.level.subdomain` shall be configured in the JDBC connection URL to tell the Kyuubi server which engine you want to use.

Hybrid

All supported share levels can be used together in a single Kyuubi server or cluster.

Related Configurations

- kyuubi.engine.share.level(kyuubi.session.engine.share.level)
  - Default: USER
  - Candidates: USER, CONNECTION, GROUP, SERVER
  - Meaning: The base level for how an engine is created, cached and shared to sessions.
  - Usage: It can be set both in the server configuration file and also connection URL. The latter has higher priority.
- kyuubi.session.engine.idle.timeout
  - Default: PT30M (30 min)
  - Candidates: a proper timeout
  - Meaning: Time to live since engine becomes idle
  - Usage: It can be set both in the server configuration file and also connection URL. The latter has higher priority.
- kyuubi.engine.share.level.subdomain(kyuubi.engine.share.level.sub.domain)
  - Default:
  - Candidates: a valid zookeeper a child node
  - Meaning: Add a subdomain under the base level to make further isolation for engines
  - Usage: It can be set both in the server configuration file and also connection URL. The latter has higher priority.

Conclusion

With this feature, end-users are able to leverage engines in different ways to handle their different workloads, such as large-scale ETL jobs and interactive ad hoc queries.

The TTL Of Kyuubi Engines

For a multi-tenant cluster, its overall resource utilization is a KPI that measures how effectively its resource is utilized against its availability or capacity. To better improve the overall resource utilization of the cluster,

- At cluster layer, we leverage the capabilities, such as Capacity Scheduler, of resource scheduling management services, such as YARN and K8s.
- At application layer, we'd be better to acquire and release resources according to the real workloads.

The Big Contributors Of Resource Waste

- The time to wait for the resource to be allocated, such as the scheduling delay, the start/stop cost.
  - A longer time-to-live(TTL) for allocated resources can significantly reduce such time costs within an application.
- The time being idle of the resource.
  - A shorter time to live for allocated resources can make all resources in rapid turnarounds across applications.

TTL Types In Kyuubi Engines

- Engine TTL
  - The TTL of engines describes how long an engine will be cached after all sessions are disconnected.
- Executor TTL
  - The TTL of the executor describes how long an executor will be cached when no tasks come.

Configurations

Engine TTL

The above two configurations can be used together to set the TTL of engines. These configurations are user-facing and able to use in JDBC urls. Note that, for connection share level engines that will be terminated at once when the connection is disconnected, these configurations not necessarily work in this case.

Executor TTL

Executor TTL is part of functionality of Apache Spark's Dynamic Resource Allocation.

The Spark SQL Engine Configuration Guide

Kyuubi aims to bring Spark to end-users who need not qualify with Spark or something else related to the big data area. End-users can write SQL queries through JDBC against Kyuubi and nothing more. The Kyuubi server-side or the corresponding engines could do most of the optimization. On the other hand, we don't wholly restrict end-users to special handling of specific cases to benefit from the following documentations. Even if you don't use Kyuubi, as a simple Spark user, I'm sure you'll find the next articles instructive.

How To Use Spark Dynamic Resource Allocation (DRA) in Kyuubi

When we adopt Kyuubi in a production environment, we always want to use the environment's computing resources more cost-effectively and efficiently. Cluster managers such as K8S and Yarn manage the cluster compute resources, divided into different queues or namespaces with different ACLs and quotas.

In Kyuubi, we acquire computing resources from the cluster manager to submit the engines. The engines respond to various types of client requests, some of which consume many computing resources to process, while others may require very few resources to complete. If we have fixed-sized engines, a.k.a. with a fixed number for `spark.executor.instances`, it may cause a waste of resources for some lightweight workloads, while for some heavyweight workloads, it should probably not have enough concurrency capacity resulting in poor performance.

When the engine has executor idled, we should release it back to the resource pool promptly, and conversely, when the engine is doing chubby tasks, we should be able to get and use more resources more efficiently. On the one hand, we need to rely on the resource manager's capabilities for efficient resource allocation, resource isolation, and sharing. On the other hand, we need to enable Spark's DRA feature for the engines' executors' elastic scaling.

The Basics of Dynamic Resource Allocation

Spark provides a mechanism to dynamically adjust the application resources based on the workload, which means that an application may give resources back to the cluster if they are no longer used and request them again later when there is demand. This feature is handy if multiple applications share resources on YARN, Kubernetes, and other platforms.

For Kyuubi engines, which are typical Spark applications, the dynamic allocation allows Spark to dynamically scale the cluster resources allocated to them based on the workloads. When dynamic allocation is enabled, and an engine has a backlog of pending tasks, it can request executors via `ExecutorAllocationManager`. When the engine has executors that become idle, the executors are released, and the occupied resources are given back to the cluster manager. Then other engines or other applications run in the same queue could acquire the resources.

How to Enable Dynamic Resource Allocation

The prerequisite for enabling this feature is for downstream stages to have proper access to shuffle data, even if the executors that generated the data are recycled.

Spark provides two implementations for shuffle data tracking. If either is enabled, we can use the DRA feature properly.

Dynamic Resource Allocation w/ External Shuffle Service

Having an external shuffle service (ESS) makes sure that all the data is stored outside of executors. This prerequisite was needed as Spark needed to ensure that the executors' removal does not remove shuffle data. When deploying Kyuubi with a cluster manager that provides ESS, enable DRA for all the engines with the configurations below.

```
spark.dynamicAllocation.enabled=true
spark.shuffle.service.enabled=true
```

Another thing to be sure of is that `spark.shuffle.service.port` should be configured to point to the port on which the ESS is running.

Dynamic Allocation w/o External Shuffle Service

Implementations of the ESS feature are cluster manager dependent. Yarn, for instance, where the ESS needs to be deployed cluster-widely and is actually running in the Yarn's `NodeManager` component. Nevertheless, if run Kyuubi's engines on Kubernetes, the ESS is not an option yet. Since Spark 3.0, the DRA can run without ESS. The relative feature called `Shuffle Tracking` was introduced by SPARK-27963.

When deploying Kyuubi with a cluster manager that without ESS or the ESS is not attractive, enable DRA with `Shuffle Tracking` instead for all the engines with the configurations below.

```
spark.dynamicAllocation.enabled=true
spark.dynamicAllocation.shuffleTracking.enabled=true
```

When `Shuffle Tracking` is enabled, `spark.dynamicAllocation.shuffleTracking.timeout(default: infinity)` controls the timeout for executors that are holding shuffle data. Spark will rely on the shuffles being

garbage collected to be able to release executors by default. When the garbage collection is not cleaning up shuffles quickly enough, this timeout forces Spark to delete executors even when they are storing shuffle data.

Sizing for engines w/ Dynamic Resource Allocation

Resources for a single executor, such as CPUs and memory, can be fixed size. So, the range [`minExecutors`, `maxExecutors`] determines how many recourses the engine can take from the cluster manager.

On the one hand, the `minExecutors` tells Spark to keep how many executors at least. If it is set too close to 0(default), the engine might complain about a lack of resources if the cluster manager is quite busy and for a long time. However, the larger the `minExecutors` goes, the more resources may be wasted during the engine's idle time.

On the other hand, the `maxExecutors` determines the upper bound executors of an engine could reach. From the individual engine perspective, this value is the larger, the better, to handle heavier queries. However, we must limit it to a reasonable range in terms of the entire cluster's resources. Otherwise, a large query may trigger the engine where it runs to consume too many resources from the queue/namespace and occupy them for a considerable time, which could be a bad idea for using the resources efficiently. In this case, we would prefer that such an enormous task be done more slowly in a limited amount of concurrency.

The following Spark configurations consist of sizing for the DRA.

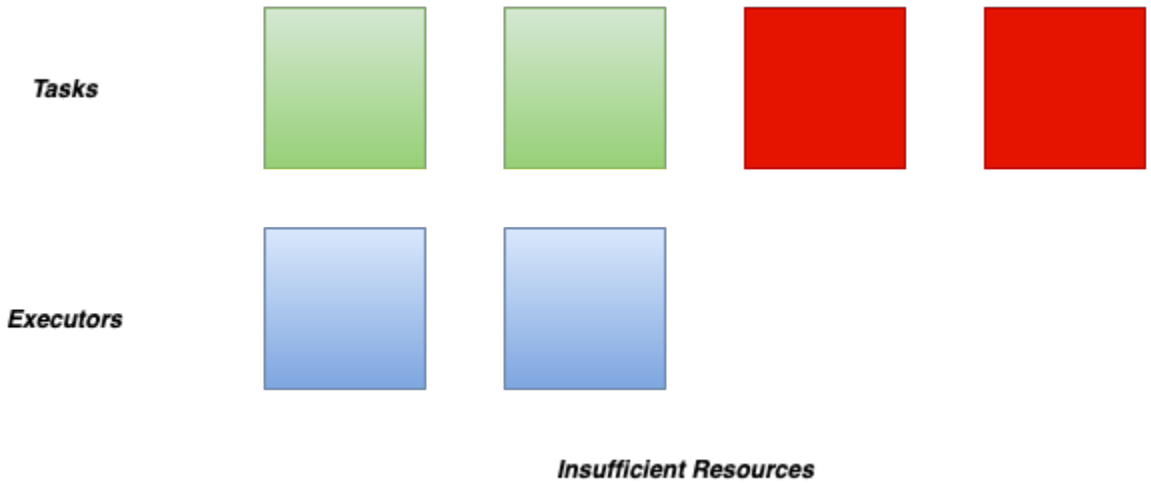
```
spark.dynamicAllocation.minExecutors=10
spark.dynamicAllocation.maxExecutors=500
```

Additionally, another config called `spark.dynamicAllocation.initialExecutors` can be used to decide how many executors to request during engine bootstrapping or failover.

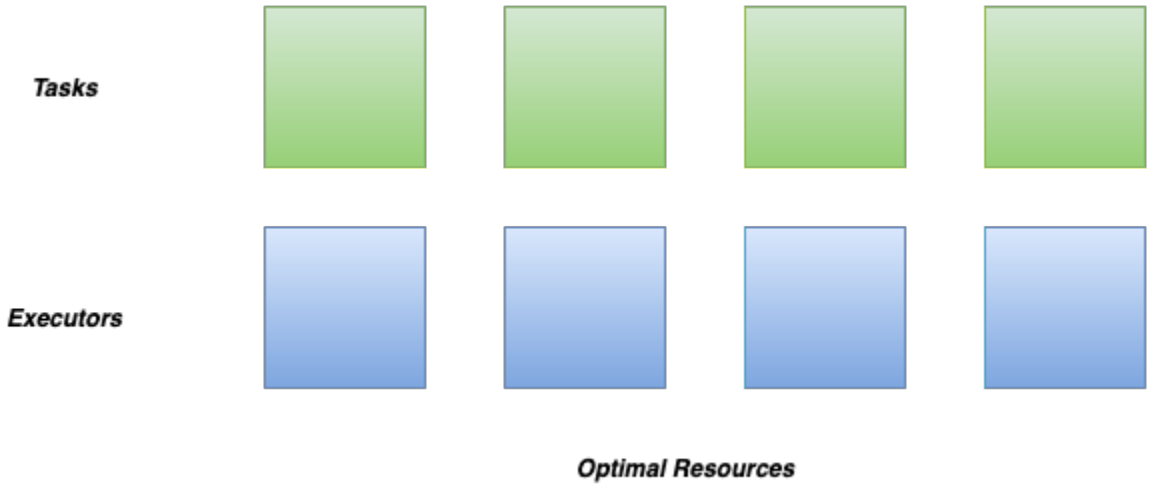
Ideally, the size relationship between them should be as `minExecutors` <= `initialExecutors` < `maxExecutors`.

Resource Allocation Policy

When the DRA notices that the current resources are insufficient for the current workload, it will request more executors.

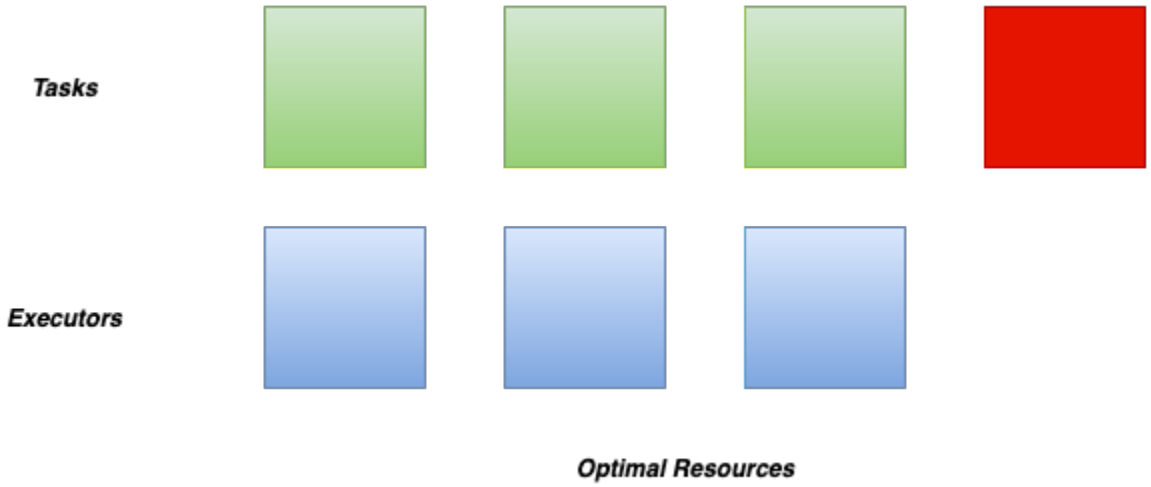


By default, the dynamic allocation will request enough executors to maximize the parallelism according to the number of tasks to process.

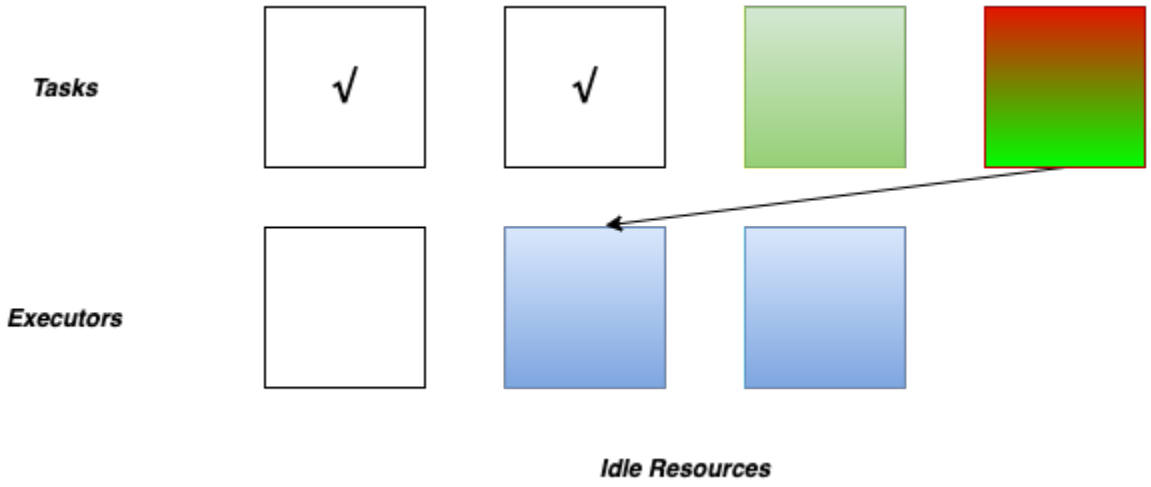


While this minimizes the latency of the job, but with small tasks, the default behavior can waste many resources due to executor allocation overhead, as some executors might not even do any work.

In this case, we can adjust `spark.dynamicAllocation.executorAllocationRatio` a bit lower to reduce the number of executors w.r.t. full parallelism. For instance, 0.5 will divide the target number of executors by 2.



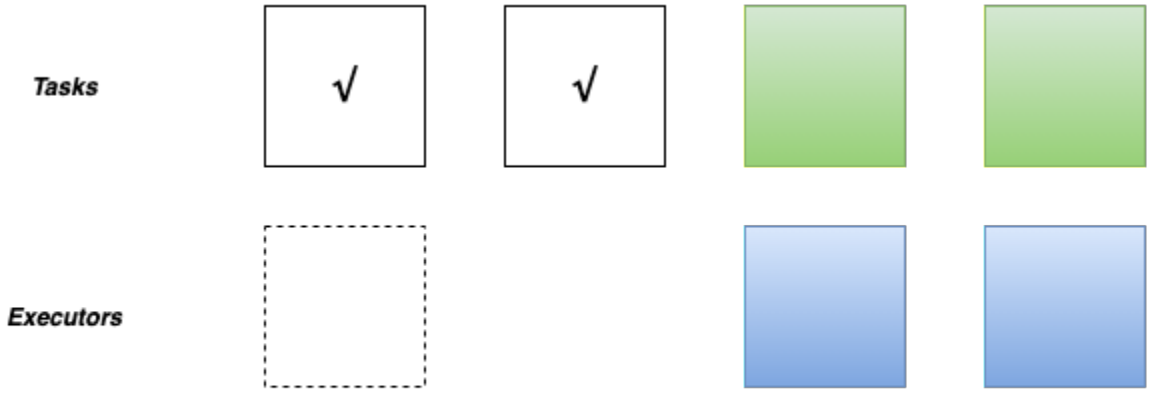
After finish one task, Spark Driver will schedule a new task for the executor with available cores. When pending tasks become fewer and fewer, some executors become idle for no new coming tasks.



Idle Resources

If one executor reached the maximum timeout, it will be removed.

```
spark.dynamicAllocation.executorIdleTimeout=60s
spark.dynamicAllocation.cachedExecutorIdleTimeout=infinity
```



Optimal Resources

If the DRA finds there have been pending tasks backlogged for more than the timeouts, new executors will be requested, controlled by the following configs.

```
spark.dynamicAllocation.schedulerBacklogTimeout=1s
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout=1s
```

Best Practices for Applying DRA to Kyuubi

Kyuubi is a long-running service to make it easier for end-users to use Spark SQL without having much of Spark's basic knowledge. It is essential to have a basic configuration for resource management that works for most scenarios on the server-side.

Setting Default Configurations

Configuring by `spark-defaults.conf` at the engine side is the best way to set up Kyuubi with DRA. All engines will be instantiated with DRA enabled.

Here is a config setting that we use in our platform when deploying Kyuubi.

```
spark.dynamicAllocation.enabled=true
##false if prefer shuffle tracking than ESS
spark.shuffle.service.enabled=true
spark.dynamicAllocation.initialExecutors=10
spark.dynamicAllocation.minExecutors=10
spark.dynamicAllocation.maxExecutors=500
spark.dynamicAllocation.executorAllocationRatio=0.5
spark.dynamicAllocation.executorIdleTimeout=60s
spark.dynamicAllocation.cachedExecutorIdleTimeout=30min
# true if prefer shuffle tracking than ESS
spark.dynamicAllocation.shuffleTracking.enabled=false
spark.dynamicAllocation.shuffleTracking.timeout=30min
spark.dynamicAllocation.schedulerBacklogTimeout=1s
spark.dynamicAllocation.sustainedSchedulerBacklogTimeout=1s
spark.cleaner.periodicGC.interval=5min
```

Note that, `spark.cleaner.periodicGC.interval=5min` is useful here when `spark.dynamicAllocation.shuffleTracking.enabled` is enabled, as we can tell Spark to be more active for shuffle data GC.

Setting User Default Settings

On the server-side, the workloads for different users might be different.

Then we can set different defaults for them via the User Defaults in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`

```
# For a user named kent
___kent___.spark.dynamicAllocation.maxExecutors=20
# For a user named bob
___bob___.spark.dynamicAllocation.maxExecutors=600
```

In this case, the user named `kent` can only use 20 executors for his engines, but `bob` can use 600 executors for better performance or handle heavy workloads.

Dynamically Setting

All AQE related configurations are static of Spark core and unchangeable by `SET` commands before each SQL query. For example,

```
SET spark.dynamicAllocation.maxExecutors=33;
SELECT * FROM default.tableA;
```

For the above case, the value - 33 will not affect as Spark does not support change core configurations in runtime.

Instead, end-users can set them via JDBC Connection URL for some specific cases.

References

1. Spark Official Online Document: Dynamic Resource Allocation
2. Spark Official Online Document: Dynamic Resource Allocation Configurations
3. SPARK-27963: Allow dynamic allocation without an external shuffle service

How To Use Spark Adaptive Query Execution (AQE) in Kyuubi

The Basics of AQE

Spark Adaptive Query Execution (AQE) is a query re-optimization that occurs during query execution.

In terms of technical architecture, the AQE is a framework of dynamic planning and replanning of queries based on runtime statistics, which supports a variety of optimizations such as,

- Dynamically Switch Join Strategies
- Dynamically Coalesce Shuffle Partitions
- Dynamically Handle Skew Joins

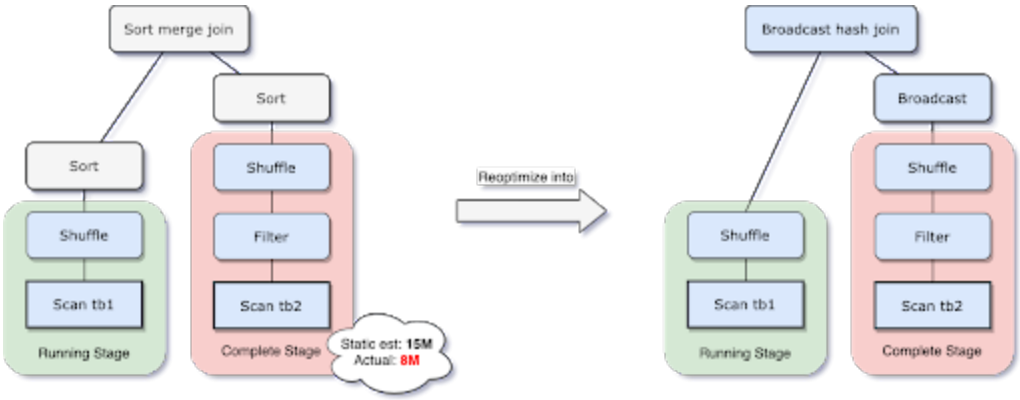
In Kyuubi, we strongly recommended that you turn on all capabilities of AQE by default for Kyuubi engines, no matter on what platform you run Kyuubi and Spark.

Dynamically Switch Join Strategies

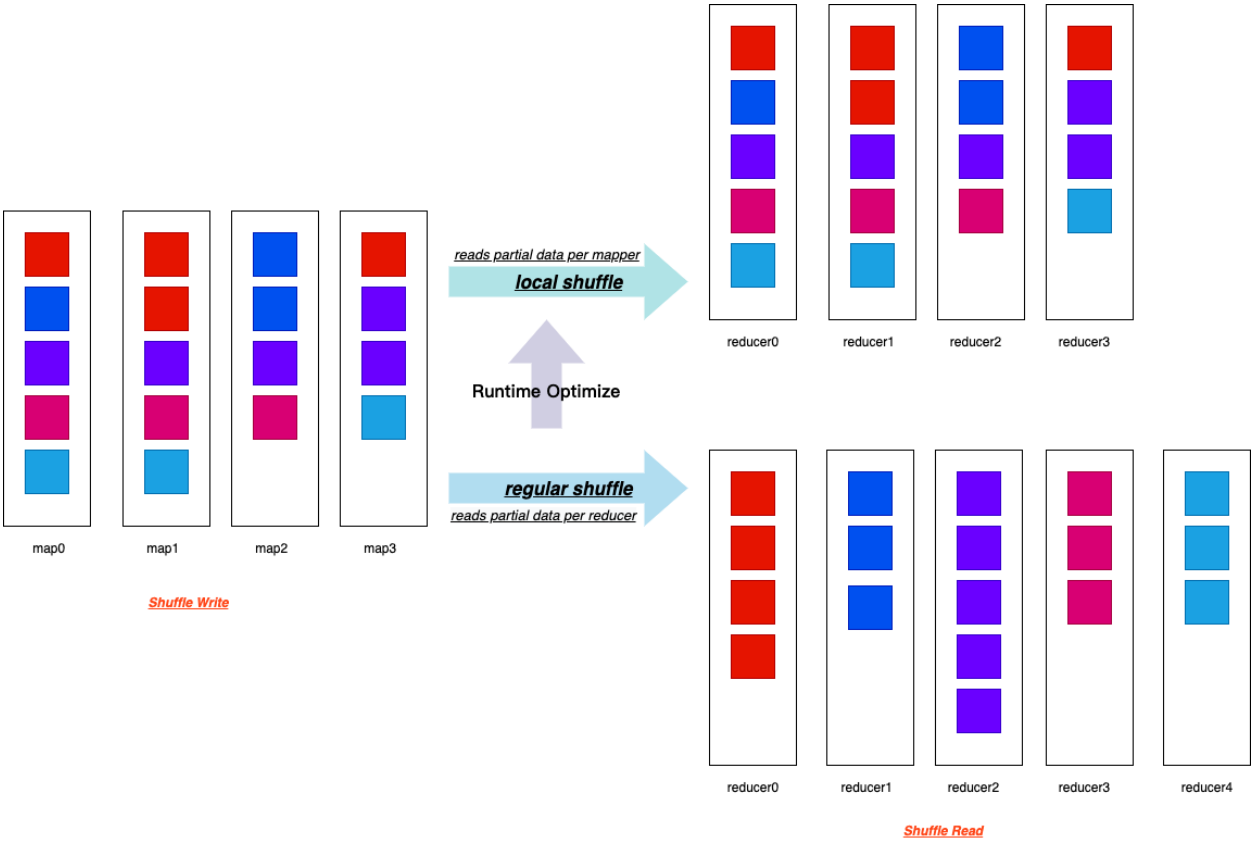
Spark supports several join strategies, among which `BroadcastHash Join` is usually the most performant when any join side fits well in memory. And for this reason, Spark plans a `BroadcastHash Join` if the estimated size of a join relation is less than the `spark.sql.autoBroadcastJoinThreshold`.

```
spark.sql.autoBroadcastJoinThreshold=10M
```

Without AQE, the estimated size of join relations comes from the statistics of the original table. It can go wrong in most real-world cases. For example, the join relation is a convergent but composite operation rather than a single table scan. In this case, Spark might not be able to switch the join-strategy to `BroadcastHash Join`. While with AQE, we can runtime calculate the size of the composite operation accurately. And then, Spark now can replan the join strategy unmistakably if the size fits `spark.sql.autoBroadcastJoinThreshold`



What's more, when `spark.sql.adaptive.localShuffleReader.enabled=true` and after converting `SortMerge Join` to `BroadcastHash Join`, Spark also does future optimize to reduce the network traffic by converting a regular shuffle to a localized shuffle.



As shown in the above fig, the local shuffle reader can read all necessary shuffle files from its local storage, actually without performing the shuffle across the network.

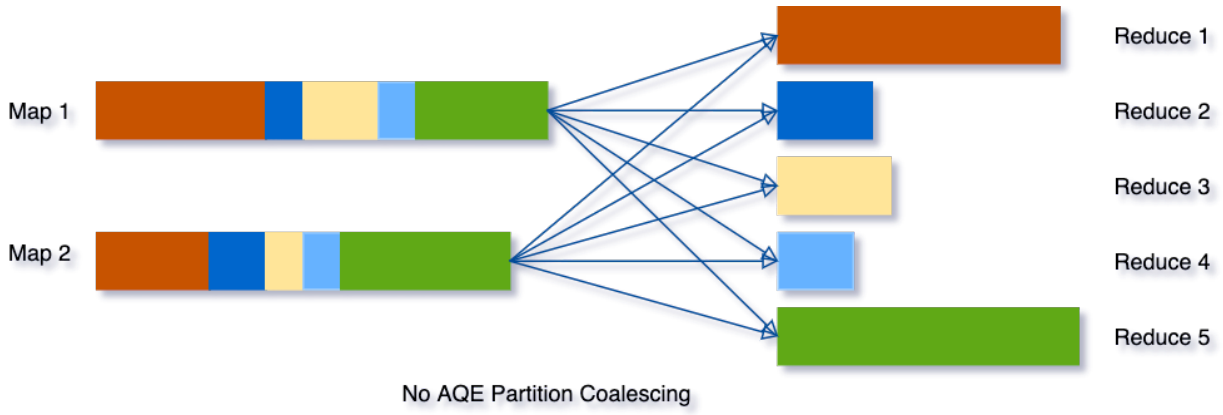
The local shuffle reader optimization consists of avoiding shuffle when the `SortMerge Join` transforms to `BroadcastHash Join` after applying the AQE rules.

Dynamically Coalesce Shuffle Partitions

Without this feature, Spark itself could be a small files maker sometimes, especially in a pure SQL way like Kyuubi does, for example,

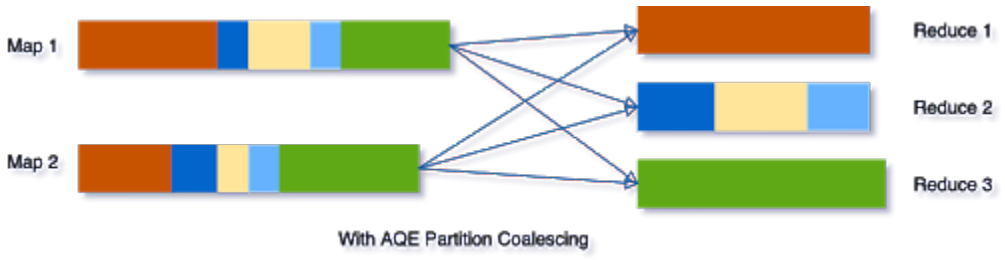
1. When `spark.sql.shuffle.partitions` is set too large compared to the total output size, there comes very small or empty files after a shuffle stage.
2. When Spark performs a series of optimized `BroadcastHash Join` and `Union` together, the final output size for each partition might be reduced by the join conditions. However, the total final output file numbers get to explode.
3. Some pipeline jobs with selective filters to produce temporary data.
4. e.t.c

Reading small files leads to very small partitions or tasks. Spark tasks will have worse I/O throughput and tend to suffer more from scheduling overhead and task setup overhead.



Combining small partitions saves resources and improves cluster throughput. Spark provides several ways to handle small file issues, for example, adding an extra shuffle operation on the partition columns with the `distribute by` clause or using `HINT`[5]. In most scenarios, you need to have a good grasp of your data, Spark jobs, and configurations to apply these solutions case by case. Mostly, the daily used config - `spark.sql.shuffle.partitions` is data-dependent and unchangeable with a single Spark SQL query. For real-life Spark jobs with multiple stages, it' impossible to use it as one size to fit all.

But with AQE, things become more comfortable for you as Spark will do the partition coalescing automatically.



It can simplify the tuning of shuffle partition numbers when running Spark SQL queries. You do not need to set a proper shuffle partition number to fit your dataset.

To enable this feature, we need to set the below two configs to true.

```
spark.sql.adaptive.enabled=true
spark.sql.adaptive.coalescePartitions.enabled=true
```

Other Tips for Best Practises

For further tuning our Spark jobs with this feature, we also need to be aware of these configs.

```
spark.sql.adaptive.advisoryPartitionSizeInBytes=128m
spark.sql.adaptive.coalescePartitions.minPartitionNum=1
spark.sql.adaptive.coalescePartitions.initialPartitionNum=200
```

How to set `spark.sql.adaptive.advisoryPartitionSizeInBytes`?

It stands for the advisory size in bytes of the shuffle partition during adaptive query execution, which takes effect when Spark coalesces small shuffle partitions or splits skewed shuffle partition. The default value of `spark.sql.adaptive.advisoryPartitionSizeInBytes` is 64M. Typically, if we are reading and writing data with HDFS, matching it with the block size of HDFS should be the best choice, i.e. 128MB or 256MB.

Consequently, all blocks or partitions in Spark and files in HDFS are chopped up to 128MB/256MB chunks. And think about it, now all tasks for scans, sinks, and middle shuffle maps are dealing with mostly even-sized data partitions. It will make us much easier to set up executor resources or even one size to fit all.

How to set `spark.sql.adaptive.coalescePartitions.minPartitionNum`?

It stands for the suggested (not guaranteed) minimum number of shuffle partitions after coalescing. If not set, the default value is the default parallelism of the Spark application. The default parallelism is defined by `spark.default.parallelism` or else the total count of cores registered. I guess the motivation of this behavior made by the Spark community is to maximize the use of the resources and concurrency of the application.

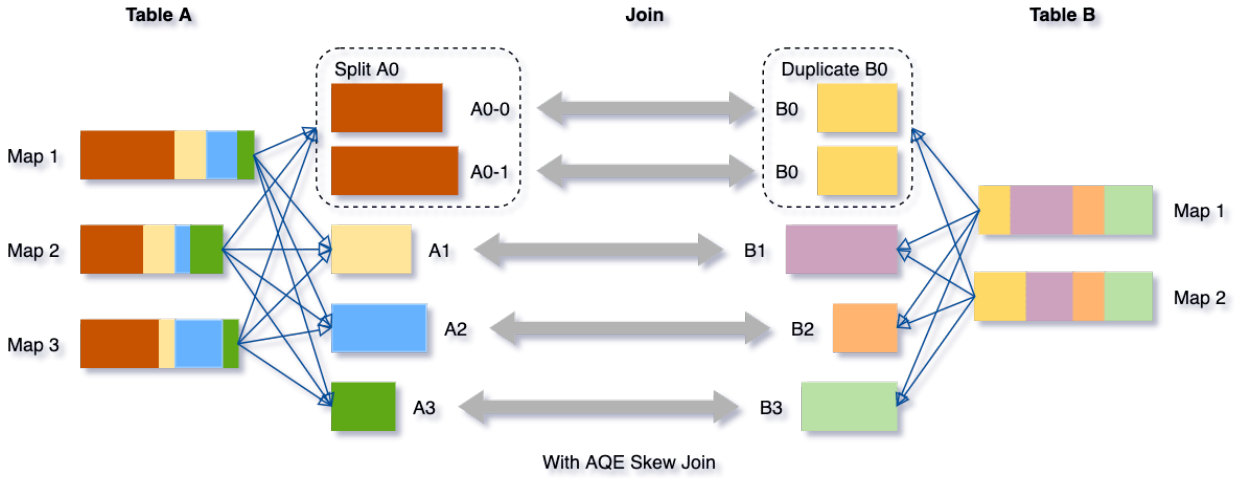
But there are always exceptions. Relating these two seemingly unrelated parameters can be somehow tricky for users. This config is optional by default which means users may not touch it in most real-world cases. But `spark.default.parallelism` has a long history and is well known then. If users set the default parallelism to an illegitimate high value unexpectedly, it could block AQE from coalescing partitions to a fair number. Another scenario that requires special attention is writing data. Usually, coalescing partitions to avoid small file issues is more critical than task concurrency for final output stages. A better data layout can benefit plenty of downstream jobs. I suggest that we set `spark.sql.adaptive.coalescePartitions.minPartitionNum` to 1 in this case as Spark will try its best to but not guaranteed to coalesce partitions for output.

How to set `spark.sql.adaptive.coalescePartitions.initialPartitionNum`?

It stands for the initial number of shuffle partitions before coalescing. By default, it equals to `spark.sql.shuffle.partitions(200)`. Firstly, it's better to set it explicitly rather than falling back to `spark.sql.shuffle.partitions`. Spark community suggests set a large number to it as Spark will dynamically coalesce shuffle partitions, which I cannot agree more.

Dynamically Handle Skew Joins

Without AQE, the data skewness is very likely to occur for map-reduce computing models in the shuffle phase. Data skewness can cause Spark jobs to have one or more tailing tasks, severely downgrading queries' performance. This feature dynamically handles skew in `SortMerge Join` by splitting (and replicating if needed) skewed tasks into roughly evenly sized tasks. For example, The optimization will split oversized partitions into subpartitions and join them to the other join side's corresponding partition.



To enable this feature, we need to set the below two configs to true.

```
spark.sql.adaptive.enabled=true
spark.sql.adaptive.skewJoin.enabled=true
```

Other Tips for Best Practises

For further tuning our Spark jobs with this feature, we also need to be aware of these configs.

```
spark.sql.adaptive.skewJoin.skewedPartitionFactor=5
spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes=256M
spark.sql.adaptive.advisoryPartitionSizeInBytes=64M
```

How to set `spark.sql.adaptive.skewJoin.skewedPartitionFactor` and `skewedPartitionThresholdInBytes`?

Spark uses these two configs and the median(**not average**) partition size to detect whether a partition skew or not.

```
partition size > skewedPartitionFactor * the median partition size && \
skewedPartitionThresholdInBytes
```

As Spark splits skewed partitions targeting spark.sql.adaptive.advisoryPartitionSizeInBytes, ideally `skewedPartitionThresholdInBytes` should be larger than `advisoryPartitionSizeInBytes`. In this case, anytime you increase `advisoryPartitionSizeInBytes`, you should also increase `skewedPartitionThresholdInBytes` if you tend to enable the feature.

Hidden Features

DemoteBroadcastHashJoin

Internally, Spark has an optimization rule that detects a join child with a high ratio of empty partitions and adds a no-broadcast-hash-join hint to avoid broadcasting it.

```
spark.sql.adaptive.nonEmptyPartitionRatioForBroadcastJoin=0.2
```

By default, if there are only less than 20% partitions of the dataset contain data, Spark will not broadcast the dataset.

EliminateJoinToEmptyRelation

This optimization rule detects and converts a Join to an empty LocalRelation.

Disabling the Hidden Features

We can exclude some of the AQE additional rules if performance regression or bug occurs. For example,

```
SET spark.sql.adaptive.optimizer.excludedRules=org.apache.spark.sql.execution.adaptive.
↪DemoteBroadcastHashJoin
```

Best Practices for Applying AQE to Kyuubi

Kyuubi is a long-running service to make it easier for end-users to use Spark SQL without having much of Spark's basic knowledge. It is essential to have a basic configuration that works for most scenarios on the server-side.

Setting Default Configurations

Configuring by `spark-defaults.conf` at the engine side is the best way to set up Kyuubi with AQE. All engines will be instantiated with AQE enabled.

Here is a config setting that we use in our platform when deploying Kyuubi.

```
spark.sql.adaptive.enabled=true
spark.sql.adaptive.forceApply=false
spark.sql.adaptive.logLevel=info
spark.sql.adaptive.advisoryPartitionSizeInBytes=256m
spark.sql.adaptive.coalescePartitions.enabled=true
spark.sql.adaptive.coalescePartitions.minPartitionNum=1
spark.sql.adaptive.coalescePartitions.initialPartitionNum=8192
spark.sql.adaptive.fetchShuffleBlocksInBatch=true
spark.sql.adaptive.localShuffleReader.enabled=true
spark.sql.adaptive.skewJoin.enabled=true
spark.sql.adaptive.skewJoin.skewedPartitionFactor=5
spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes=400m
spark.sql.adaptive.nonEmptyPartitionRatioForBroadcastJoin=0.2
spark.sql.adaptive.optimizer.excludedRules
spark.sql.autoBroadcastJoinThreshold=-1
```

Tips

Turn on AQE by default can significantly improve the user experience. Other sub-features are all enabled. `advisoryPartitionSizeInBytes` is targeting the HDFS block size `minPartitionNum` is set to 1 for the reason of coalescing first. `initialPartitionNum` has a high value. Since AQE requires at least one shuffle, ideally, we need to set `autoBroadcastJoinThreshold` to -1 to involving `SortMerge Join` with a shuffle for all user queries with joins. But then, the Dynamically Switch Join Strategies feature seems can not be applied later in this case. It appears to be a typo limitation of Spark AQE so far.

Dynamically Setting

All AQE related configurations are runtime changeable, which means that it can still modify some specific configs by `SET` syntaxes for each SQL query with more precise control on the client-side.

Spark Known issues

SPARK-33933: Broadcast timeout happened unexpectedly in AQE

For Spark versions(<3.1), we need to increase `spark.sql.broadcastTimeout(300s)` higher even the broadcast relation is tiny.

For other potential problems that may be found in the AQE features of Spark, you may refer to SPARK-33828: SQL Adaptive Query Execution QA.

References

1. Adaptive Query Execution
2. Adaptive Query Execution: Speeding Up Spark SQL at Runtime
3. SPARK-31412: New Adaptive Query Execution in Spark SQL
4. SPARK-28560: Optimize shuffle reader to local shuffle reader when smj converted to bhj in adaptive execution
5. Coalesce and Repartition Hint for SQL Queries

Solution for Big Result Sets

Typically, when a user submits a SELECT query to Spark SQL engine, the Driver calls `collect` to trigger calculation and collect the entire data set of all tasks(a.k.a. partitions of an RDD), after all partitions data arrived, then the client pulls the result set from the Driver through the Kyuubi Server in small batch.

Therefore, the bottleneck is the Spark Driver for a query with a big result set. To avoid OOM, Spark has a configuration `spark.driver.maxResultSize` which default is `1g`, you should enlarge it as well as `spark.driver.memory` if your query has result set in several GB. But what if the result set size is dozens GB or event hundreds GB? It would be best if you have incremental collection mode.

Incremental collection

Since v1.4.0-incubating, Kyuubi supports incremental collection mode, it is a solution for big result sets. This feature is disabled in default, you can turn on it by setting the configuration `kyuubi.operation.incremental.collect` to `true`.

The incremental collection changes the gather method from `collect` to `toLocalIterator`. `toLocalIterator` is a Spark action that sequentially submits Jobs to retrieve partitions. As each partition is retrieved, the client through pulls the result set from the Driver through the Kyuubi Server streamingly. It reduces the Driver memory significantly from the size of the complete result set to the maximum partition.

The incremental collection is not the silver bullet, you should turn it on carefully, because it can significantly hurt performance. And even in incremental collection mode, when multiple queries execute concurrently, each query still requires one partition of data in Driver memory. Therefore, it is still important to control the number of concurrent queries to avoid OOM.

Use in single connections

As above explains, the incremental collection mode is not suitable for common query sense, you can enable incremental collection mode for specific queries by using

```
beeline -u 'jdbc:hive2://kyuubi:10009/?spark.driver.maxResultSize=8g;spark.driver.
↪memory=12g#kyuubi.engine.share.level=CONNECTION;kyuubi.operation.incremental.
↪collect=true' \
    --incremental=true \
    -f big_result_query.sql
```

`--incremental=true` is required for beeline client, otherwise, the entire result sets is fetched and buffered before being displayed, which may cause client side OOM.

Change incremental collection mode in session

The configuration `kyuubi.operation.incremental.collect` can also be changed using `SET` in session.

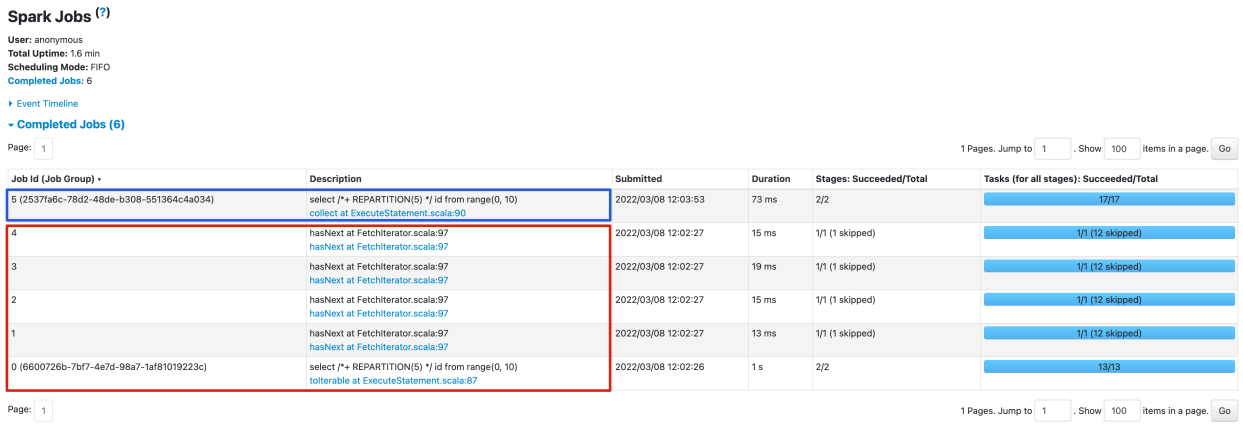
```
~ beeline -u 'jdbc:hive2://localhost:10009'
Connected to: Apache Kyuubi (Incubating) (version 1.5.0-SNAPSHOT)

0: jdbc:hive2://localhost:10009/> set kyuubi.operation.incremental.collect=true;
+----------------------------------------+--------+
|                  key                   | value  |
+----------------------------------------+--------+
| kyuubi.operation.incremental.collect   | true   |
+----------------------------------------+--------+
1 row selected (0.039 seconds)

0: jdbc:hive2://localhost:10009/> select /*+ REPARTITION(5) */ * from range(1, 10);
+-----+
| id  |
+-----+
| 2   |
| 6   |
| 7   |
```

(continues on next page)

(continued from previous page)

```
| 0   |
| 5   |
| 3   |
| 4   |
| 1   |
| 8   |
| 9   |
+-----+
10 rows selected (1.929 seconds)

0: jdbc:hive2://localhost:10009/> set kyuubi.operation.incremental.collect=false;
+----------------------------------------+--------+
|                  key                   | value  |
+----------------------------------------+--------+
| kyuubi.operation.incremental.collect   | false  |
+----------------------------------------+--------+
1 row selected (0.027 seconds)

0: jdbc:hive2://localhost:10009/> select /*+ REPARTITION(5) */ * from range(1, 10);
+-----+
| id  |
+-----+
| 2   |
| 6   |
| 7   |
| 0   |
| 5   |
| 3   |
| 4   |
| 1   |
| 8   |
| 9   |
+-----+
10 rows selected (0.128 seconds)
```

From the Spark UI, we can see that in incremental collection mode, the query produces 5 jobs (in red square), and in normal mode, only produces 1 job (in blue square).



3.3 Kyuubi Security Overview

Securing Kyuubi involves enabling authentication(authn), authorization(authz) and encryption, etc.

3.3.1 Kyuubi Authentication Mechanism

In a secure cluster, services should be able to identify and authenticate callers. As the fact that the user claims does not necessarily mean this is true.

The authentication process of kyuubi is used to verify the user identity that a client used to talk to the kyuubi server. Once done, a trusted connection will be set up between the client and server if successful; otherwise, rejected.

Note: This only authenticate whether a user or client can connect with Kyuubi server or not using the provided identity. For other secured services that this user wants to interact with, he/she also needs to pass the authentication process of each service, for instance, Hive Metastore, YARN, HDFS.

The related configurations can be found at Authentication Configurations

Configure Kyuubi to use Kerberos Authentication

If you are deploying Kyuubi with a kerberized Hadoop cluster, it is strongly recommended that `kyuubi.authentication` should be set to *KERBEROS* too.

Kerberos Overview

Kerberos is a network authentication protocol that provides the tools of authentication and strong cryptography over the network. The Kerberos protocol uses strong cryptography so that a client or a server can prove its identity to its server or client across an insecure network connection. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.

The Kerberos architecture is centered around a trusted authentication service called the key distribution center, or KDC. Users and services in a Kerberos environment are referred to as principals; each principal shares a secret, such as a password, with the KDC.

Enable Kerberos Authentication

To enable the Kerberos authentication method, we need to

Create a Kerberos principal and keytab

You can use the following commands in a Linux-based Kerberos environment to set up the identity and update the keytab file:

The `kyuubi.keytab` file must be owned and readable by the Linux login user.

```
# kadmin
  : addprinc -randkey superuser/FQDN@REALM
  : ktadd -k ./kyuubi.keytab superuser/FQDN@REALM
```

Note: A widespread use case of kyuubi is to replace HiveServer2/Hive QL with Kyuubi/Spark SQL. If an existing HiveServer2 environment is already there, copying the environment and reusing the keytab and principal of HiveServer2 is a convenient way.

Enable Hadoop Impersonation

If background cluster is also an kerberized Hadoop cluster, we need to enable the impersonation capability of the superuser we use to start kyuubi server.

You can configure proxy user using properties `hadoop.proxyuser.$superuser.hosts` along with either or both of `hadoop.proxyuser.$superuser.groups` and `hadoop.proxyuser.$superuser.users`.

For instance, by specifying as below in `core-site.xml`, the `superuser` named `admin` can connect only from `host1` and `host2` to impersonate a user belonging to `group1` and `group2`.

```
<property>
  <name>hadoop.proxyuser.admin.hosts</name>
  <value>host1,host2</value>
</property>
<property>
  <name>hadoop.proxyuser.admin.groups</name>
  <value>group1,group2</value>
</property>
```

Here,

- `admin` is the principal(short name) used to start kyuubi servers
- `host1` and `host2` are node addresses of kyuubi servers
- `group1` and `group2` are groups of client users

Note: These configurations need to be configured in the Hadoop cluster and refreshed to take effect.

Note: If you are using the keytab of existing HiveServer2, this step can also be omitted

Configure the authentication properties

Configure the following properties to `$KYUUBI_HOME/conf/kyuubi-defaults.conf` on each node where kyuubi server is installed.

```
kyuubi.authentication=KERBEROS
kyuubi.kinit.principal=superuser/FQDN@REALM
kyuubi.kinit.keytab=/path/to/kyuubi.keytab
```

These configurations also need to be set to enable KERBEROS authentication.

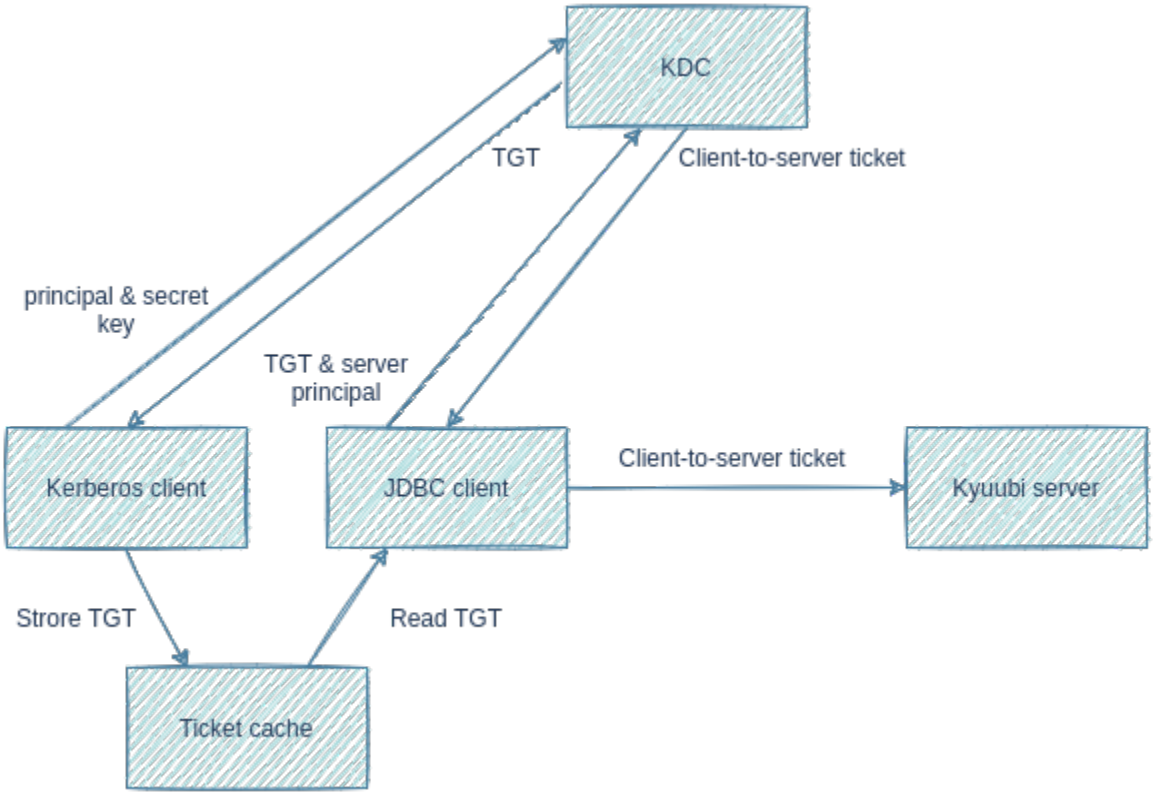
Refresh all the kyuubi server instances

Restart all the kyuubi server instances or Refresh Configurations to activate the settings.

Configure Kerberos for clients to Access Kerberized Kyuubi

Instructions

When Kyuubi is secured by Kerberos, the authentication procedure becomes a little complicated.



The graph above shows a simplified kerberos authentication procedure:

1. Kerberos client sends user principal and secret key to KDC. Secret key can be a password or a keytab file.
2. KDC returns a `ticket-granting ticket`(TGT).
3. Kerberos client stores TGT into a ticket cache.
4. JDBC client, such as beeline and BI tools, reads TGT from the ticket cache.
5. JDBC client sends TGT and server principal to KDC.
6. KDC returns a `client-to-server ticket`.
7. JDBC client sends `client-to-server ticket` to Kyuubi server to prove its identity.

In the rest part of this page, we will describe steps needed to pass through this authentication.

Install Kerberos Client

Usually, Kerberos client is installed as default. You can validate it using klist tool.

Linux command and output:

```
$ klist -V
Kerberos 5 version 1.15.1
```

MacOS command and output:

```
$ klist --version
klist (Heimdal 1.5.1apple1)
Copyright 1995-2011 Kungliga Tekniska Högskolan
Send bug-reports to heimdal-bugs@h5l.org
```

Windows command and output:

```
> klist -V
Kerberos for Windows
```

If the client is not installed, you should install it ahead based on the OS platform.We recommend you to install the MIT Kerberos Distribution as all commands in this guide is based on it.

Configure Kerberos Client

Kerberos client needs a configuration file for tuning up the creation of Kerberos ticket cache. Following is the configuration file's default location on different OS:

You can use `KRB5_CONFIG` environment variable to overwrite the default location.

The configuration file should be configured to point to the same KDC as Kyuubi points to.

Get Kerberos TGT

Execute `kinit` command to get TGT from KDC.

Suppose user principal is `kyuubi_user@KYUUBI.APACHE.ORG` and user keytab file name is `kyuubi_user.keytab`, the command should be:

```
$ kinit -kt kyuubi_user.keytab kyuubi_user@KYUUBI.APACHE.ORG

(Command is identical on different OS platform)
```

You may also execute `kinit` command with principal and password to get TGT:

```
$ kinit kyuubi_user@KYUUBI.APACHE.ORG
Password for kyuubi_user@KYUUBI.APACHE.ORG: password

(Command is identical on different OS platform)
```

If the command executes successfully, TGT will be store in ticket cache.Use `klist` command to print TGT info in ticket cache:

```
$ klist

Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: kyuubi_user@KYUUBI.APACHE.ORG

Valid starting       Expires              Service principal
2021-12-13T18:44:58  2021-12-14T04:44:58  krbtgt/KYUUBI.APACHE.ORG@KYUUBI.APACHE.ORG
    renew until 2021-12-14T18:44:57

(Command is identical on different OS platform. Ticket cache location may be different.)
```

Ticket cache may have different storage type on different OS platform.

For example,

You can find your ticket cache type and location in the `Ticket cache` part of `klist` output.

Note:

- Ensure your ticket cache type is `FILE` as JVM can only read ticket cache stored as file.
- Do not store TGT into default ticket cache if you are running Kyuubi and execute `kinit` on the same host with the same OS user. The default ticket cache is already used by Kyuubi server.

Either because the default ticket cache is not a file, or because it is used by Kyuubi server, you should store ticket cache in another file location.This can be achieved by specifying a file location with `-c` argument in `kinit` command.

For example,

```
$ kinit -c /tmp/krb5cc_beeline -kt kyuubi_user.keytab kyuubi_user@KYUUBI.APACHE.ORG

(Command is identical on different OS platform)
```

To check the ticket cache, specify the file location with `-c` argument in `klist` command.

For example,

```
$ klist -c /tmp/krb5cc_beeline

(Command is identical on different OS platform)
```

Add Kerberos Client Configuration File to JVM Search Path

The JVM, which JDBC client is running on, also needs to read the Kerberos client configuration file. However, JVM uses different default locations from Kerberos client, and does not honour `KRB5_CONFIG` environment variable.

You can use JVM system property, `java.security.krb5.conf`, to overwrite the default location.

Add Kerberos Ticket Cache to JVM Search Path

JVM determines the ticket cache location in the following order:

1. Path specified by `KRB5CCNAME` environment variable. Path must start with `FILE:`.
2. `/tmp/krb5cc_%{uid}` on Unix-like OS, e.g. Linux, MacOS
3. `${user.home}/krb5cc_${user.name}` if `${user.name}` is not null
4. `${user.home}/krb5cc` if `${user.name}` is null

Note:

- `${user.home}` and `${user.name}` are JVM system properties.
- `${user.home}` should be replaced with `${user.dir}` if `${user.home}` is null.

Ensure your ticket cache is stored as a file and put it in one of the above locations.

Ensure core-site.xml Exists in Classpath

Like hadoop clients, `hadoop.security.authentication` should be set to `KERBEROS` in `core-site.xml` to let Hive JDBC driver use Kerberos authentication. `core-site.xml` should be placed under beeline's classpath or BI tools' classpath.

Beeline

Here are the usual locations where `core-site.xml` should exist for different beeline distributions:

If `core-site.xml` is not found in above locations, create one with the following content:

```
<configuration>
  <property>
    <name>hadoop.security.authentication</name>
    <value>kerberos</value>
  </property>
</configuration>
```

BI Tools

As to BI tools, ways to add `core-site.xml` varies.Take DBeaver as an example. We can add files to DBeaver's classpath through its `Global libraries` preference.As `Global libraries` only accepts jar files, you should package `core-site.xml` into a jar file.

```
$ jar -c -f core-site.jar core-site.xml

(Command is identical on different OS platform)
```

Connect with JDBC URL

The last step is to connect to Kyuubi with the right JDBC URL.The JDBC URL should be in format:

```
jdbc:hive2://<kyuubi_server_address>:<kyuubi_server_port>/<db>;principal=<kyuubi_server_
↪principal>
```

or

```
jdbc:hive2://<kyuubi_server_address>:<kyuubi_server_port>/<db>;kyuubiServerPrincipal=
↪<kyuubi_server_principal>
```

Note:

- `principal` is inherited from Hive JDBC Driver and is a little ambiguous, and we could use `kyuubiServerPrincipal` as its alias.
- `kyuubi_server_principal` is the value of `kyuubi.kinit.principal` set in `kyuubi-defaults.conf`.
- As a command line argument, JDBC URL should be quoted to avoid being split into 2 commands by ";".
- As to DBeaver, `<db>;principal=<kyuubi_server_principal>` should be set as the `Database/Schema` argument.

Configure Kyuubi to use LDAP Authentication

Warning: the page is still in-progress.

Configure Kyuubi to Use JDBC Authentication

Kyuubi supports authentication via JDBC query. A query is prepared with user/password value and sent to the database configured in JDBC URL. Authentication passes if the result set is not empty.

The SQL statement must start with the `SELECT` clause. Placeholders are supported and listed below for substitution:

- `${user}`
- `${password}`

For example, `SELECT 1 FROM auth_db.auth_table WHERE user=${user} AND passwd=MD5(CONCAT(salt, ${password}))` will be prepared as `SELECT 1 FROM auth_db.auth_table WHERE user=? AND passwd=MD5(CONCAT(salt,?))` with value replacement of `user` and `password` in string type.

Enable JDBC Authentication

To enable the JDBC authentication method, we need to

- Put the JDBC driver jar file to `$KYUUBI_HOME/jars` directory to make it visible for the classpath of the kyuubi server.
- Configure the following properties to `$KYUUBI_HOME/conf/kyuubi-defaults.conf` on each node where kyuubi server is installed.

Configure the authentication properties

Configure the following properties to `$KYUUBI_HOME/conf/kyuubi-defaults.conf` on each node where kyuubi server is installed.

```
kyuubi.authentication=JDBC
kyuubi.authentication.jdbc.driver.class = com.mysql.jdbc.Driver
kyuubi.authentication.jdbc.url = jdbc:mysql://127.0.0.1:3306/auth_db
kyuubi.authentication.jdbc.user = bowenliang123
kyuubi.authentication.jdbc.password = bowenliang123@kyuubi
kyuubi.authentication.jdbc.query = SELECT 1 FROM auth_table WHERE user=${user} AND␣
↪passwd=MD5(CONCAT(salt,${password}))
```

Authentication with In-memory Database

Used with auto created in-memory database, JDBC authentication could be applied for token validation without starting up a dedicated database service or setting up a custom plugin.

Consider authentication for a pair of a username and a token which contacted with an `expire_time` in 'yyyyMMddHHmm' format and a MD5 signature generated with sequence of `expire_time`, `username` and a secret key. With the following example, an H2 in-memory database will be auto crated with Kyuubi Server and used for authentication with its system function `HASH` and checking token expire time with `NOW()`.

```
kyuubi.authentication=JDBC
kyuubi.authentication.jdbc.driver.class = org.h2.Driver
kyuubi.authentication.jdbc.url = jdbc:h2:mem:
kyuubi.authentication.jdbc.user = no_user
kyuubi.authentication.jdbc.query = SELECT 1 FROM ( \
  SELECT ${user} as username, 'secret_key' as secret_key, \
  SUBSTRING(${password}, 0, 12) as expire_time, \
  SUBSTRING(${password}, 13) as signed \
  ) WHERE signed = RAWTOHEX(HASH('MD5', CONCAT(secret_key, username, expire_time))) \
  AND PARSEDATETIME(expire_time,'yyyyMMddHHmm') > NOW()
```

Configure Kyuubi to use Custom Authentication

Besides the builtin authentication methods, kyuubi supports custom authentication implementations of *org.apache.kyuubi.service.authentication.PasswdAuthenticationProvider*.

```
package org.apache.kyuubi.service.authentication

import javax.security.sasl.AuthenticationException

trait PasswdAuthenticationProvider {

  /**
   * The authenticate method is called by the Kyuubi Server authentication layer
   * to authenticate users for their requests.
   * If a user is to be granted, return nothing/throw nothing.
   * When a user is to be disallowed, throw an appropriate [[AuthenticationException]].
   *
```

(continues on next page)

(continued from previous page)

```
   * @param user     The username received over the connection request
   * @param password The password received over the connection request
   *
   * @throws AuthenticationException When a user is found to be invalid by the␣
↪implementation
   */
  @throws[AuthenticationException]
  def authenticate(user: String, password: String): Unit
}
```

Build A Custom Authenticator

To create custom Authenticator class derived from the above interface, we need to:

- Referencing the library

```
<dependency>
   <groupId>org.apache.kyuubi</groupId>
   <artifactId>kyuubi-common_2.12</artifactId>
   <version>1.5.2-incubating</version>
   <scope>provided</scope>
</dependency>
```

- Implement PasswdAuthenticationProvider - Sample Code

Enable Custom Authentication

To enable the custom authentication method, we need to

- Put the jar package to `$KYUUBI_HOME/jars` directory to make it visible for the classpath of the kyuubi server.
- Configure the following properties to `$KYUUBI_HOME/conf/kyuubi-defaults.conf` on each node where kyuubi server is installed.
- Restart all the kyuubi server instances

3.3.2 Kyuubi Authorization Guide

Kyuubi Spark AuthZ Plugin

New in version 1.6.0.

Kyuubi AuthZ Plugin For Spark SQL

Security is one of the fundamental features for enterprise adoption with Kyuubi. When deploying Kyuubi against secured clusters, storage-based authorization is enabled by default, which only provides file-level coarse-grained authorization mode. When row/column-level fine-grained access control is required, we can enhance the data access model with the Kyuubi Spark AuthZ plugin.

Authorization in Kyuubi

Storage-based Authorization

As Kyuubi supports multi tenancy, a tenant can only visit authorized resources, including computing resources, data, etc. Most file systems, such as HDFS, support ACL management based on files and directories.

A so called Storage-based authorization mode is supported by Kyuubi by default. In this model, all objects, such as databases, tables, partitions, in meta layer are mapping to folders or files in the storage layer, as well as their permissions.

Storage-based authorization offers users with database, table and partition-level coarse-gained access control.

SQL-standard authorization with Ranger

A SQL-standard authorization usually offers a row/colum-level fine-grained access control to meet the real-world data security need.

Apache Ranger is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform. This plugin enables Kyuubi with data and metadata control access ability for Spark SQL Engines, including,

- Column-level fine-grained authorization
- Row-level fine-grained authorization, a.k.a. Row-level filtering
- Data masking

The Plugin Itself

Kyuubi Spark Authz Plugin itself provides general purpose for ACL management for data & metadata while using Spark SQL. It is not necessary to deploy it with the Kyuubi server and engine, and can be used as an extension for any Spark SQL jobs. However, the authorization always requires a robust authentication layer and multi tenancy support, so Kyuubi is a perfect match.

Restrict security configuration

End-users can disable the AuthZ plugin by modifying Spark's configuration. For example:

```
select * from parquet.`/path/to/table`
```

```
set spark.sql.optimizer.excludedRules=org.apache.kyuubi.plugin.spark.authz.ranger.
↪RuleAuthorization
```

Kyuubi provides a mechanism to ban security configurations to enhance the security of production environments

Note: How do we modify the Spark engine configurations please refer to the documentation Spark Configurations

Restrict session level config

You can specify config *kyuubi.session.conf.ignore.list* values and config *kyuubi.session.conf.restrict.list* values to disable changing session+ level configuration on the server side. For example:

```
kyuubi.session.conf.ignore.list    spark.driver.memory,spark.sql.optimizer.excludedRules
```

```
kyuubi.session.conf.restrict.list    spark.driver.memory,spark.sql.optimizer.
↪excludedRules
```

Restrict operation level config

You can specify config *spark.kyuubi.conf.restricted.list* values to disable changing operation level configuration on the engine side, this means that the config key in the restricted list cannot set dynamic configuration via SET syntax. For examples:

```
spark.kyuubi.conf.restricted.list  spark.sql.adaptive.enabled,spark.sql.adaptive.
↪skewJoin.enabled
```

Note:

1. Note that config *spark.sql.runSQLOnFiles* values and config *spark.sql.extensions* values are by default in the engine restriction configuration list
2. A set statement with key equal to *spark.sql.optimizer.excludedRules* and value containing *org.apache.kyuubi.plugin.spark.authz.ranger.** also does not allow modification.

Building Kyuubi Spark AuthZ Plugin

Build with Apache Maven

Kyuubi Spark AuthZ Plugin is built using Apache Maven. To build it, `cd` to the root direct of kyuubi project and run:

```
build/mvn clean package -pl :kyuubi-spark-authz_2.12 -DskipTests
```

After a while, if everything goes well, you will get the plugin finally in two parts:

- The main plugin jar, which is under `./extensions/spark/kyuubi-spark-authz/target/kyuubi-spark-authz_${scala.binary.version}-${project.version}.jar`
- The least transitive dependencies needed, which are under `./extensions/spark/kyuubi-spark-authz/target/scala-${scala.binary.version}/jars`

Build against Different Apache Spark Versions

The maven option `spark.version` is used for specifying Spark version to compile with and generate corresponding transitive dependencies. By default, it is always built with the latest `spark.version` defined in kyuubi project main pom file. Sometimes, it may be incompatible with other Spark distributions, then you may need to build the plugin on your own targeting the Spark version you use.

For example,

```
build/mvn clean package -pl :kyuubi-spark-authz_2.12 -DskipTests -Dspark.version=3.0.2
```

The available `spark.version`s are shown in the following table.

Currently, Spark released with Scala 2.12 are supported.

Build against Different Apache Ranger Versions

The maven option `ranger.version` is used for specifying Ranger version to compile with and generate corresponding transitive dependencies. By default, it is always built with the latest `ranger.version` defined in kyuubi project main pom file. Sometimes, it may be incompatible with other Ranger Admins, then you may need to build the plugin on your own targeting the Ranger Admin version you connect with.

```
build/mvn clean package -pl :kyuubi-spark-authz_2.12 -DskipTests -Dranger.version=0.7.0
```

The available `ranger.version`s are shown in the following table.

Currently, all ranger releases are supported.

Test with ScalaTest Maven plugin

If you omit `-DskipTests` option in the command above, you will also get all unit tests run.

```
build/mvn clean package -pl :kyuubi-spark-authz_2.12
```

If any bug occurs and you want to debug the plugin yourself, you can configure `-DdebugForkedProcess=true` and `-DdebuggerPort=5005`(optional).

```
build/mvn clean package -pl :kyuubi-spark-authz_2.12 -DdebugForkedProcess=true
```

The tests will suspend at startup and wait for a remote debugger to attach to the configured port.

We will appreciate if you can share the bug or the fix to the Kyuubi community.

Installing and Configuring Kyuubi Spark AuthZ Plugin

Pre-install

- Apache Ranger

  This plugin works as a ranger rest client with Apache Ranger admin server to do privilege check. Thus, a ranger server need to be installed ahead and available to use.

- Building(optional)

  If your ranger admin or spark distribution is not compatible with the official pre-built artifact in maven central. You need to build the plugin targeting the spark/ranger you are using by yourself.

Install

With the `kyuubi-spark-authz_*.jar` and its transitive dependencies available for spark runtime classpath, such as

- Copied to `$SPARK_HOME/jars`, or
- Specified to `spark.jars` configuration

Configure

Settings for Connecting Ranger Admin

ranger-spark-security.xml

- Create `ranger-spark-security.xml` in `$SPARK_HOME/conf` and add the following configurations for pointing to the right Ranger admin server.

```
<configuration>
    <property>
        <name>ranger.plugin.spark.policy.rest.url</name>
        <value>ranger admin address like http://ranger-admin.org:6080</value>
    </property>

    <property>
        <name>ranger.plugin.spark.service.name</name>
        <value>a ranger hive service name</value>
    </property>

    <property>
        <name>ranger.plugin.spark.policy.cache.dir</name>
        <value>./a ranger hive service name/policycache</value>
    </property>

    <property>
        <name>ranger.plugin.spark.policy.pollIntervalMs</name>
        <value>5000</value>
    </property>

    <property>
        <name>ranger.plugin.spark.policy.source.impl</name>
        <value>org.apache.ranger.admin.client.RangerAdminRESTClient</value>
    </property>

</configuration>
```

Using Macros in Row Level Filters

Macros are now supported for using user/group/tag in row filter expressions, introduced in Ranger 2.3. This feature helps significantly simplify row filter expressions by using user/group/tag's attributes instead of explicit conditions. Considering a user with an attribute `born_city` of value `Guangzhou` , the row filter condition as `city='${{USER.born_city}}'` will be transformed to `city='Guangzhou'` in execution plan. More supported macros and usage refer to RANGER-3605 and RANGER-3550. Add the following configs to `ranger-spark-security.xml` to enable UserStore Enricher required by macros.

```
    <property>
        <name>ranger.plugin.spark.enable.implicit.userstore.enricher</name>
        <value>true</value>
        <description>Enable UserStoreEnricher for fetching user and group attributes if␣
↪using macros or scripts in row-filters since Ranger 2.3</description>
    </property>

    <property>
        <name>ranger.plugin.hive.policy.cache.dir</name>
        <value>./a ranger hive service name/policycache</value>
        <description>As Authz plugin reuses hive service def, a policy cache path is␣
↪required for caching UserStore and Tags for "hive" service def, while "ranger.plugin.
↪spark.policy.cache.dir config" is the path for caching policies in service. </
↪description>
    </property>
```

Showing all disallowed privileges

By default, Authz plugin checks required privileges one by one and throw the first unsatisfied privilege in exception. By setting `ranger.plugin.spark.authorize.in.single.call` to `true`, Authz plugin executes access checks in single call and throws all disallowed privileges in exception message.

```
<property>
    <name>ranger.plugin.spark.authorize.in.single.call</name>
    <value>true</value>
    <description>Enable access checks in single call with all disallowed privileges␣
↪thrown in exception. Default value is false.</description>
</property>
```

ranger-spark-audit.xml

Create `ranger-spark-audit.xml` in `$SPARK_HOME/conf` and add the following configurations to enable/disable auditing.

```
<configuration>

    <property>
        <name>xasecure.audit.is.enabled</name>
        <value>true</value>
    </property>
```

(continues on next page)

(continued from previous page)

```
    <property>
        <name>xasecure.audit.destination.db</name>
        <value>false</value>
    </property>

    <property>
        <name>xasecure.audit.destination.db.jdbc.driver</name>
        <value>com.mysql.jdbc.Driver</value>
    </property>

    <property>
        <name>xasecure.audit.destination.db.jdbc.url</name>
        <value>jdbc:mysql://10.171.161.78/ranger</value>
    </property>

    <property>
        <name>xasecure.audit.destination.db.password</name>
        <value>rangeradmin</value>
    </property>

    <property>
        <name>xasecure.audit.destination.db.user</name>
        <value>rangeradmin</value>
    </property>

</configuration>
```

Settings for Spark Session Extensions

Add `org.apache.kyuubi.plugin.spark.authz.ranger.RangerSparkExtension` to the spark configuration `spark.sql.extensions`.

```
spark.sql.extensions=org.apache.kyuubi.plugin.spark.authz.ranger.RangerSparkExtension
```

3.3.3 Kinit Auxiliary Service

Kinit auxiliary service is a critical service both for authentication between Kyuubi client/server and for authentication between Kyuubi server/Hadoop cluster in a Kerberos environment. It will get a Kerberos Ticket Cache from KDC and periodically re-kinit to keep the Ticket Cache fresh.

Note:

- Kinit auxiliary service is critical to Kyuubi Kerberos authentication, but not vice versa.
- Kinit auxiliary service can also work with other authentication mode.

Installing and Configuring the Kerberos Clients

Usually, Kerberos client is installed as default. You can validate it using `klist` tool.

```
$ klist -V
Kerberos 5 version 1.15.1
```

If the client is not installed, you should install it ahead based on the OS platform that you prepare to run Kyuubi.

`krb5.conf` is a configuration file for tuning up the creation of Kerberos ticket cache. The default location is `/etc` on Linux, and we can use `KRB5_CONFIG` environmental variable to overwrite the location of the configuration file.

Replace or configure `krb5.conf` to point to the KDC.

Kerberos Ticket

Kerberos client is aimed to generate a Ticket Cache file. Then, Kyuubi can use this Ticket Cache to authenticate with those kerberized services, e.g. HDFS, YARN, and Hive Metastore server, etc.

A Kerberos ticket cache contains a service and a client principal names, lifetime indicators, flags, and the credential itself, e.g.

```
$ klist

Ticket cache: FILE:/tmp/krb5cc_5441
Default principal: spark/kyuubi.host.name@KYUUBI.APACHE.ORG

Valid starting       Expires              Service principal
2020-11-25T13:17:18  2020-11-26T13:17:18  krbtgt/KYUUBI.APACHE.ORG@KYUUBI.APACHE.ORG
        renew until 2020-12-02T13:17:18
```

Kerberos credentials can be stored in Kerberos ticket cache. For example, `/tmp/krb5cc_5441` in the above case.

They are valid for relatively short period. So, we always need to refresh it for long-running services like Kyuubi.

Configurations

When working with a Kerberos-enabled Hadoop cluster, we should ensure that `hadoop.security.authentication` is set to `KERBEROS` in `$HADOOP_CONF_DIR/core-site.xml` or `$KYUUBI_HOME/conf/kyuubi-defaults.conf`. Then we need to specify `kyuubi.kinit.principal` and `kyuubi.kinit.keytab` for authentication.

For example,

```
kyuubi.kinit.principal=spark/kyuubi.apache.org@KYUUBI.APACHE.ORG
kyuubi.kinit.keytab=/path/to/kyuuib.keytab
```

Note:`kyuubi.kinit.principal` must be in the format: `<user>/<host>@<realm>`, and `<host>` must be a FQDN of the host Kyuubi is running.

Kyuubi will use this `principal` to impersonate client users, so the cluster should enable it to do impersonation for some particular user from some particular hosts.

For example,

```
hadoop.proxyuser.<user name in principal>.groups *
hadoop.proxyuser.<user name in principal>.hosts *
```

Further Readings

- Hadoop in Secure Mode
- Use Kerberos for authentication in Spark

3.3.4 Hadoop Credentials Manager

In order to pass the authentication of a kerberos secured hadoop cluster, kyuubi currently submits engines in two ways:

1. Submits with current kerberos user and extra `SparkSubmit` argument `--proxy-user`.
2. Submits with `spark.kerberos.principal` and `spark.kerberos.keytab` specified.

If engine is submitted with `--proxy-user` specified, its delegation tokens of hadoop cluster services are obtained by current kerberos user and can not be renewed by itself.Thus, engine's lifetime is limited by the lifetime of delegation tokens.To remove this limitation, kyuubi renews delegation tokens at server side in Hadoop Credentials Manager.

Engine submitted with principal and keytab can renew delegation tokens by itself. But for implementation simplicity, kyuubi server will also renew delegation tokens for it.

Configurations

Cluster Services

Kyuubi currently supports renew delegation tokens of Hadoop filesystems and Hive metastore servers.

Hadoop client configurations

Set `HADOOP_CONF_DIR` in `$KYUUBI_HOME/conf/kyuubi-env.sh` if it hasn't been set yet, e.g.

```
$ echo "export HADOOP_CONF_DIR=/path/to/hadoop/conf" >> $KYUUBI_HOME/conf/kyuubi-env.sh
```

Extra Hadoop filesystems can be specified in `$KYUUBI_HOME/conf/kyuubi-defaults.conf` by `kyuubi.credentials.hadoopfs.uris` in comma separated list.

Hive metastore configurations

Via kyuubi-defaults.conf

Specify Hive metastore configurations In `$KYUUBI_HOME/conf/kyuubi-defaults.conf`. Hadoop Credentials Manager will load the configurations when initialized.

Via hive-site.xml

Place your copy of `hive-site.xml` into `$KYUUBI_HOME/conf`, Kyuubi will load this config file to its classpath.

This version of configuration has lower priority than those in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`.

Via JDBC Connection URL

Hive configurations specified in JDBC connection URL are ignored by Hadoop Credentials Manager as Hadoop Credentials Manager is initialized when Kyuubi server starts.

Credentials Renewal

Required Security Configs

The necessary configurations for hdfs and hive to obtain delegation token are as follows:

3.4 Monitoring

In this section, you will learn how to monitor Kyuubi with logging, metrics etc..

3.4.1 Monitoring Kyuubi - Logging System

Kyuubi uses Apache Log4j2 for logging since version v1.5.0. For versions v1.4.1 and below, it uses Apache Log4j.

In general, there are mainly three components in the Kyuubi architecture that will produce component-oriented logs to help you trace breadcrumbs for SQL workloads against Kyuubi.

- Logs of Kyuubi Server
- Logs of Kyuubi Engines
- Operation logs

In addition, a Kyuubi deployment for production usually relies on some other external systems. For example, both Kyuubi servers and engines will use Apache Zookeeper for service discovery. The instructions for external system loggings will not be included in this article.

Logs of Kyuubi Server

Logs of Kyuubi Server show us the activities of the server instance including how start/stop, how does it response client requests, etc.

Configuring Server Logging

Basic Configurations

You can configure it by adding a `log4j2.xml` file in the `$KYUUBI_HOME/conf` directory. One way to start is to make a copy of the existing `log4j2.xml.template` located there.

For example,

```
# cd $KYUUBI_HOME
cp conf/log4j2.xml.template conf/log4j2.xml
```

With or without the above step, by default the server logging will redirect the logs to a file named `kyuubi-${env:USER}-org.apache.kyuubi.server.KyuubiServer-${env:HOSTNAME}.out` under the directory of `$KYUUBI_HOME/logs`.

For example, you can easily find where the server log goes when staring a Kyuubi server from the console output.

```
$ export SPARK_HOME=/Users/kentyao/Downloads/spark/spark-3.2.0-bin-hadoop3.2
$ cd ~/svn-kyuubi/v1.3.1-incubating-rc0/apache-kyuubi-1.3.1-incubating-bin
$ bin/kyuubi start
```

```
Starting Kyuubi Server from /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/apache-
↪kyuubi-1.3.1-incubating-bin
Warn: Not find kyuubi environment file /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/
↪apache-kyuubi-1.3.1-incubating-bin/conf/kyuubi-env.sh, using default ones...
JAVA_HOME: /Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home
KYUUBI_HOME: /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/apache-kyuubi-1.3.1-
↪incubating-bin
KYUUBI_CONF_DIR: /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/apache-kyuubi-1.3.1-
↪incubating-bin/conf
KYUUBI_LOG_DIR: /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/apache-kyuubi-1.3.1-
↪incubating-bin/logs
KYUUBI_PID_DIR: /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/apache-kyuubi-1.3.1-
↪incubating-bin/pid
KYUUBI_WORK_DIR_ROOT: /Users/kentyao/svn-kyuubi/v1.3.1-incubating-rc0/apache-kyuubi-1.3.
↪1-incubating-bin/work
SPARK_HOME: /Users/kentyao/Downloads/spark/spark-3.2.0-bin-hadoop3.2
SPARK_CONF_DIR: /Users/kentyao/Downloads/spark/spark-3.2.0-bin-hadoop3.2/conf
HADOOP_CONF_DIR:
YARN_CONF_DIR:
Starting org.apache.kyuubi.server.KyuubiServer, logging to /Users/kentyao/svn-kyuubi/v1.
↪3.1-incubating-rc0/apache-kyuubi-1.3.1-incubating-bin/logs/kyuubi-kentyao-org.apache.
↪kyuubi.server.KyuubiServer-hulk.local.out
Welcome to
  __  __                           __
 /\ \/\ \                         /\ \       __
 \ \ \/'/'  __  __  __  __  __  __\ \ \____/\_\
  \ \ , <  /\ \/\ \/\ \/\ \/\ \/\ \\ \ '__`\/\ \
   \ \ \\`\\ \ \_\ \ \ \_\ \ \ \_\ \\ \ \L\ \ \ \
    \ \_\ \_\/`____ \ \____/\ \____/ \ \_,__/\ \_\
     \/_/\/_/`/___/> \/___/  \/___/   \/___/  \/_/
                /\___/
                \/__/
```

KYUUBI_LOG_DIR

You may also notice that there is an environment variable called KYUUBI_LOG_DIR in the above example.

KYUUBI_LOG_DIR determines which folder we want to put our server log files.

For example, the below command will locate the log files to /Users/kentyao/tmp.

```
$ mkdir /Users/kentyao/tmp
$ KYUUBI_LOG_DIR=/Users/kentyao/tmp bin/kyuubi start
```

```
Starting org.apache.kyuubi.server.KyuubiServer, logging to /Users/kentyao/tmp/kyuubi-
↪kentyao-org.apache.kyuubi.server.KyuubiServer-hulk.local.out
```

KYUUBI_MAX_LOG_FILES

KYUUBI_MAX_LOG_FILES controls how many log files will be remained after a Kyuubi server reboots.

Custom Log4j2 Settings

Taking control of $KYUUBI_HOME/conf/log4j2.xml will also give us the ability of customizing server logging as we want.

For example, we can disable the console appender and enable the file appender like,

```
<Configuration status="INFO">
  <Appenders>
    <File name="fa" fileName="log/dummy.log">
      <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} %p %c: %m%n"/>
      <Filters>
        <RegexFilter regex=".*Thrift error occurred during processing of message.*"␣
↪onMatch="DENY" onMismatch="NEUTRAL"/>
      </Filters>
    </File>
  </Appenders>
  <Loggers>
    <Root level="INFO">
      <AppenderRef ref="fa"/>
    </Root>
  </Loggers>
</Configuration>
```

Then everything goes to log/dummy.log.

Logs of Spark SQL Engine

Spark SQL Engine is one type of Kyuubi Engines and also a typical Spark application. Thus, its logs mainly contain the logs of a Spark Driver. Meanwhile, it also includes how all the services of an engine start/stop, how does it response the incoming calls from Kyuubi servers, etc.

In general, when an exception occurs, we are able to find more information and clues in the engine's logs.

Configuring Engine Logging

Please refer to Apache Spark online documentation -Configuring Logging for instructions.

Where to Find the Engine Log

The engine logs locate differently based on the deploy mode and the cluster manager. When using local backend or `client` deploy mode for other cluster managers, such as YARN, you can find the whole engine log in `$KYUUBI_WORK_DIR_ROOT/${session username}/kyuubi-spark-sql-engine.log.${num}`. Different session users have different folders to group all live and historical engine logs. Each engine will have one and only engine log. When using `cluster` deploy mode, the local engine logs only contain very little information, the main parts of engine logs are on the remote driver side, e.g. for YARN cluster, they are in ApplicationMasters' log.

Logs of Flink SQL Engine

Flink SQL Engine is one type of Kyuubi Engines and also a typical Flink application. Thus, its logs mainly contain the logs of a Flink JobManager and TaskManager. Meanwhile, it also includes how all the services of an engine start/stop, how does it response the incoming calls from Kyuubi servers, etc.

In general, when an exception occurs, we are able to find more information and clues in the engine's logs.

Configuring Engine Logging

Please refer to Apache Flink online documentation -Configuring Logging for instructions.

Where to Find the Engine Log

The engine logs locate differently based on the deploy mode and the cluster manager. When using local backend or `client` deploy mode for other cluster managers, such as YARN, you can find the whole engine log in `$KYUUBI_WORK_DIR_ROOT/${session username}/kyuubi-flink-sql-engine.log.${num}`. Different session users have different folders to group all live and historical engine logs. Each engine will have one and only engine log. When using `cluster` deploy mode, the local engine logs only contain very little information, the main parts of engine logs are on the remote driver side, e.g. for YARN cluster, they are in ApplicationMasters' log.

Operation Logs

Operation log will show how SQL queries are executed, such as query planning, execution, and statistic reports.

Operation logs can reveal directly to end-users how their queries are being executed on the server/engine-side, including some process-oriented information, and why their queries are slow or in error.

For example, when you, as an end-user, use `beeline` to connect a Kyuubi server and execute query like below.

```
bin/beeline -u 'jdbc:hive2://10.242.189.214:2181/;serviceDiscoveryMode=zooKeeper;
↪zooKeeperNamespace=kyuubi' -n kent -e 'select * from src;'
```

You will both get the final results and the corresponding operation logs telling you the journey of the query.

```
0: jdbc:hive2://10.242.189.214:2181/> select * from src;
2021-10-27 17:00:19.399 INFO operation.ExecuteStatement: Processing kent's␣
↪query[fb5f57d2-2b50-4a46-961b-3a5c6a2d2597]: INITIALIZED_STATE -> PENDING_STATE,␣
↪statement: select * from src
2021-10-27 17:00:19.401 INFO operation.ExecuteStatement: Processing kent's␣
↪query[fb5f57d2-2b50-4a46-961b-3a5c6a2d2597]: PENDING_STATE -> RUNNING_STATE,␣
↪statement: select * from src
2021-10-27 17:00:19.400 INFO operation.ExecuteStatement: Processing kent's␣
↪query[26e169a2-6c06-450a-b758-e577ac673d70]: INITIALIZED_STATE -> PENDING_STATE,␣
↪statement: select * from src
2021-10-27 17:00:19.401 INFO operation.ExecuteStatement: Processing kent's␣
↪query[26e169a2-6c06-450a-b758-e577ac673d70]: PENDING_STATE -> RUNNING_STATE,␣
↪statement: select * from src
2021-10-27 17:00:19.402 INFO operation.ExecuteStatement:
           Spark application name: kyuubi_USER_kent_6d4b5e53-ddd2-420c-b04f-326fb2b17e18
                 application ID: local-1635318669122
                 application web UI: http://10.242.189.214:50250
                 master: local[*]
                 deploy mode: client
                 version: 3.2.0
           Start time: 2021-10-27T15:11:08.416
           User: kent
2021-10-27 17:00:19.408 INFO metastore.HiveMetaStore: 6: get_database: default
2021-10-27 17:00:19.408 INFO HiveMetaStore.audit: ugi=kent        ip=unknown-ip-
↪addr        cmd=get_database: default
2021-10-27 17:00:19.424 WARN conf.HiveConf: HiveConf of name hive.internal.ss.authz.
↪settings.applied.marker does not exist
2021-10-27 17:00:19.424 WARN conf.HiveConf: HiveConf of name hive.stats.jdbc.timeout␣
↪does not exist
2021-10-27 17:00:19.424 WARN conf.HiveConf: HiveConf of name hive.stats.retries.wait␣
↪does not exist
2021-10-27 17:00:19.424 INFO metastore.HiveMetaStore: 6: Opening raw store with␣
↪implementation class:org.apache.hadoop.hive.metastore.ObjectStore
2021-10-27 17:00:19.425 INFO metastore.ObjectStore: ObjectStore, initialize called
2021-10-27 17:00:19.430 INFO metastore.MetaStoreDirectSql: Using direct SQL, underlying␣
↪DB is DERBY
2021-10-27 17:00:19.431 INFO metastore.ObjectStore: Initialized ObjectStore
2021-10-27 17:00:19.434 INFO metastore.HiveMetaStore: 6: get_table : db=default tbl=src
2021-10-27 17:00:19.434 INFO HiveMetaStore.audit: ugi=kent        ip=unknown-ip-
↪addr        cmd=get_table : db=default tbl=src
2021-10-27 17:00:19.449 INFO metastore.HiveMetaStore: 6: get_table : db=default tbl=src
```

(continues on next page)

(continued from previous page)

```
2021-10-27 17:00:19.450 INFO HiveMetaStore.audit: ugi=kent        ip=unknown-ip-
↪addr        cmd=get_table : db=default tbl=src
2021-10-27 17:00:19.510 INFO operation.ExecuteStatement: Processing kent's␣
↪query[26e169a2-6c06-450a-b758-e577ac673d70]: RUNNING_STATE -> RUNNING_STATE,␣
↪statement: select * from src
2021-10-27 17:00:19.544 INFO memory.MemoryStore: Block broadcast_5 stored as values in␣
↪memory (estimated size 343.6 KiB, free 408.6 MiB)
2021-10-27 17:00:19.558 INFO memory.MemoryStore: Block broadcast_5_piece0 stored as␣
↪bytes in memory (estimated size 33.5 KiB, free 408.5 MiB)
2021-10-27 17:00:19.559 INFO spark.SparkContext: Created broadcast 5 from
2021-10-27 17:00:19.600 INFO mapred.FileInputFormat: Total input files to process : 1
2021-10-27 17:00:19.627 INFO spark.SparkContext: Starting job: collect at␣
↪ExecuteStatement.scala:97
2021-10-27 17:00:19.629 INFO kyuubi.SQLOperationListener: Query [26e169a2-6c06-450a-b758-
↪e577ac673d70]: Job 5 started with 1 stages, 1 active jobs running
2021-10-27 17:00:19.631 INFO kyuubi.SQLOperationListener: Query [26e169a2-6c06-450a-b758-
↪e577ac673d70]: Stage 5 started with 1 tasks, 1 active stages running
2021-10-27 17:00:19.713 INFO kyuubi.SQLOperationListener: Finished stage: Stage(5, 0);␣
↪Name: 'collect at ExecuteStatement.scala:97'; Status: succeeded; numTasks: 1; Took: 83␣
↪msec
2021-10-27 17:00:19.713 INFO scheduler.DAGScheduler: Job 5 finished: collect at␣
↪ExecuteStatement.scala:97, took 0.085454 s
2021-10-27 17:00:19.713 INFO scheduler.StatsReportListener: task runtime:(count: 1,␣
↪mean: 78.000000, stdev: 0.000000, max: 78.000000, min: 78.000000)
2021-10-27 17:00:19.713 INFO scheduler.StatsReportListener:         0%        5
↪%         10%         25%         50%         75%         90%         95%         100%
2021-10-27 17:00:19.713 INFO scheduler.StatsReportListener:         78.0 ms         78.0␣
↪ms         78.0 ms         78.0 ms         78.0 ms         78.0 ms         78.0␣
↪ms         78.0 ms         78.0 ms
2021-10-27 17:00:19.714 INFO scheduler.StatsReportListener: shuffle bytes␣
↪written:(count: 1, mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-27 17:00:19.714 INFO scheduler.StatsReportListener:         0%        5
↪%         10%         25%         50%         75%         90%         95%         100%
2021-10-27 17:00:19.714 INFO scheduler.StatsReportListener:         0.0 B         0.0␣
↪B         0.0 B         0.0 B         0.0 B         0.0 B         0.0 B         0.0␣
↪B         0.0 B
2021-10-27 17:00:19.714 INFO scheduler.StatsReportListener: fetch wait time:(count: 1,␣
↪mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-27 17:00:19.714 INFO scheduler.StatsReportListener:         0%        5
↪%         10%         25%         50%         75%         90%         95%         100%
2021-10-27 17:00:19.714 INFO scheduler.StatsReportListener:         0.0 ms         0.0␣
↪ms         0.0 ms         0.0 ms         0.0 ms         0.0 ms         0.0 ms         0.0␣
↪ms         0.0 ms
2021-10-27 17:00:19.715 INFO scheduler.StatsReportListener: remote bytes read:(count: 1,␣
↪mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-27 17:00:19.715 INFO scheduler.StatsReportListener:         0%        5
↪%         10%         25%         50%         75%         90%         95%         100%
2021-10-27 17:00:19.715 INFO scheduler.StatsReportListener:         0.0 B         0.0␣
↪B         0.0 B         0.0 B         0.0 B         0.0 B         0.0 B         0.0␣
↪B         0.0 B
2021-10-27 17:00:19.715 INFO scheduler.StatsReportListener: task result size:(count: 1,␣
↪mean: 1471.000000, stdev: 0.000000, max: 1471.000000, min: 1471.000000)
```

(continues on next page)

(continued from previous page)

```
2021-10-27 17:00:19.715 INFO scheduler.StatsReportListener:         0%        5
↪%        10%        25%        50%        75%        90%        95%        100%
2021-10-27 17:00:19.715 INFO scheduler.StatsReportListener:         1471.0 B        1471.
↪0 B        1471.0 B        1471.0 B        1471.0 B        1471.0 B        1471.0␣
↪B        1471.0 B        1471.0 B
2021-10-27 17:00:19.717 INFO scheduler.StatsReportListener: executor (non-fetch) time␣
↪pct: (count: 1, mean: 61.538462, stdev: 0.000000, max: 61.538462, min: 61.538462)
2021-10-27 17:00:19.717 INFO scheduler.StatsReportListener:         0%        5
↪%        10%        25%        50%        75%        90%        95%        100%
2021-10-27 17:00:19.717 INFO scheduler.StatsReportListener:         62 %        62
↪%        62 %        62 %        62 %        62 %        62 %        62 %        62 %
2021-10-27 17:00:19.718 INFO scheduler.StatsReportListener: fetch wait time pct: (count:␣
↪1, mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-27 17:00:19.718 INFO scheduler.StatsReportListener:         0%        5
↪%        10%        25%        50%        75%        90%        95%        100%
2021-10-27 17:00:19.718 INFO scheduler.StatsReportListener:         0 %         0
↪%         0 %         0 %         0 %         0 %         0 %         0 %         0 %
2021-10-27 17:00:19.718 INFO scheduler.StatsReportListener: other time pct: (count: 1,␣
↪mean: 38.461538, stdev: 0.000000, max: 38.461538, min: 38.461538)
2021-10-27 17:00:19.718 INFO scheduler.StatsReportListener:         0%        5
↪%        10%        25%        50%        75%        90%        95%        100%
2021-10-27 17:00:19.718 INFO scheduler.StatsReportListener:         38 %        38
↪%        38 %        38 %        38 %        38 %        38 %        38 %        38 %
2021-10-27 17:00:19.719 INFO kyuubi.SQLOperationListener: Query [26e169a2-6c06-450a-b758-
↪e577ac673d70]: Job 5 succeeded, 0 active jobs running
2021-10-27 17:00:19.728 INFO codegen.CodeGenerator: Code generated in 12.277091 ms
2021-10-27 17:00:19.729 INFO operation.ExecuteStatement: Processing kent's␣
↪query[26e169a2-6c06-450a-b758-e577ac673d70]: RUNNING_STATE -> FINISHED_STATE,␣
↪statement: select * from src, time taken: 0.328 seconds
2021-10-27 17:00:19.731 INFO operation.ExecuteStatement: Query[fb5f57d2-2b50-4a46-961b-
↪3a5c6a2d2597] in FINISHED_STATE
2021-10-27 17:00:19.731 INFO operation.ExecuteStatement: Processing kent's␣
↪query[fb5f57d2-2b50-4a46-961b-3a5c6a2d2597]: RUNNING_STATE -> FINISHED_STATE,␣
↪statement: select * from src, time taken: 0.33 seconds
+-------------------------------------------------+---------------------+
|                    version()                    | DATE '2021-10-27'   |
+-------------------------------------------------+---------------------+
| 3.2.0 5d45a415f3a29898d92380380cfd82bfc7f579ea  | 2021-10-27          |
+-------------------------------------------------+---------------------+
1 row selected (0.341 seconds)
```

Further Readings

- Monitoring Kyuubi - Events System
- Monitoring Kyuubi - Server Metrics
- Trouble Shooting
- Spark Online Documentation
  - Monitoring and Instrumentation

3.4.2 Monitoring Kyuubi - Server Metrics

Kyuubi has a configurable metrics system based on the Dropwizard Metrics Library. This allows users to report Kyuubi metrics to a variety of `kyuubi.metrics.reporters`. The metrics provide instrumentation for specific activities and Kyuubi server.

Configurations

The metrics system is configured via `$KYUUBI_HOME/conf/kyuubi-defaults.conf`.

Metrics

These metrics include:

Before v1.5.0, if you use these metrics:

- `kyuubi.statement.total`
- `kyuubi.statement.opened`
- `kyuubi.statement.failed.${errorType}`

Since v1.5.0, you can use the following metrics to replace:

- `kyuubi.operation.total.ExecuteStatement`
- `kyuubi.operation.opened.ExecuteStatement`
- `kyuubi.operation.failed.ExecuteStatement.${errorType}`

3.4.3 Trouble Shooting

Common Issues

java.lang.UnsupportedClassVersionError .. Unsupported major.minor version 52.0

```
Exception in thread "main" java.lang.UnsupportedClassVersionError: org/apache/kyuubi/
↪server/KyuubiServer : Unsupported major.minor version 52.0
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:803)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:142)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:442)
        at java.net.URLClassLoader.access$100(URLClassLoader.java:64)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:354)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:348)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:347)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:425)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:312)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:358)
        at sun.launcher.LauncherHelper.checkAndLoadMain(LauncherHelper.java:482)
```

Firstly, you should check the version of Java JRE used to run Kyuubi is actually matched with the version of Java compiler used to build Kyuubi.

```
$ java -version
java version "1.7.0_171"
OpenJDK Runtime Environment (rhel-2.6.13.2.el7-x86_64 u171-b01)
OpenJDK 64-Bit Server VM (build 24.171-b01, mixed mode)
```

```
$ cat RELEASE
Kyuubi 1.0.0-SNAPSHOT (git revision 39e5da5) built for
Java 1.8.0_251
Scala 2.12
Spark 3.0.1
Hadoop 2.7.4
Hive 2.3.7
Build flags:
```

To fix this problem you should export `JAVA_HOME` with a compatible one in `conf/kyuubi-env.sh`

```
echo "export JAVA_HOME=/path/to/jdk1.8.0_251" >> conf/kyuubi-env.sh
```

org.apache.spark.SparkException: When running with master 'yarn' either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment

```
Exception in thread "main" org.apache.spark.SparkException: When running with master
↪'yarn' either HADOOP_CONF_DIR or YARN_CONF_DIR must be set in the environment.
        at org.apache.spark.deploy.SparkSubmitArguments.error(SparkSubmitArguments.
↪scala:630)
        at org.apache.spark.deploy.SparkSubmitArguments.
↪validateSubmitArguments(SparkSubmitArguments.scala:270)
        at org.apache.spark.deploy.SparkSubmitArguments.
↪validateArguments(SparkSubmitArguments.scala:233)
        at org.apache.spark.deploy.SparkSubmitArguments.<init>(SparkSubmitArguments.
↪scala:119)
        at org.apache.spark.deploy.SparkSubmit$$anon$2$$anon$3.<init>(SparkSubmit.
↪scala:990)
        at org.apache.spark.deploy.SparkSubmit$$anon$2.parseArguments(SparkSubmit.
↪scala:990)
        at org.apache.spark.deploy.SparkSubmit.doSubmit(SparkSubmit.scala:85)
        at org.apache.spark.deploy.SparkSubmit$$anon$2.doSubmit(SparkSubmit.scala:1007)
        at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:1016)
        at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

When Kyuubi gets the `spark.master=yarn`, `HADOOP_CONF_DIR` should also be exported in `$KYUUBI_HOME/conf/kyuubi-env.sh`.

To fix this problem you should export `HADOOP_CONF_DIR` to the folder that contains the hadoop client settings in `conf/kyuubi-env.sh`.

```
echo "export HADOOP_CONF_DIR=/path/to/hadoop/conf" >> conf/kyuubi-env.sh
```

javax.security.sasl.SaslException: GSS initiate failed [Caused by GSSException: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)];

org.apache.hadoop.security.AccessControlException: Permission denied: user=hzyanqin, access=WRITE, inode="/user":hdfs:hdfs:drwxr-xr-x

```
org.apache.hadoop.security.AccessControlException: Permission denied: user=hzyanqin,␣
↪access=WRITE, inode="/user":hdfs:hdfs:drwxr-xr-x
        at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.
↪check(FSPermissionChecker.java:350)
        at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.
↪checkPermission(FSPermissionChecker.java:251)
        at org.apache.ranger.authorization.hadoop.RangerHdfsAuthorizer
↪$RangerAccessControlEnforcer.checkPermission(RangerHdfsAuthorizer.java:306)
        at org.apache.hadoop.hdfs.server.namenode.FSPermissionChecker.
↪checkPermission(FSPermissionChecker.java:189)
        at org.apache.hadoop.hdfs.server.namenode.FSDirectory.
↪checkPermission(FSDirectory.java:1767)
        at org.apache.hadoop.hdfs.server.namenode.FSDirectory.
↪checkPermission(FSDirectory.java:1751)
        at org.apache.hadoop.hdfs.server.namenode.FSDirectory.
↪checkAncestorAccess(FSDirectory.java:1710)
        at org.apache.hadoop.hdfs.server.namenode.FSDirMkdirOp.mkdirs(FSDirMkdirOp.
↪java:60)
        at org.apache.hadoop.hdfs.server.namenode.FSNamesystem.mkdirs(FSNamesystem.
↪java:3062)
        at org.apache.hadoop.hdfs.server.namenode.NameNodeRpcServer.
↪mkdirs(NameNodeRpcServer.java:1156)
        at org.apache.hadoop.hdfs.protocolPB.
↪ClientNamenodeProtocolServerSideTranslatorPB.
↪mkdirs(ClientNamenodeProtocolServerSideTranslatorPB.java:652)
        at org.apache.hadoop.hdfs.protocol.proto.ClientNamenodeProtocolProtos
↪$ClientNamenodeProtocol$2.callBlockingMethod(ClientNamenodeProtocolProtos.java)
        at org.apache.hadoop.ipc.ProtobufRpcEngine$Server$ProtoBufRpcInvoker.
↪call(ProtobufRpcEngine.java:503)
        at org.apache.hadoop.ipc.RPC$Server.call(RPC.java:989)
        at org.apache.hadoop.ipc.Server$RpcCall.run(Server.java:871)
        at org.apache.hadoop.ipc.Server$RpcCall.run(Server.java:817)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.
↪java:1893)
        at org.apache.hadoop.ipc.Server$Handler.run(Server.java:2606)

        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.
↪newInstance(NativeConstructorAccessorImpl.java:62)
        at sun.reflect.DelegatingConstructorAccessorImpl.
↪newInstance(DelegatingConstructorAccessorImpl.java:45)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
        at org.apache.hadoop.ipc.RemoteException.instantiateException(RemoteException.
↪java:106)
```

(continues on next page)

(continued from previous page)

```
        at org.apache.hadoop.ipc.RemoteException.unwrapRemoteException(RemoteException.
↪java:73)
        at org.apache.hadoop.hdfs.DFSClient.primitiveMkdir(DFSClient.java:3007)
        at org.apache.hadoop.hdfs.DFSClient.mkdirs(DFSClient.java:2975)
        at org.apache.hadoop.hdfs.DistributedFileSystem$21.doCall(DistributedFileSystem.
↪java:1047)
        at org.apache.hadoop.hdfs.DistributedFileSystem$21.doCall(DistributedFileSystem.
↪java:1043)
        at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.
↪java:81)
        at org.apache.hadoop.hdfs.DistributedFileSystem.
↪mkdirsInternal(DistributedFileSystem.java:1061)
        at org.apache.hadoop.hdfs.DistributedFileSystem.mkdirs(DistributedFileSystem.
↪java:1036)
        at org.apache.hadoop.fs.FileSystem.mkdirs(FileSystem.java:1881)
        at org.apache.hadoop.fs.FileSystem.mkdirs(FileSystem.java:600)
        at org.apache.spark.deploy.yarn.Client.prepareLocalResources(Client.scala:441)
        at org.apache.spark.deploy.yarn.Client.createContainerLaunchContext(Client.
↪scala:876)
        at org.apache.spark.deploy.yarn.Client.submitApplication(Client.scala:196)
        at org.apache.spark.scheduler.cluster.YarnClientSchedulerBackend.
↪start(YarnClientSchedulerBackend.scala:60)
        at org.apache.spark.scheduler.TaskSchedulerImpl.start(TaskSchedulerImpl.
↪scala:201)
        at org.apache.spark.SparkContext.<init>(SparkContext.scala:555)
        at org.apache.spark.SparkContext$.getOrCreate(SparkContext.scala:2574)
        at org.apache.spark.sql.SparkSession$Builder.$anonfun$getOrCreate$2(SparkSession.
↪scala:934)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.SparkSession$Builder.getOrCreate(SparkSession.scala:928)
        at org.apache.kyuubi.engine.spark.SparkSQLEngine$.createSpark(SparkSQLEngine.
↪scala:72)
        at org.apache.kyuubi.engine.spark.SparkSQLEngine$.main(SparkSQLEngine.scala:101)
        at org.apache.kyuubi.engine.spark.SparkSQLEngine.main(SparkSQLEngine.scala)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
↪java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.spark.deploy.JavaMainApplication.start(SparkApplication.scala:52)
        at org.apache.spark.deploy.SparkSubmit.org$apache$spark$deploy$SparkSubmit$
↪$runMain(SparkSubmit.scala:928)
        at org.apache.spark.deploy.SparkSubmit$$anon$1.run(SparkSubmit.scala:165)
        at org.apache.spark.deploy.SparkSubmit$$anon$1.run(SparkSubmit.scala:163)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.
↪java:1746)
        at org.apache.spark.deploy.SparkSubmit.doRunMain$1(SparkSubmit.scala:163)
        at org.apache.spark.deploy.SparkSubmit.submit(SparkSubmit.scala:203)
        at org.apache.spark.deploy.SparkSubmit.doSubmit(SparkSubmit.scala:90)
        at org.apache.spark.deploy.SparkSubmit$$anon$2.doSubmit(SparkSubmit.scala:1007)
```

(continues on next page)

(continued from previous page)

```
        at org.apache.spark.deploy.SparkSubmit$.main(SparkSubmit.scala:1016)
        at org.apache.spark.deploy.SparkSubmit.main(SparkSubmit.scala)
```

The user do not have permission to create to Hadoop home dir, which is `/user/hzyanqin` in the case above.

To fix this problem you need to create this directory first and grant ACL permission for `hzyanqin`.

org.apache.thrift.TApplicationException: Invalid method name: 'get_table_req'

```
Caused by: org.apache.thrift.TApplicationException: Invalid method name: 'get_table_req'
        at org.apache.thrift.TServiceClient.receiveBase(TServiceClient.java:79)
        at org.apache.hadoop.hive.metastore.api.ThriftHiveMetastore$Client.recv_get_
↪table_req(ThriftHiveMetastore.java:1567)
        at org.apache.hadoop.hive.metastore.api.ThriftHiveMetastore$Client.get_table_
↪req(ThriftHiveMetastore.java:1554)
        at org.apache.hadoop.hive.metastore.HiveMetaStoreClient.
↪getTable(HiveMetaStoreClient.java:1350)
        at org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClient.
↪getTable(SessionHiveMetaStoreClient.java:127)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
↪java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.hive.metastore.RetryingMetaStoreClient.
↪invoke(RetryingMetaStoreClient.java:173)
        at com.sun.proxy.$Proxy37.getTable(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.
↪java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.hive.metastore.HiveMetaStoreClient$SynchronizedHandler.
↪invoke(HiveMetaStoreClient.java:2336)
        at com.sun.proxy.$Proxy37.getTable(Unknown Source)
        at org.apache.hadoop.hive.ql.metadata.Hive.getTable(Hive.java:1274)
        ... 93 more
```

This error means that you are using incompatible version of Hive metastore client to connect the Hive metastore server.

To fix this problem you could use a compatible version of Hive client by configuring `spark.sql.hive.metastore.jars` and `spark.sql.hive.metastore.version` at Spark side.

hive.server2.thrift.max.worker.threads

```
Unexpected end of file when reading from HS2 server. The root cause might be too many␣
↪concurrent connections. Please ask the administrator to check the number of active␣
↪connections, and adjust hive.server2.thrift.max.worker.threads if applicable.
Error: org.apache.thrift.transport.TTransportException (state=08S01,code=0)
```

In Kyuubi, we should increase `kyuubi.frontend.min.worker.threads` instead of `hive.server2.thrift.max.worker.threads`

Failed to create function using jar

```
CREATE TEMPORARY FUNCTION TEST AS 'com.netease.UDFTest' using jar 'hdfs:///tmp/udf.jar'
```

```
Error operating EXECUTE_STATEMENT: org.apache.spark.sql.AnalysisException: Can not load␣
↪class 'com.netease.UDFTest' when registering the function 'test', please make sure it␣
↪is on the classpath;
        at org.apache.spark.sql.catalyst.catalog.SessionCatalog.$anonfun$registerFunction
↪$1(SessionCatalog.scala:1336)
        at scala.Option.getOrElse(Option.scala:189)
        at org.apache.spark.sql.catalyst.catalog.SessionCatalog.
↪registerFunction(SessionCatalog.scala:1333)
        at org.apache.spark.sql.execution.command.CreateFunctionCommand.run(functions.
↪scala:82)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.sideEffectResult
↪$lzycompute(commands.scala:70)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.
↪sideEffectResult(commands.scala:68)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.
↪executeCollect(commands.scala:79)
        at org.apache.spark.sql.Dataset.$anonfun$logicalPlan$1(Dataset.scala:229)
        at org.apache.spark.sql.Dataset.$anonfun$withAction$1(Dataset.scala:3618)
        at org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId
↪$5(SQLExecution.scala:100)
        at org.apache.spark.sql.execution.SQLExecution$.
↪withSQLConfPropagated(SQLExecution.scala:160)
        at org.apache.spark.sql.execution.SQLExecution$.$anonfun$withNewExecutionId
↪$1(SQLExecution.scala:87)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:764)
        at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.
↪scala:64)
        at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3616)
        at org.apache.spark.sql.Dataset.<init>(Dataset.scala:229)
        at org.apache.spark.sql.Dataset$.$anonfun$ofRows$2(Dataset.scala:100)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:764)
        at org.apache.spark.sql.Dataset$.ofRows(Dataset.scala:97)
        at org.apache.spark.sql.SparkSession.$anonfun$sql$1(SparkSession.scala:607)
        at org.apache.spark.sql.SparkSession.withActive(SparkSession.scala:764)
        at org.apache.spark.sql.SparkSession.sql(SparkSession.scala:602)
        at org.apache.kyuubi.engine.spark.operation.ExecuteStatement.org$apache$kyuubi
↪$engine$spark$operation$ExecuteStatement$$executeStatement(ExecuteStatement.scala:64)
        at org.apache.kyuubi.engine.spark.operation.ExecuteStatement$$anon$1.
↪run(ExecuteStatement.scala:80)
```

(continues on next page)

(continued from previous page)

```
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.
↪java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.
↪java:617)
        at java.lang.Thread.run(Thread.java:745)
```

If you get this exception when creating a function, you can check your JDK version. You should update JDK to JDK1.8.0_121 and later, since JDK1.8.0_121 fix a security issue Additional access restrictions for URLClassLoader.newInstance.

Failed to start Spark 3.1 with error msg 'Cannot modify the value of a Spark config'

Here is the error message

```
Caused by: org.apache.spark.sql.AnalysisException: Cannot modify the value of a Spark␣
↪config: spark.yarn.queue
        at org.apache.spark.sql.RuntimeConfig.requireNonStaticConf(RuntimeConfig.
↪scala:156)
        at org.apache.spark.sql.RuntimeConfig.set(RuntimeConfig.scala:40)
        at org.apache.kyuubi.engine.spark.session.SparkSQLSessionManager.$anonfun
↪$openSession$2(SparkSQLSessionManager.scala:68)
        at org.apache.kyuubi.engine.spark.session.SparkSQLSessionManager.$anonfun
↪$openSession$2$adapted(SparkSQLSessionManager.scala:56)
        at scala.collection.immutable.Map$Map4.foreach(Map.scala:236)
        at org.apache.kyuubi.engine.spark.session.SparkSQLSessionManager.
↪openSession(SparkSQLSessionManager.scala:56)
        ... 12 more
```

This is because Spark-3.1 will check the config which you set and throw exception if the config is static or used in other module (e.g. yarn/core).

You can add a config `spark.sql.legacy.setCommandRejectsSparkCoreConfs=false` in `spark-defaults.conf` to disable this behavior.

3.5 Tools

3.5.1 Kubernetes Tools Spark Block Cleaner

Requirements

You'd better have cognition upon the following things when you want to use spark-block-cleaner.

- Read this article
- An active Kubernetes cluster
- Kubectl
- Docker

Scenes

When you're using Spark On Kubernetes with Client mode and don't use `emptyDir` for Spark `local-dir` type, you may face the same scenario that executor pods deleted without clean all the Block files. It may cause disk overflow.

Therefore, we chose to use Spark Block Cleaner to clear the block files accumulated by Spark.

Principle

When deploying Spark Block Cleaner, we will configure volumes for the destination folder. Spark Block Cleaner will perceive the folder by the parameter `CACHE_DIRS`.

Spark Block Cleaner will clear the perceived folder in a fixed loop(which can be configured by `SCHEDULE_INTERVAL`). And Spark Block Cleaner will select folder start with `blockmgr` and `spark` for deletion using the logic Spark uses to create those folders.

Before deleting those files, Spark Block Cleaner will determine whether it is a recently modified file(depending on whether the file has not been acted on within the specified time which configured by `FILE_EXPIRED_TIME`). Only delete files those beyond that time interval.

And Spark Block Cleaner will check the disk utilization after clean, if the remaining space is less than the specified value(control by `FREE_SPACE_THRESHOLD`), will trigger deep clean(which file expired time control by `DEEP_CLEAN_FILE_EXPIRED_TIME`).

Usage

Before you start using Spark Block Cleaner, you should build its docker images.

Build Block Cleaner Docker Image

In the `KYUUBI_HOME` directory, you can use the following cmd to build docker image.

```
docker build ./tools/spark-block-cleaner/kubernetes/docker
```

Modify spark-block-cleaner.yml

You need to modify the `${KYUUBI_HOME}/tools/spark-block-cleaner/kubernetes/spark-block-cleaner.yml` to fit your current environment.

In Kyuubi tools, we recommend using `DaemonSet` to start, and we offer default yaml file in daemonSet way.

Base file structure:

```
apiVersion
kind
metadata
  name
  namespace
spec
  select
  template
    metadata
    spce
```

(continues on next page)

(continued from previous page)

```
      containers
      - image
      - volumeMounts
      - env
    volumes
```

You can use affect the performance of Spark Block Cleaner through configure parameters in containers env part of `spark-block-cleaner.yml`.

```
env:
  - name: CACHE_DIRS
    value: /data/data1,/data/data2
  - name: FILE_EXPIRED_TIME
    value: 604800
  - name: DEEP_CLEAN_FILE_EXPIRED_TIME
    value: 432000
  - name: FREE_SPACE_THRESHOLD
    value: 60
  - name: SCHEDULE_INTERVAL
    value: 3600
```

The most important thing, configure volumeMounts and volumes corresponding to Spark local-dirs.

For example, Spark use /spark/shuffle1 as local-dir, you can configure like:

```
volumes:
  - name: block-files-dir-1
    hostPath:
      path: /spark/shuffle1
```

```
volumeMounts:
  - name: block-files-dir-1
    mountPath: /data/data1
```

```
env:
  - name: CACHE_DIRS
    value: /data/data1
```

Start daemonSet

After you finishing modifying the above, you can use the following command `kubectl apply -f ${KYUUBI_HOME}/tools/spark-block-cleaner/kubernetes/spark-block-cleaner.yml` to start daemonSet.

Related parameters

3.5.2 Managing kyuubi servers and engines Tool

Usage

```
bin/kyuubi-ctl --help
```

Output

```
kyuubi 1.6.0-SNAPSHOT
Usage: kyuubi-ctl [create|get|delete|list] [options]

  -zk, --zk-quorum <value>
                           The connection string for the zookeeper ensemble, using zk␣
↪quorum manually.
  -n, --namespace <value>  The namespace, using kyuubi-defaults/conf if absent.
  -s, --host <value>       Hostname or IP address of a service.
  -p, --port <value>       Listening port of a service.
  -v, --version <value>    Using the compiled KYUUBI_VERSION default, change it if the␣
↪active service is running in another.
  -b, --verbose            Print additional debug output.

Command: create [server]

Command: create server
        Expose Kyuubi server instance to another domain.

Command: get [server|engine] [options]
        Get the service/engine node info, host and port needed.
Command: get server
        Get Kyuubi server info of domain
Command: get engine
        Get Kyuubi engine info belong to a user.
  -u, --user <value>       The user name this engine belong to.
  -et, --engine-type <value>
                           The engine type this engine belong to.
  -es, --engine-subdomain <value>
                           The engine subdomain this engine belong to.
  -esl, --engine-share-level <value>
                           The engine share level this engine belong to.

Command: delete [server|engine] [options]
        Delete the specified service/engine node, host and port needed.
Command: delete server
        Delete the specified service node for a domain
```

(continues on next page)

(continued from previous page)

```
Command: delete engine
        Delete the specified engine node for user.
  -u, --user <value>       The user name this engine belong to.
  -et, --engine-type <value>
                           The engine type this engine belong to.
  -es, --engine-subdomain <value>
                           The engine subdomain this engine belong to.
  -esl, --engine-share-level <value>
                           The engine share level this engine belong to.

Command: list [server|engine] [options]
        List all the service/engine nodes for a particular domain.
Command: list server
        List all the service nodes for a particular domain
Command: list engine
        List all the engine nodes for a user
  -u, --user <value>       The user name this engine belong to.
  -et, --engine-type <value>
                           The engine type this engine belong to.
  -es, --engine-subdomain <value>
                           The engine subdomain this engine belong to.
  -esl, --engine-share-level <value>
                           The engine share level this engine belong to.

  -h, --help               Show help message and exit.
```

Manage kyuubi servers

You can specify the zookeeper address(`--zk-quorum`) and namespace(`--namespace`), version(`--version`) parameters to query a specific kyuubi server cluster.

List server

List all the service nodes for a particular domain.

```
bin/kyuubi-ctl list server
```

Create server

Expose Kyuubi server instance to another domain.

First read `kyuubi.ha.zookeeper.namespace` in `conf/kyuubi-defaults.conf`, if there are server instances under this namespace, register them in the new namespace specified by the `--namespace` parameter.

```
bin/kyuubi-ctl create server --namespace XXX
```

Get server

Get Kyuubi server info of domain.

```
bin/kyuubi-ctl get server --host XXX --port YYY
```

Delete server

Delete the specified service node for a domain.

After the server node is deleted, the kyuubi server stops opening new sessions and waits for all currently open sessions to be closed before the process exits.

```
bin/kyuubi-ctl delete server --host XXX --port YYY
```

Manage kyuubi engines

You can also specify the engine type(`--engine-type`), engine share level subdomain(`--engine-subdomain`) and engine share level(`--engine-share-level`).

If not specified, the configuration item `kyuubi.engine.type` of `kyuubi-defaults.conf` read, the default value is `SPARK_SQL`, `kyuubi.engine.share.level.subdomain`, the default value is `default`, `kyuubi.engine.share.level`, the default value is `USER`.

If the engine pool mode is enabled through `kyuubi.engine.pool.size`, the subdomain consists of `kyuubi.engine.pool.name` and a number below size, e.g. `engine-pool-0` .

`--engine-share-level` supports the following enum values.

- CONNECTION

The engine Ref Id (UUID) must be specified via `--engine-subdomain`.

- USER:

Default Value.

- GROUP:

The `--user` parameter is the group name corresponding to the user.

- SERVER:

The `--user` parameter is the user who started the kyuubi server.

List engine

List all the engine nodes for a user.

```
bin/kyuubi-ctl list engine --user AAA
```

The management share level is SERVER, the user who starts the kyuubi server is A, the engine is TRINO, and the subdomain is adhoc.

```
bin/kyuubi-ctl list engine --user A --engine-type TRINO --engine-subdomain adhoc --
↪engine-share-level SERVER
```

Get engine

Get Kyuubi engine info belong to a user.

```
bin/kyuubi-ctl get engine --user AAA --host XXX --port YYY
```

Delete engine

Delete the specified engine node for user.

After the engine node is deleted, the kyuubi engine stops opening new sessions and waits for all currently open sessions to be closed before the process exits.

```
bin/kyuubi-ctl delete engine --user AAA --host XXX --port YYY
```

3.5.3 Kyuubi Administer Tool

New in version 1.6.0.

Kyuubi administer tool(kyuubi-admin) provides administrators with some maintenance operations against a kyuubi server or cluster.

Installation

To install kyuubi-admin, you need to unpack the tarball. For example,

`tar zxf apache-kyuubi-1.7.0-bin.tgz`

This will result in the creation of a subdirectory named apache-kyuubi-1.7.0-bin shown below,

```
apache-kyuubi-1.7.0-bin
├── ...
├── bin
|   ├── kyuubi-admin
|   ├── ...
├── ...
```

Usage

```
bin/kyuubi-admin --help
```

Refresh config

Refresh the config with specified type.

Usage: `bin/kyuubi-admin refresh config [options] [<configType>]`

Config Type	Description
hadoopConf	The hadoop conf used for proxy user verification.
userDefaultsConf	The user defaults configs with key in format in the form of *___{username}___.{config key}* from default property file.
unlimitedUsers	The users without maximum connections limitation.

List Engines

Prints a table of the key information about the specified engines.

Usage: `bin/kyuubi-admin list engine [options]`

Options	Description
-et, –engine-type	The engine type. If not specified, it will read the configuration item kyuubi.engine.type from kyuubi-defaults.conf.
-esl, –engine-share-level	The engine share level. If not specified, it will read the configuration item kyuubi.engine.share.level from kyuubi-defaults.conf.
-es, –engine-subdomain	The subdomain for the share level of an engine. If not specified, it will read the configuration item kyuubi.engine.share.level.subdomain from kyuubi-defaults.conf.
–hs2ProxyUser	The proxy user to impersonate. When specified, it will list engines for the hs2ProxyUser.

Delete an Engine

Delete the specified engine.

Usage: `bin/kyuubi-admin delete engine [options]`

Options	Description
-et, –engine-type	The engine type. If not specified, it will read the configuration item kyuubi.engine.type from kyuubi-defaults.conf.
-esl, –engine-share-level	The engine share level. If not specified, it will read the configuration item kyuubi.engine.share.level from kyuubi-defaults.conf.
-es, –engine-subdomain	The subdomain for the share level of an engine. If not specified, it will read the configuration item kyuubi.engine.share.level.subdomain from kyuubi-defaults.conf. Default value is “default”.
–hs2ProxyUser	The proxy user to impersonate. When specified, it will delete engines for the hs2ProxyUser.

3.6 Clients

This section aims to document the APIs, clients and tools for end-users who are not necessary to care about deployment at the kyuubi server side.

Kyuubi provides standards-based drivers for JDBC, and ODBC enabling developers to build database applications in their language of choice.

In addition, APIs like REST, Thrift, etc., allow developers to access kyuubi directly and flexibly.

Note: When you try some of the examples in this section, make sure you have a available server.

3.6.1 JDBC Drivers

Kyuubi Hive JDBC Driver

New in version 1.4.0: Since 1.4.0, kyuubi community maintains a forked hive jdbc driver module and provides both shaded and non-shaded packages.

This packages aims to support some missing functionalities of the original hive jdbc. For kyuubi engines that support multiple catalogs, it provides meta APIs for better support. The behaviors of the original hive jdbc have remained.

To access a Hive data warehouse or new lakehouse formats, such as Apache Iceberg/Hudi, delta lake using the kyuubi jdbc driver for Apache kyuubi, you need to configure the following:

- The list of driver library files - *Referencing the JDBC Driver Libraries*.
- The Driver or DataSource class - *Registering the Driver Class*.
- The connection URL for the driver - *Building the Connection URL*

Referencing the JDBC Driver Libraries

Before you use the jdbc driver for Apache Kyuubi, the JDBC application or Java code that you are using to connect to your data must be able to access the driver JAR files.

Using the Driver in Java Code

In the code, specify the artifact *kyuubi-hive-jdbc-shaded* from Maven Central according to the build tool you use.

Maven

```
<dependency>
    <groupId>org.apache.kyuubi</groupId>
    <artifactId>kyuubi-hive-jdbc-shaded</artifactId>
    <version>1.5.2-incubating</version>
</dependency>
```

Sbt

```
libraryDependencies += "org.apache.kyuubi" % "kyuubi-hive-jdbc-shaded" % "1.5.2-
↪incubating"
```

Gradle

```
implementation group: 'org.apache.kyuubi', name: 'kyuubi-hive-jdbc-shaded', version: '1.
↪5.2-incubating'
```

Using the Driver in a JDBC Application

For JDBC Applications, such as BI tools, SQL IDEs, please check the specific guide for detailed information.

Note: Is your favorite tool missing? Report an feature request or help us document it.

Registering the Driver Class

Before connecting to your data, you must register the JDBC Driver class for your application.

- org.apache.kyuubi.jdbc.KyuubiHiveDriver
- org.apache.kyuubi.jdbc.KyuubiDriver (Deprecated)

The following sample code shows how to use the java.sql.DriverManager class to establish a connection for JDBC:

```
private static Connection connectViaDM() throws Exception
{
   Connection connection = null;
   connection = DriverManager.getConnection(CONNECTION_URL);
   return connection;
}
```

Building the Connection URL

Basic Connection URL format

Use the connection URL to supply connection information to the kyuubi server or cluster that you are accessing. The following is the format of the connection URL for the Kyuubi Hive JDBC Driver

```
jdbc:subprotocol://host:port/schema;<clientProperties;><[#|?]sessionProperties>
```

- subprotocol: kyuubi or hive2
- host: DNS or IP address of the kyuubi server
- port: The number of the TCP port that the server uses to listen for client requests
- dbName: Optional database name to set the current database to run the query against, use *default* if absent.
- clientProperties: Optional *semicolon(;)* separated *key*=*value* parameters identified and affect the client behavior locally. e.g., user=foo;password=bar.
- sessionProperties: Optional *semicolon(;)* separated *key*=*value* parameters used to configure the session, operation or background engines. For instance, *kyuubi.engine.share.level=CONNECTION* determines the background engine instance is used only by the current connection. *spark.ui.enabled=false* disables the Spark UI of the engine.

Important:

- The sessionProperties MUST come after a leading number sign(#) or question mark (?).
- Properties are case-sensitive
- Do not duplicate properties in the connection URL

Connection URL over Http

New in version 1.6.0.

```
jdbc:subprotocol://host:port/schema;transportMode=http;httpPath=<http_endpoint>
```

- http_endpoint is the corresponding HTTP endpoint configured by *kyuubi.frontend.thrift.http.path* at the server side.

Connection URL over Service Discovery

```
jdbc:subprotocol://<zookeeper quorum>/;serviceDiscoveryMode=zooKeeper;
↪zooKeeperNamespace=kyuubi
```

- zookeeper quorum is the corresponding zookeeper cluster configured by *kyuubi.ha.zookeeper.quorum* at the server side.
- zooKeeperNamespace is the corresponding namespace configured by *kyuubi.ha.zookeeper.namespace* at the server side.

Authentication

DataTypes

Hive JDBC Driver

Instructions

Kyuubi does not provide its own JDBC Driver so far, as it is fully compatible with Hive JDBC and ODBC drivers that let you connect to popular Business Intelligence (BI) tools to query, analyze and visualize data though Spark SQL engines.

Install Hive JDBC

For programing, the easiest way to get `hive-jdbc` is from the maven central. For example,

- **maven**

```
<dependency>
    <groupId>org.apache.hive</groupId>
    <artifactId>hive-jdbc</artifactId>
    <version>2.3.8</version>
</dependency>
```

- **sbt**

```
libraryDependencies += "org.apache.hive" % "hive-jdbc" % "2.3.8"
```

- **gradle**

```
implementation group: 'org.apache.hive', name: 'hive-jdbc', version: '2.3.8'
```

For BI tools, please refer to Quick Start to check the guide for the BI tool used. If you find there is no specific document for the BI tool that you are using, don't worry, the configuration part for all BI tools are basically the same. Also, we will appreciate if you can help us to improve the document.

JDBC URL

JDBC URLs have the following format:

```
jdbc:hive2://<host>:<port>/<dbName>;<sessionVars>?<kyuubiConfs>#<[spark|hive]Vars>
```

Example

```
jdbc:hive2://localhost:10009/default;hive.server2.proxy.user=proxy_user?kyuubi.engine.
↪share.level=CONNECTION;spark.ui.enabled=false#var_x=y
```

Unsupported Hive Features

- Connect to HiveServer2 using HTTP transport. `transportMode=http`

MySQL Connectors

New in version 1.4.0.

Kyuubi provides an frontend service that enables the connectivity and accessibility from MySQL connectors.

> **Warning:** The document you are visiting now is incomplete, please help kyuubi community to fix it if appropriate for you.

Trino JDBC Driver

Instructions

Kyuubi currently supports the Trino connection protocol, so we can use Trino-JDBC to connect to the kyuubi server and submit SQL to Spark, Trino and other engines for execution.

Start Kyuubi Trino Server

First we should configure the trino protocol and the service port in the `kyuubi.conf`

```
kyuubi.frontend.protocols TRINO
kyuubi.frontend.trino.bind.port 10999 #default port
```

Install Trino JDBC

Download trino-jdbc-363.jar and add it to the classpath of your Java application.

The driver is also available from Maven Central:

```
<dependency>
    <groupId>io.trino</groupId>
    <artifactId>trino-jdbc</artifactId>
    <version>363</version>
</dependency>
```

JDBC URL

When your driver is loaded, registered and configured, you are ready to connect to Trino from your application. The following JDBC URL formats are supported:

```
jdbc:trino://host:port
```

Trino JDBC example

```
String trinoHost = "localhost";
String trinoPort = "10999";
String trinoUser = "default";
String trinoPassword = null;
Connection connection = null;
ResultSet rs = null;

try {
    // Create the connection using the JDBC URL
    connection = DriverManager.getConnection("jdbc:trino://" + trinoHost + ":" +␣
↪trinoPort, trinoUser, trinoPassword);

    // Do whatever you need to do with the connection
    Statement stmt = connection.createStatement();
    rs = stmt.executeQuery("SELECT 1");
```

(continues on next page)

(continued from previous page)

```
    while (rs.next()) {
    // retrieve data from the ResultSet
    }

} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        // Close the connection when you're done with it
        if (rs != null) rs.close();
        if (connection != null) connection.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The configuration of the connection parameters can be found in the official trino documentation at: https://trino.io/docs/current/client/jdbc.html#connection-parameters

3.6.2 Command Line Interface(CLI)s

Kyuubi Beeline

Warning: The document you are visiting now is incomplete, please help kyuubi community to fix it if appropriate for you.

Hive Beeline

Kyuubi supports Apache Hive beeline that works with Kyuubi server. Hive beeline is a SQLLine CLI based on the Hive JDBC Driver.

Prerequisites

- Kyuubi server installed and launched.
- Hive beeline installed

Important: Kyuubi does not support embedded mode which beeline and server run in the same process. It always uses remote mode for connecting beeline with a separate server process over thrift.

Warning: The document you are visiting now is incomplete, please help kyuubi community to fix it if appropriate for you.

Trino command line interface

The Trino CLI provides a terminal-based, interactive shell for running queries. We can use it to connect Kyuubi server now.

Start Kyuubi Trino Server

First we should configure the trino protocol and the service port in the `kyuubi.conf`

```
kyuubi.frontend.protocols TRINO
kyuubi.frontend.trino.bind.port 10999 #default port
```

Install

Download trino-cli-363-executable.jar, rename it to `trino`, make it executable with `chmod +x`, and run it to show the version of the CLI:

```
wget https://repo1.maven.org/maven2/io/trino/trino-jdbc/363/trino-jdbc-363.jar
mv trino-jdbc-363.jar trino
chmod +x trino
./trino --version
```

Running the CLI

The minimal command to start the CLI in interactive mode specifies the URL of the kyuubi server with the Trino protocol:

```
./trino --server http://localhost:10999
```

If successful, you will get a prompt to execute commands. Use the help command to see a list of supported commands. Use the clear command to clear the terminal. To stop and exit the CLI, run exit or quit.:

```
trino> help

Supported commands:
QUIT
EXIT
CLEAR
EXPLAIN [ ( option [, ...] ) ] <query>
    options: FORMAT { TEXT | GRAPHVIZ | JSON }
             TYPE { LOGICAL | DISTRIBUTED | VALIDATE | IO }
DESCRIBE <table>
SHOW COLUMNS FROM <table>
SHOW FUNCTIONS
SHOW CATALOGS [LIKE <pattern>]
SHOW SCHEMAS [FROM <catalog>] [LIKE <pattern>]
SHOW TABLES [FROM <schema>] [LIKE <pattern>]
USE [<catalog>.]<schema>
```

You can now run SQL statements. After processing, the CLI will show results and statistics.

```
trino> select 1;
 _col0
-------
     1
(1 row)

Query 20230216_125233_00806_examine_6hxus, FINISHED, 1 node
Splits: 1 total, 1 done (100.00%)
0.29 [0 rows, 0B] [0 rows/s, 0B/s]

trino>
```

Many other options are available to further configure the CLI in interactive mode to refer https://trino.io/docs/current/client/cli.html#running-the-cli

3.6.3 Business Intelligence Tools and SQL IDEs

Kyuubi provides a standard JDBC/ODBC interface over thrift that allows various existing BI tools, SQL clients/IDEs to connect with.

Note: Is your favorite tool missing? Report an feature request or help us document it.

Apache Superset

Warning: The document you are visiting now is incomplete, please help kyuubi community to fix it if appropriate for you.

Cloudera Hue

What is Hue

Hue is an open source SQL Assistant for Databases & Data Warehouses.

Preparation

Get Kyuubi Started

Get the server Started first before your try Hue with Kyuubi.

```
Welcome to
  __  __                           __
 /\ \/\ \                         /\ \       __
 \ \ \/'/'  __  __  __  __  __   __\ \ \____/\_\
  \ \ , <  /\ \/\ \/\ \/\ \/\ \/\ \\ \ '__`\/\ \
   \ \ \\`\\ \ \_\ \ \ \_\ \ \ \_\ \\ \ \L\ \ \ \
    \ \_\ \_\/`____ \ \____/\ \____/ \ \_,__/\ \_\
```

(continues on next page)

(continued from previous page)

```
    \/_/\/_/`/___/> \/___/  \/___/   \/___/  \/_/
              /\___/
              \/__/
```

Run Hue in Docker

Here we demo running Kyuubi on macOS and Hue on Docker for Mac, there are several known limitations of network, and you can find workarounds from here.

Configuration

1. Copy a configuration template from Hue Docker image.

```
docker run --rm gethue/hue:latest cat /usr/share/hue/desktop/conf/hue.ini > hue.ini
```

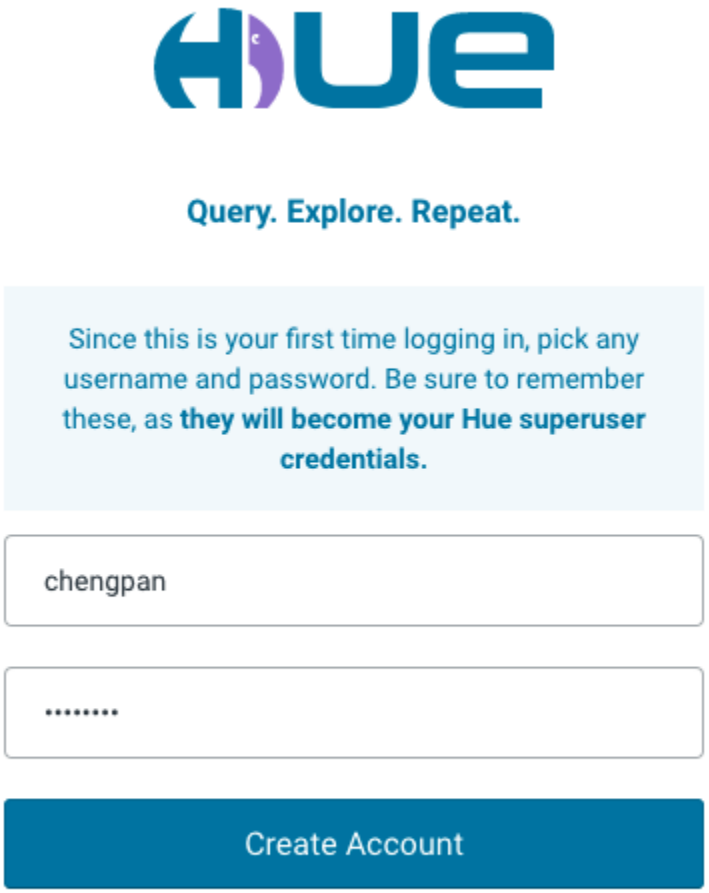
1. Modify the hue.ini

```
[beeswax]
  # Kyuubi 1.1.x support thrift version from 1 to 10
  thrift_version=7
  # change to your username to avoid permissions issue for local test
  auth_username=chengpan

[notebook]
  [[interpreters]]
    [[[sql]]]
      name=SparkSQL
      interface=hiveserver2

[spark]
  # Host of the Spark Thrift Server
  # For macOS users, use docker.for.mac.host.internal to access host network
  sql_server_host=docker.for.mac.host.internal

  # Port of the Spark Thrift Server
  sql_server_port=10009

# other configurations
...
```

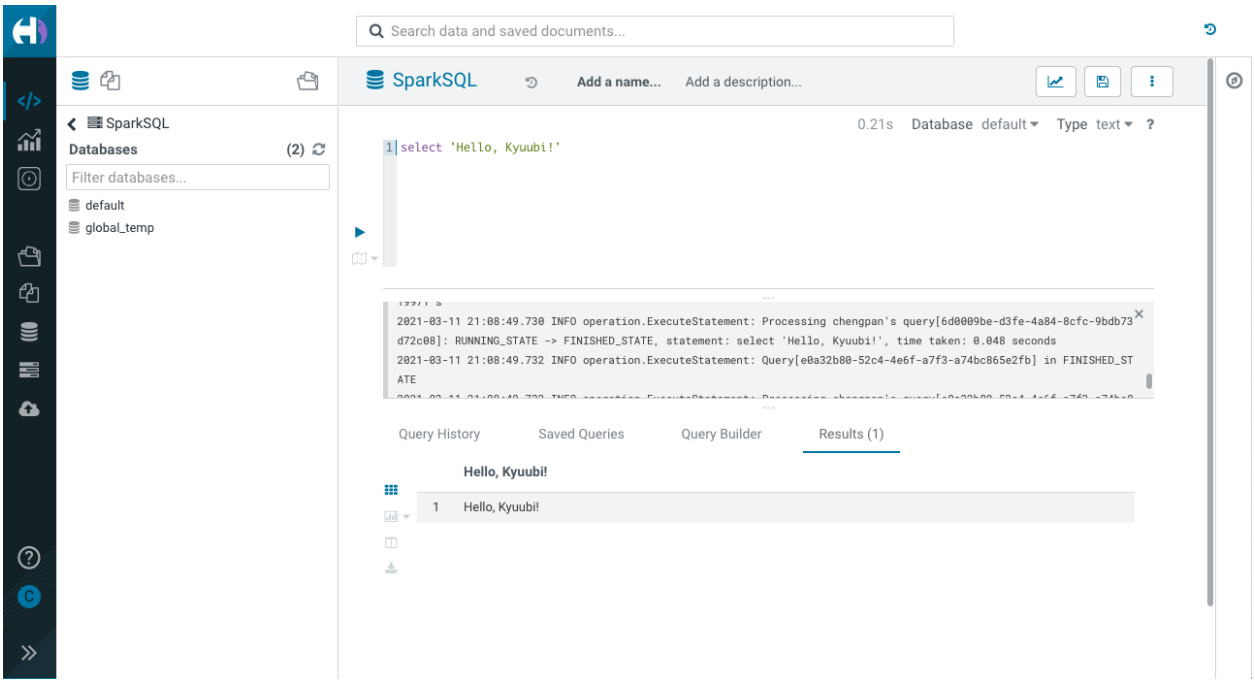
Start Hue in Docker

```
docker run -p 8888:8888 -v $PWD/hue.ini:/usr/share/hue/desktop/conf/hue.ini gethue/
↪hue:latest
```

Go http://localhost:8888/ and follow the guide to create an account.



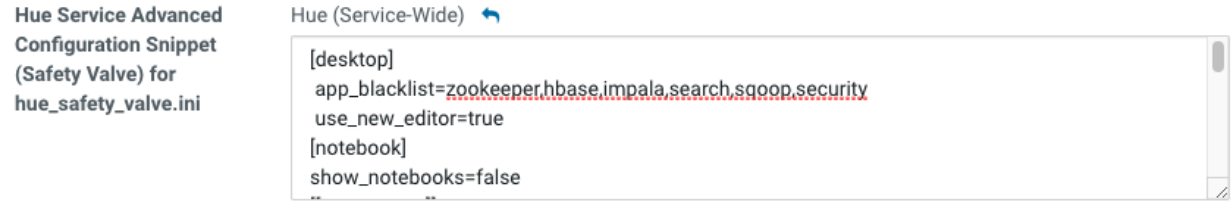
Having fun with Hue and Kyuubi!



For CDH 6.x Users

If you are using CDH 6.x, there is a trick that CDH 6.x blocks Spark in default, you need to modify the configuration to overwrite the `desktop.app_blacklist` to remove this restriction.

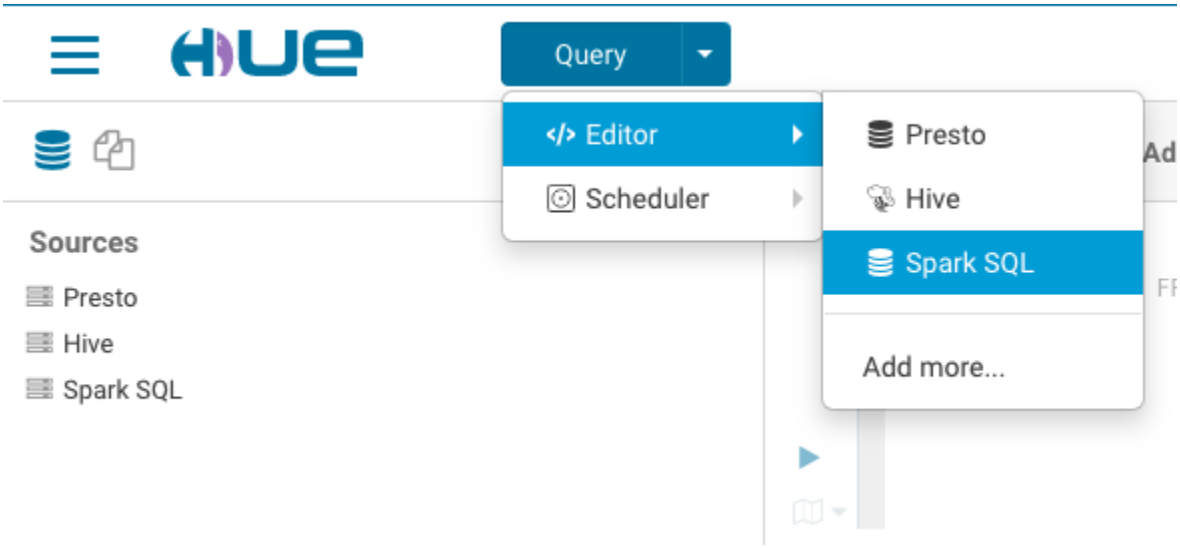
Config Hue in Cloudera Manager.



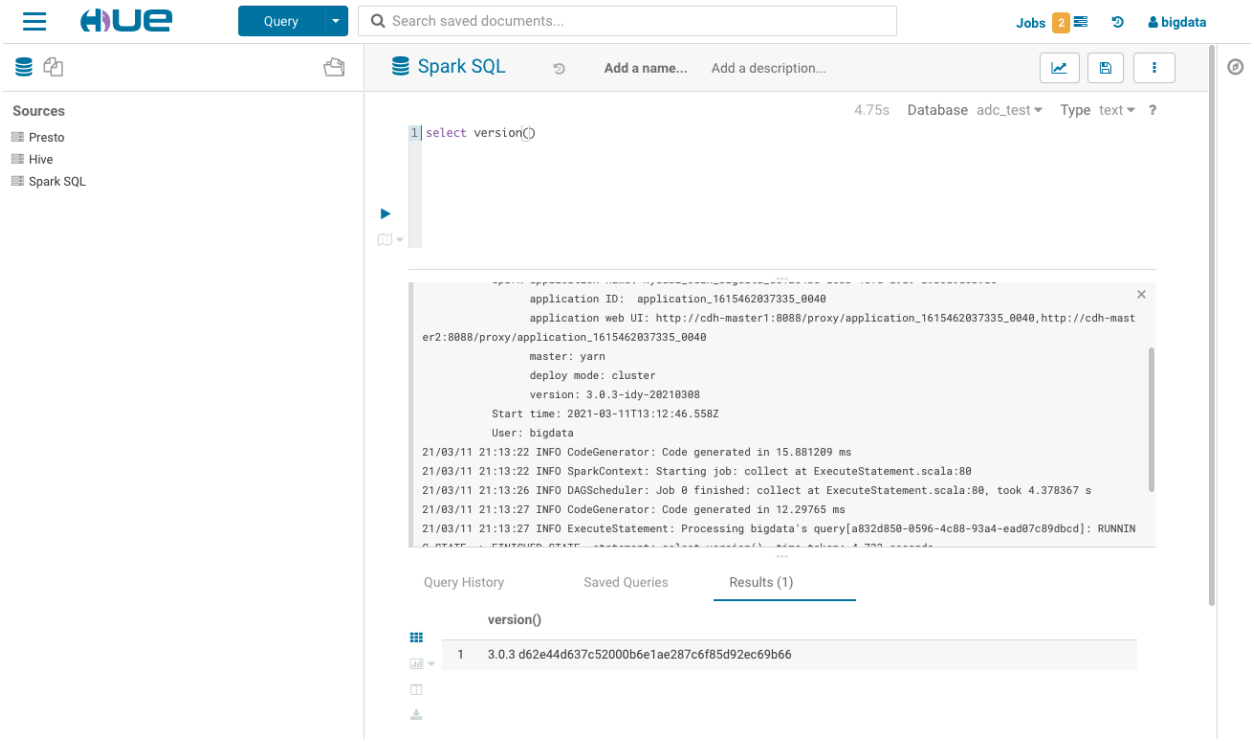
Refer following configuration and tune it to fit your environment.

```
[desktop]
 app_blacklist=zookeeper,hbase,impala,search,sqoop,security
 use_new_editor=true
[[interpreters]]
[[[sparksql]]]
  name=Spark SQL
  interface=hiveserver2
  # other interpreters
  ...
[spark]
sql_server_host=kyuubi-server-host
sql_server_port=10009
```

You need to restart the Hue Service to activate the configuration changes, and then Spark SQL will available in editor list.



Having fun with Hue and Kyuubi!



DataGrip

What is DataGrip

DataGrip is a multi-engine database environment released by JetBrains, supporting MySQL and PostgreSQL, Microsoft SQL Server and Oracle, Sybase, DB2, SQLite, HyperSQL, Apache Derby, and H2.

Preparation

Get DataGrip And Install

Please go to Download DataGrip to get and install an appropriate version for yourself.

Get Kyuubi Started

Get kyuubi server started before you try DataGrip with kyuubi.

For debugging purpose, you can use `tail -f` or `tailf` to track the server log.

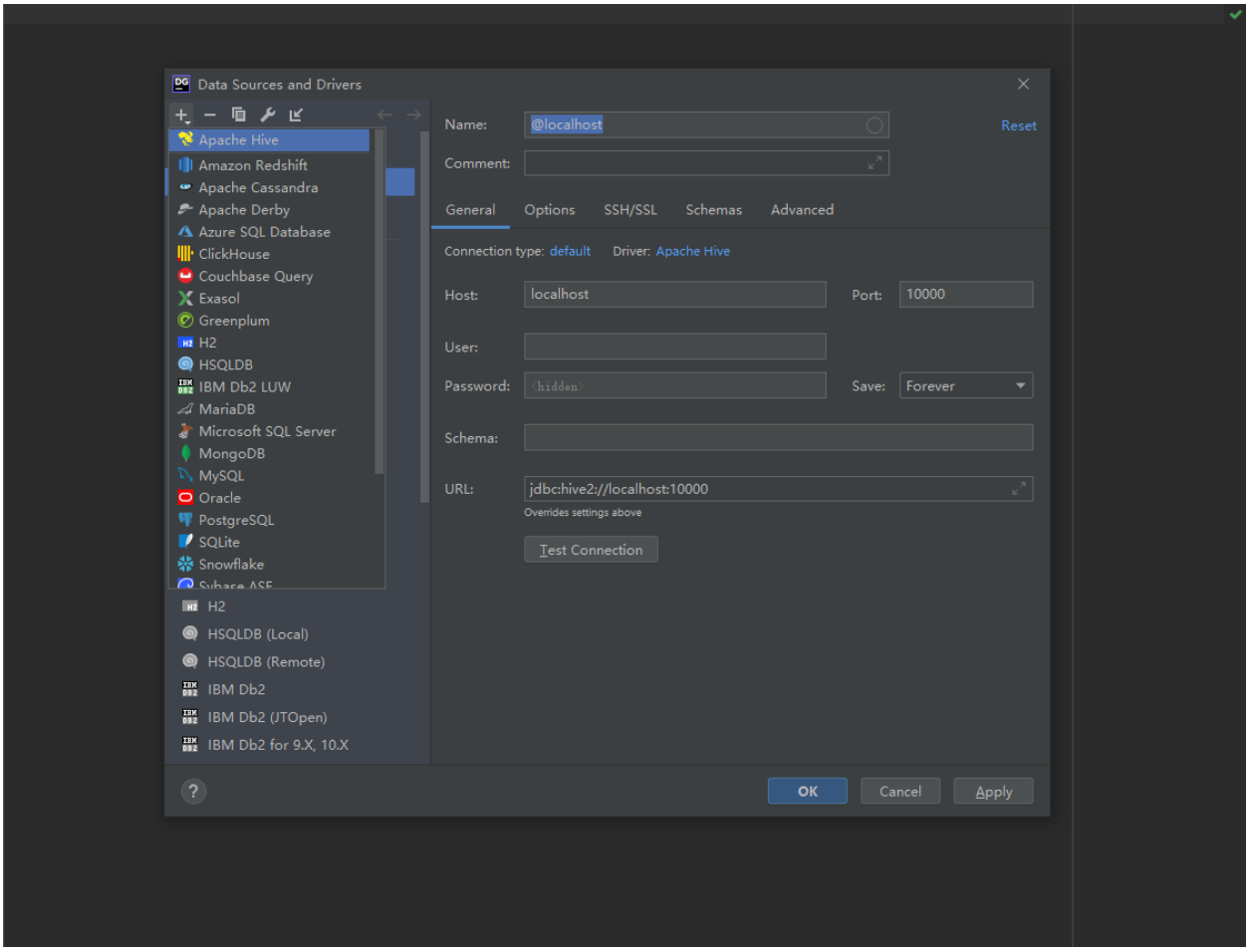
Configurations

Start DataGrip

After you install DataGrip, just launch it.

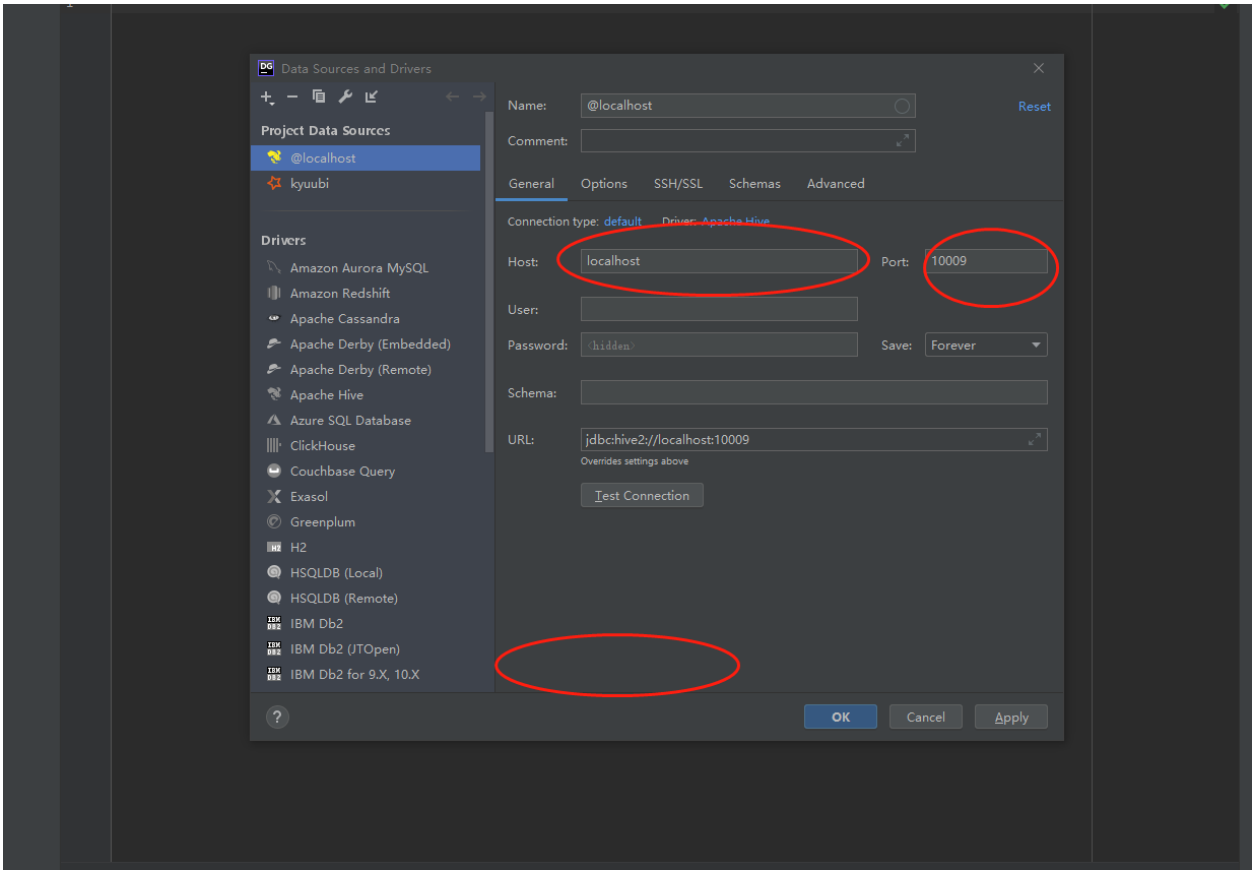
Select Database

Substantially, this step is to choose a JDBC Driver type to use later. We can choose Apache Hive to set up a driver for Kyuubi.



Datasource Driver

You should first download the missing driver files. Just click on the link below, DataGrip will download and install those.

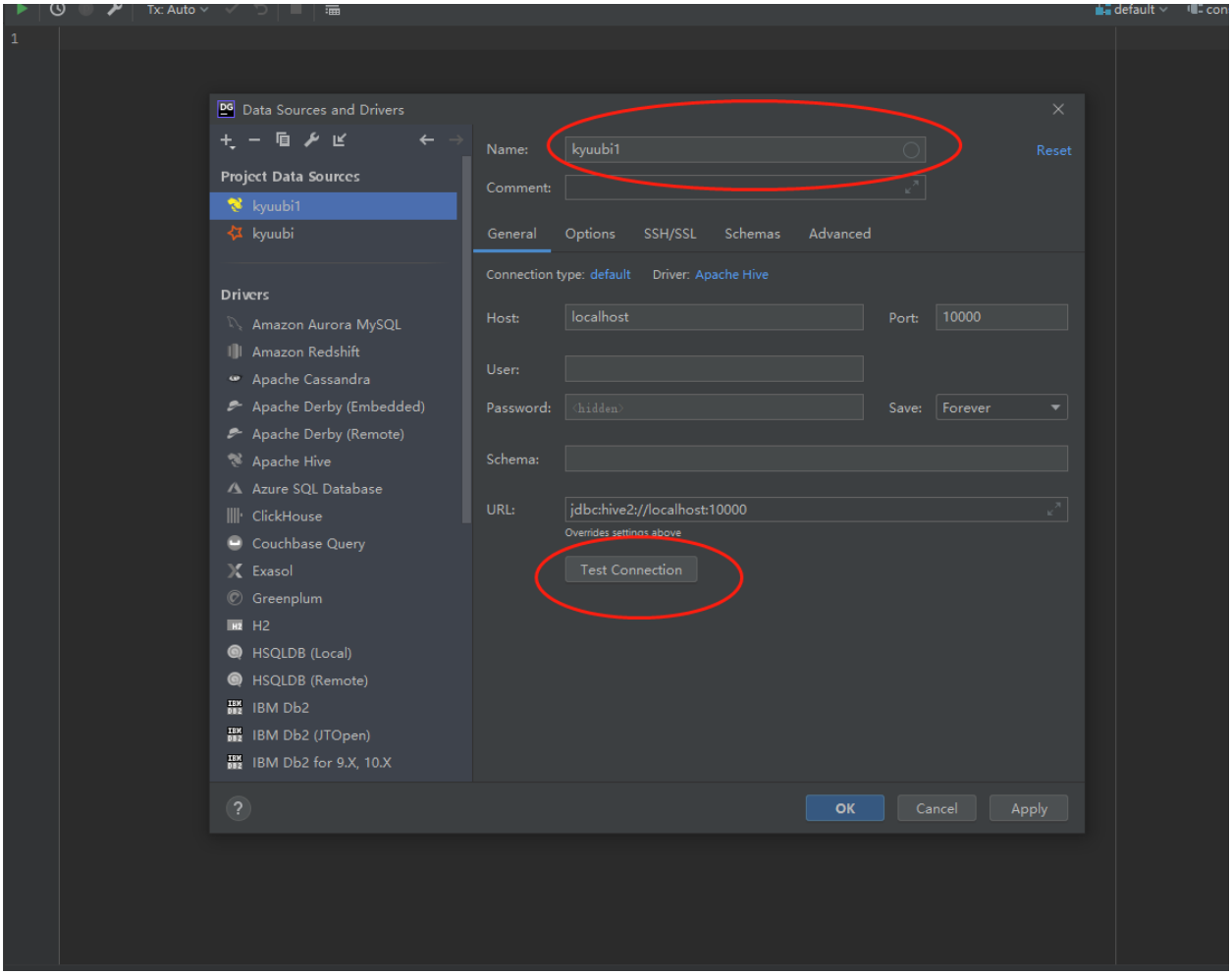


Generic JDBC Connection Settings

After install drivers, you should configure the right host and port which you can find in kyuubi server log. By default, we use `localhost` and `10009` to configure.

Of course, you can fill other configs.

After generic configs, you can use test connection to test.

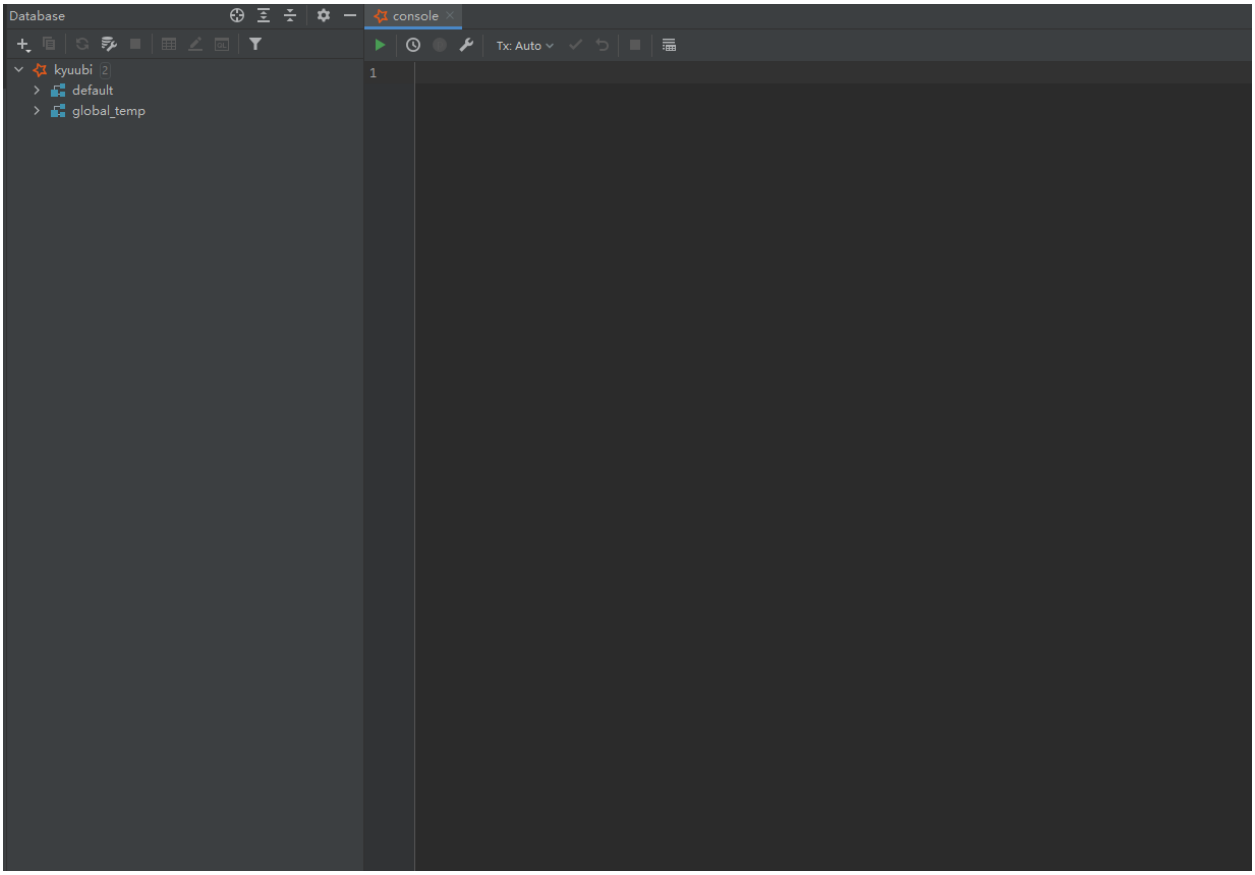


Interacting With Kyuubi Server

Now, you can interact with Kyuubi server.

The left side of the photo is the table, and the right side of the photo is the console.

You can interact through the visual interface or code.



The End

There are many other amazing features in both Kyuubi and DataGrip and here is just the tip of the iceberg. The rest is for you to discover.

DBeaver

What is DBeaver

DBeaver is a free multi-platform database tool for developers, database administrators, analysts, and all people who need to work with databases. Supports all popular databases as well as kyuubi JDBC.

See also:

DBeaver Wiki

Installation

Please go to Download DBeaver page to get and install an appropriate release version for yourself.

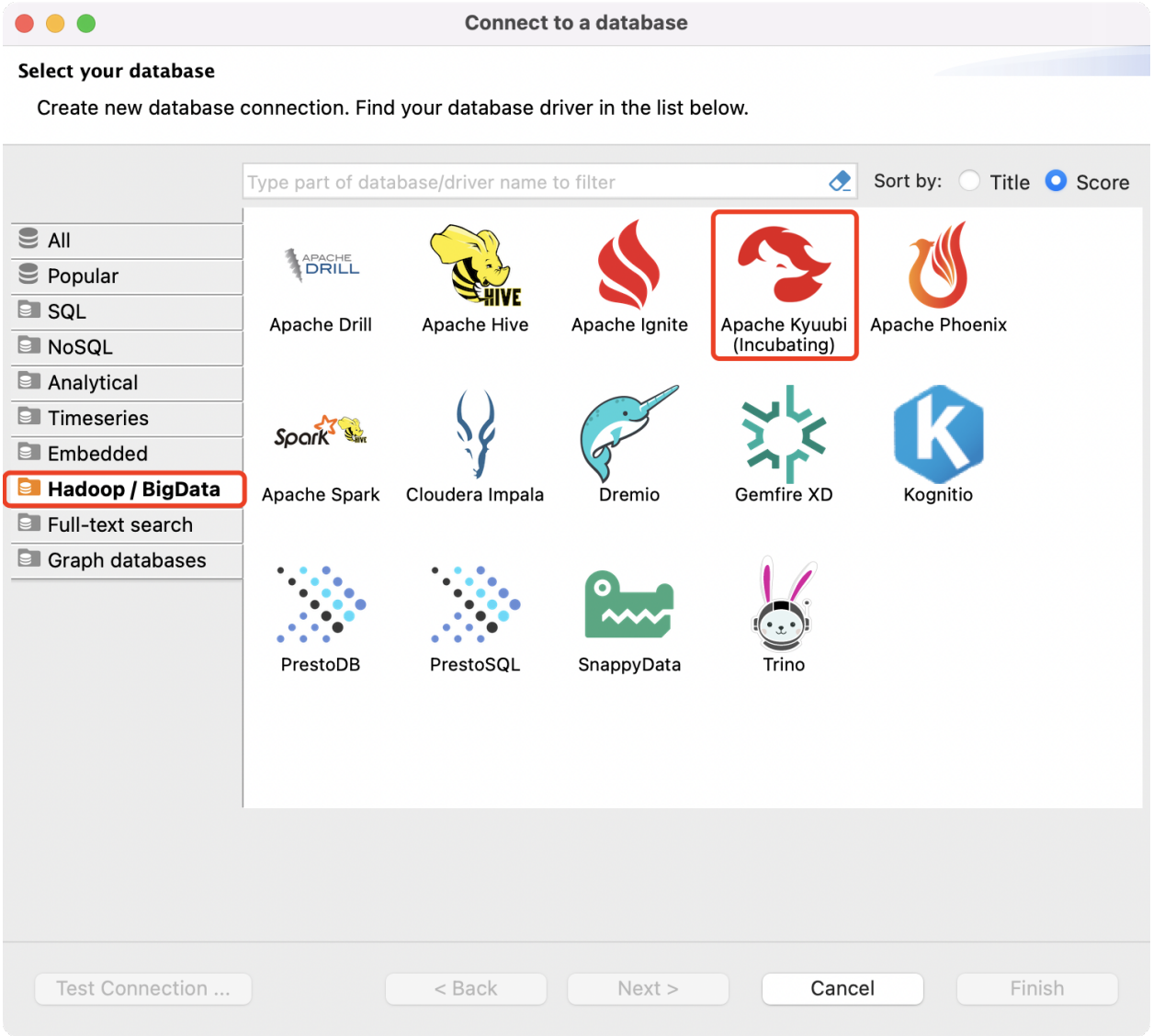
New in version 22.1.0(dbeaver): DBeaver officially supports apache kyuubi JDBC driver since 06 Jun 2022 via PR 16567.

Using DBeaver with Kyuubi

If you have successfully installed dbeaver, just hit the button to launch it.

New Connection

Firstly, we need to create a database connection against a live kyuubi server. You are able to find the kyuubi jdbc driver since dbeaver 22.1.0, as shown in the following figure.



Note: We can also choose Apache Hive or Apache Spark to set up a driver for Kyuubi, because they are compatible with the same client.

Configure Connection

Secondly, we configure the JDBC connection settings to format an underlying kyuubi JDBC connection URL string.

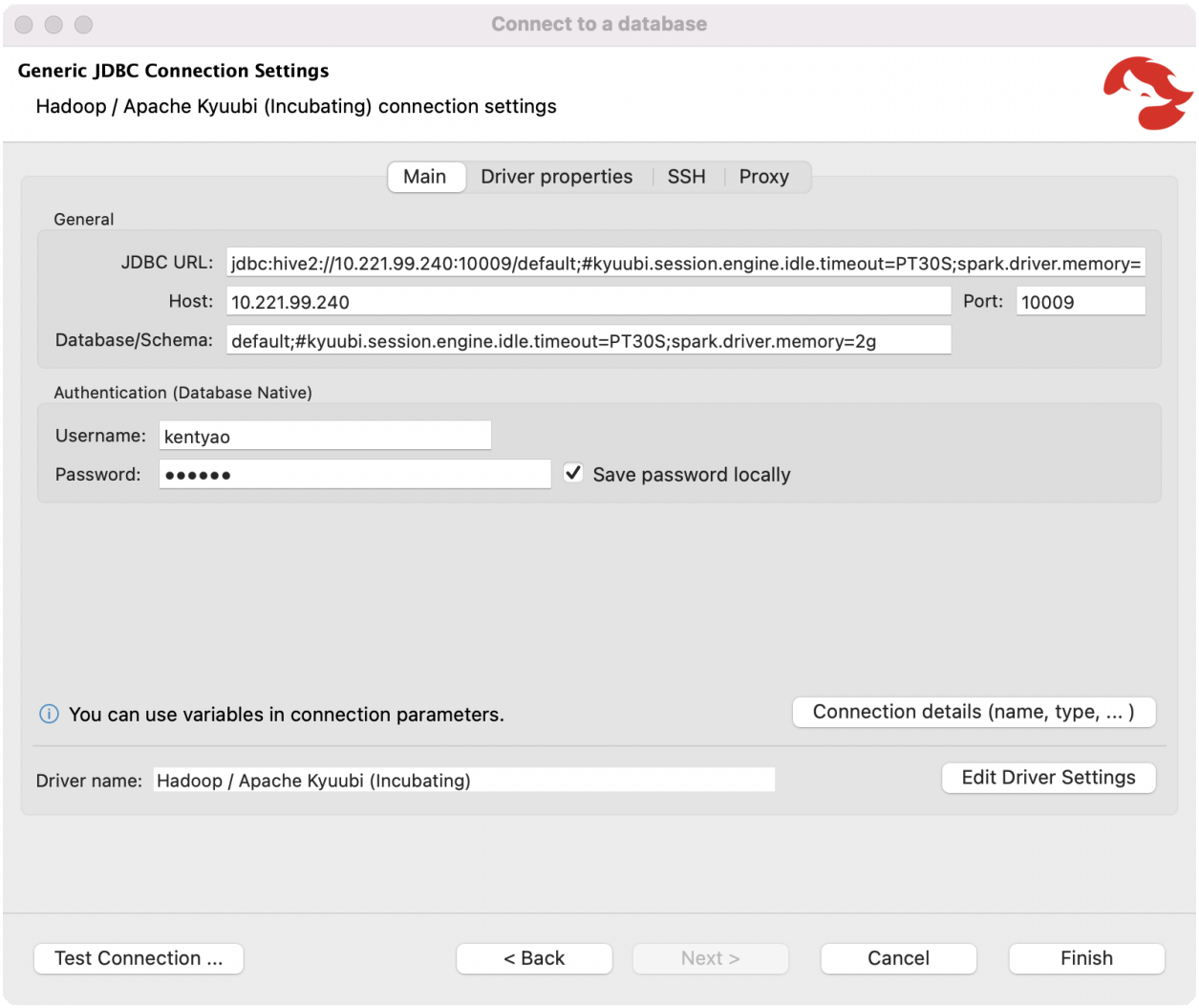
Basic Connection Settings

The basic connection setting contains a minimal set of items you need to talk with kyuubi server,

- Host - hostname or IP address that the kyuubi server bound with, default: *localhost*.
- Port - port that the kyuubi server listening to, default: *10009*.
- Database/Schema - database or schema to use, default: *default*.
- Authentication - identity information, such as user/password, based on the server authentication mechanism.

Session Configurations

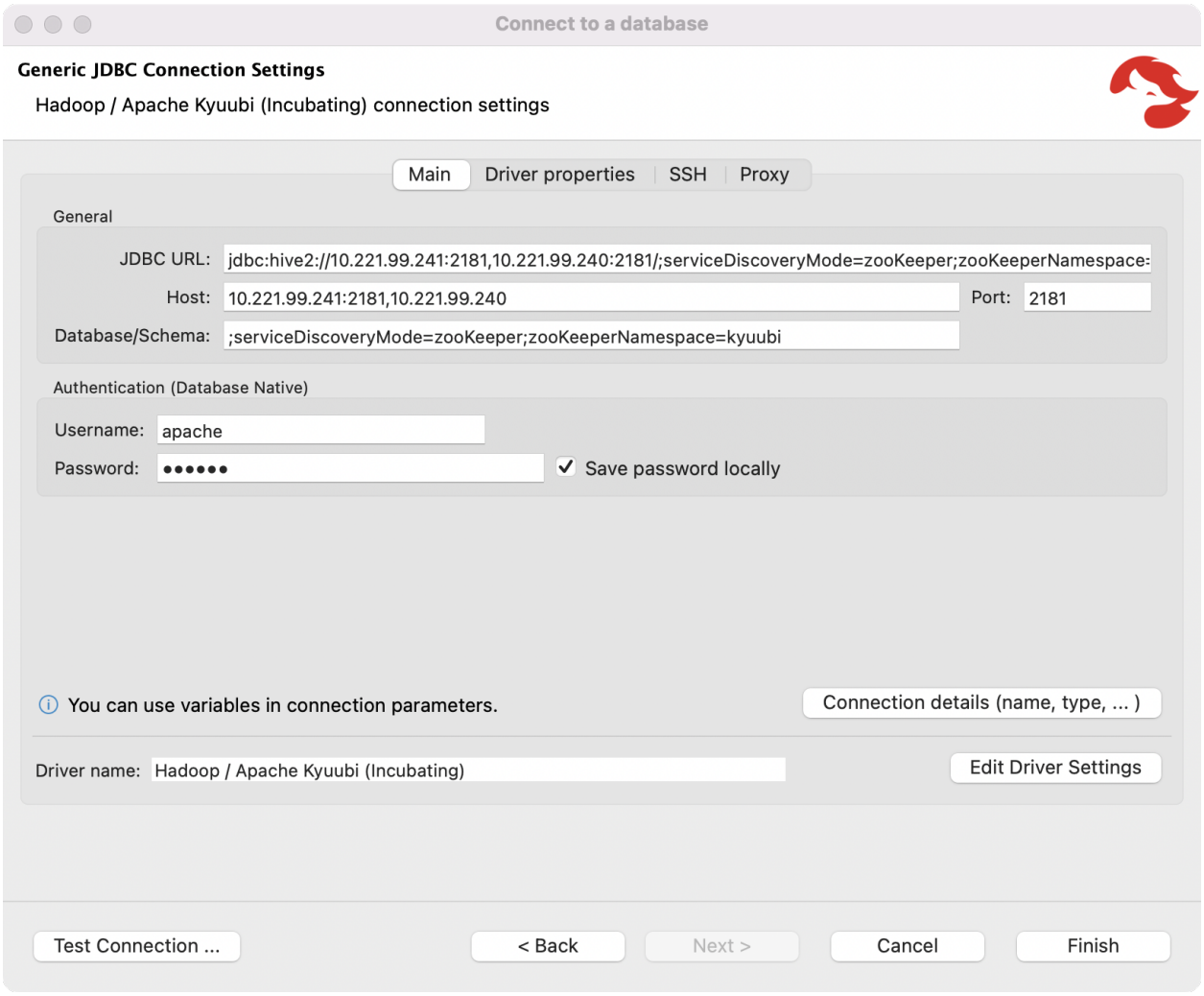
Session configuration list is an optional part of kyuubi JDBC URLs, which are very helpful to override some configurations of the kyuubi server at session scope. The setup page of dbeaver does not contain any text box for such behavior. However, we can append the semicolon-separated configuration pairs to the Database/Schema filed leading with a number sign(#). Though it's a bit weird, but it works.



As an example, shown in the picture above, the engine uses 2 gigabytes memory for the driver process of kyuubi engine and will be terminated after idle for 30 seconds.

Connecting in HA mode

Kyuubi supports HA by service discovery over Apache Zookeeper cluster.



As an example, shown in the above picture, the Host and Port fields can be used to concat the comma separated zookeeper peers, while the *serviceDiscoveryMode* and *zooKeeperNamespace* are appended to the Database/Schema field.

Test Connection

It is not necessary but recommended to click *Test Connection* to verify the connection is set correctly. If something wrong happens at the client side or server side, we can debug ahead with the error message.

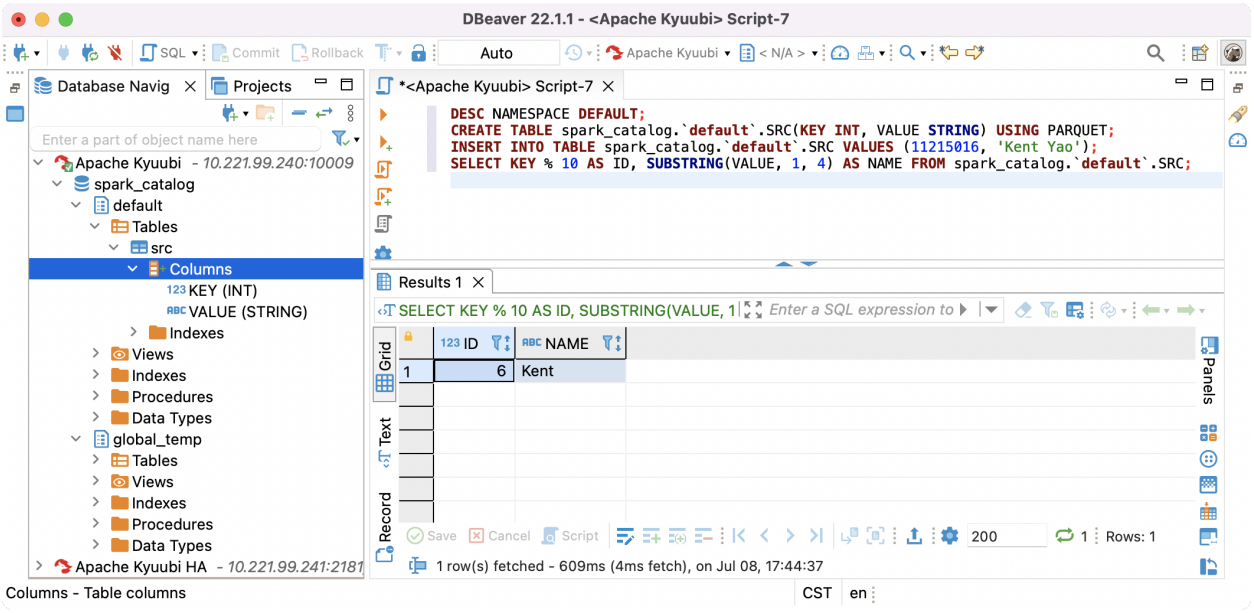
SQL Operations

Now, we can use the SQL editor to write queries to interact with Kyuubi server through the connection.

```
DESC NAMESPACE DEFAULT;
```

```
CREATE TABLE spark_catalog.`default`.SRC(KEY INT, VALUE STRING) USING PARQUET;
INSERT INTO TABLE spark_catalog.`default`.SRC VALUES (11215016, 'Kent Yao');
```

```
SELECT KEY % 10 AS ID, SUBSTRING(VALUE, 1, 4) AS NAME FROM spark_catalog.`default`.SRC;
```




```
DROP TABLE spark_catalog.`default`.SRC;
```

Client Authentication

For kerberized kyuubi clusters, please refer to Kerberos Authentication for more information.

PowerBI

Warning: The document you are visiting now is incomplete, please help kyuubi community to fix it if appropriate for you.

Tableau

Warning: The document you are visiting now is incomplete, please help kyuubi community to fix it if appropriate for you.

3.6.4 ODBC Drivers

3.6.5 Thrift APIs

3.6.6 RESTful APIs and Clients

REST API v1

Note that: now the api version is v1 and the base uri is `/api/v1`.

Session Resource

GET /sessions

Get the list of all live sessions

Response Body

GET /sessions/${sessionHandle}

Get a session event

Response Body

The *KyuubiSessionEvent*.

GET /sessions/${sessionHandle}/info/${infoType}

Get an information detail of a session

Request Parameters

Response Body

GET /sessions/count

Get the current open session count

Response Body

GET /sessions/execPool/statistic

Get statistic info of background executors

Response Body

POST /sessions

Create a session

Request Parameters

Response Body

DELETE /sessions/${sessionHandle}

Close a session.

POST /sessions/${sessionHandle}/operations/statement

Create an operation with EXECUTE_STATEMENT type

Request Body

Response Body

POST /sessions/${sessionHandle}/operations/typeInfo

Create an operation with GET_TYPE_INFO type

Response Body

POST /sessions/${sessionHandle}/operations/catalogs

Create an operation with GET_CATALOGS type

Response Body

POST /sessions/${sessionHandle}/operations/schemas

Create an operation with GET_SCHEMAS type

Request Body

Response Body

POST /sessions/${sessionHandle}/operations/tables

Request Body

Response Body

POST /sessions/${sessionHandle}/operations/tableTypes

Request Parameters

Response Body

POST /sessions/${sessionHandle}/operations/columns

Request Body

Response Body

POST /sessions/${sessionHandle}/operations/functions

Request Body

Response Body

POST /sessions/${sessionHandle}/operations/primaryKeys

Request Parameters

Response Body

POST /sessions/${sessionHandle}/operations/crossReference

Request Body

Response Body

Operation Resource

GET /operations/${operationHandle}/event

Get the current event of the operation by the specified operation handle.

Response Body

The *KyuubiOperationEvent*.

PUT /operations/${operationHandle}

Perform an action to the pending or running operation.

Request Body

GET /operations/${operationHandle}/resultsetmetadata

Get the result set schema of the operation by the specified operation handle.

Response Body

GET /operations/${operationHandle}/log

Get a list of operation log lines of the running operation by the specified operation handle.

Request Parameters

Response Body

GET /operations/${operationHandle}/rowset

Get the operation result as a list of rows by the specified operation handle.

Request Parameters

Response Body

Batch Resource

GET /batches

Returns all the batches.

Request Parameters

Response Body

POST /batches

Create a new batch.

Request Body

- Media type: `application-json`
- JSON structure:

Response Body

The created *Batch* object.

POST /batches (with uploading resource)

Create a new batch with uploading resource file.

Example of using `curl` command to send POST request to `/v1/batches` in `multipart-formdata` media type with uploading resource file from local path.

```
curl --location --request POST 'http://localhost:10099/api/v1/batches' \
--form 'batchRequest="{\"batchType\":\"SPARK\",\"className\":\"org.apache.spark.examples.
↪SparkPi\",\"name\":\"Spark Pi\"}";type=application/json' \
--form 'resourceFile=@"/local_path/example.jar"'
```

Request Body

- Media type: `multipart-formdata`
- Request body structure in multiparts:

Response Body

The created *Batch* object.

GET /batches/{batchId}

Returns the batch information.

Response Body

The *Batch*.

DELETE /batches/${batchId}

Kill the batch if it is still running.

Request Parameters

Response Body

GET /batches/${batchId}/localLog

Gets the local log lines from this batch.

Request Parameters

Response Body

Admin Resource

POST /admin/refresh/hadoop_conf

Refresh the Hadoop configurations of the Kyuubi server.

POST /admin/refresh/user_defaults_conf

Refresh the user defaults configs with key in format in the form of `___{username}___.{config key}` from default property file.

DELETE /admin/engine

Delete the specified engine.

Request Parameters

GET /admin/engine

Get a list of satisfied engines.

Request Parameters

Response Body

The *Engine* List.

REST Objects

Batch

KyuubiSessionEvent

KyuubiOperationEvent

ColumnDesc

Row

Field

Engine

3.6.7 Web UI

3.6.8 Python

PyHive

PyHive is a collection of Python DB-API and SQLAlchemy interfaces for Hive. PyHive can connect with the Kyuubi server serving in thrift protocol as HiveServer2.

Requirements

PyHive works with Python 2.7 / Python 3. Install PyHive via pip for the Hive interface.

```
pip install 'pyhive[hive]'
```

Usage

Use the Kyuubi server's host and thrift protocol port to connect.

For further information about usages and features, e.g. DB-API async fetching, using in SQLAlchemy, please refer to project homepage.

DB-API

```
from pyhive import hive
cursor = hive.connect(host=kyuubi_host,port=10009).cursor()
cursor.execute('SELECT * FROM my_awesome_data LIMIT 10')
print(cursor.fetchone())
print(cursor.fetchall())
```

Use PyHive with Pandas

PyHive provides a handy way to establish a SQLAlchemy compatible connection and works with Pandas dataframe for executing SQL and reading data via `pandas.read_sql`.

```
from pyhive import hive
import pandas as pd

# open connection
conn = hive.Connection(host=kyuubi_host,port=10009)

# query the table to a new dataframe
dataframe = pd.read_sql("SELECT id, name FROM test.example_table", conn)
```

Authentication

If password is provided for connection, make sure the `auth` param set to either `CUSTOM` or `LDAP`.

```
# open connection
conn = hive.Connection(host=kyuubi_host,port=10009,
user='user', password='password', auth='CUSTOM')
```

PySpark

PySpark is an interface for Apache Spark in Python. Kyuubi can be used as JDBC source in PySpark.

Requirements

PySpark works with Python 3.7 and above.

Install PySpark with Spark SQL and optional pandas support on Spark using PyPI as follows:

```
pip install pyspark 'pyspark[sql]' 'pyspark[pandas_on_spark]'
```

For installation using Conda or manually downloading, please refer to PySpark installation.

Preparation

Prepare JDBC driver

Prepare JDBC driver jar file. Supported Hive compatible JDBC Driver as below:

Refer to docs of the driver and prepare the JDBC driver jar file.

Prepare JDBC Hive Dialect extension

Hive Dialect support is required by Spark for wrapping SQL correctly and sending it to the JDBC driver. Kyuubi provides a JDBC dialect extension with auto-registered Hive Dialect support for Spark. Follow the instructions in Hive Dialect Support to prepare the plugin jar file `kyuubi-extension-spark-jdbc-dialect_-*.jar`.

Including jars of JDBC driver and Hive Dialect extension

Choose one of the following ways to include jar files in Spark.

- Put the jar file of JDBC driver and Hive Dialect to `$SPARK_HOME/jars` directory to make it visible for the classpath of PySpark. And adding `spark.sql.extensions = org.apache.spark.sql.dialect.KyuubiSparkJdbcDialectExtension` to `$SPARK_HOME/conf/spark_defaults.conf`.
- With spark's start shell, include the JDBC driver when submitting the application with `--packages`, and the Hive Dialect plugins with `--jars`

```
$SPARK_HOME/bin/pyspark --py-files PY_FILES \
--packages org.apache.hive:hive-jdbc:x.y.z \
--jars /path/kyuubi-extension-spark-jdbc-dialect_-*.jar
```

- Setting jars and config with SparkSession builder

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
        .config("spark.jars", "/path/hive-jdbc-x.y.z.jar,/path/kyuubi-extension-spark-
↪jdbc-dialect_-*.jar") \
        .config("spark.sql.extensions", "org.apache.spark.sql.dialect.
↪KyuubiSparkJdbcDialectExtension") \
        .getOrCreate()
```

Usage

For further information about PySpark JDBC usage and options, please refer to Spark's JDBC To Other Databases.

Using as JDBC Datasource programmingly

```
# Loading data from Kyuubi via HiveDriver as JDBC datasource
jdbcDF = spark.read \
  .format("jdbc") \
  .options(driver="org.apache.hive.jdbc.HiveDriver",
           url="jdbc:hive2://kyuubi_server_ip:port",
           user="user",
           password="password",
           query="select * from testdb.src_table"
           ) \
  .load()
```

Using as JDBC Datasource table with SQL

From Spark 3.2.0, `CREATE DATASOURCE TABLE` is supported to create jdbc source with SQL.

```
# create JDBC Datasource table with DDL
spark.sql("""CREATE TABLE kyuubi_table USING JDBC
OPTIONS (
    driver='org.apache.hive.jdbc.HiveDriver',
    url='jdbc:hive2://kyuubi_server_ip:port',
    user='user',
    password='password',
    dbtable='testdb.some_table'
)""")

# read data to dataframe
jdbcDF = spark.sql("SELECT * FROM kyuubi_table")

# write data from dataframe in overwrite mode
df.writeTo("kyuubi_table").overwrite

# write data from query
spark.sql("INSERT INTO kyuubi_table SELECT * FROM some_table")
```

Use PySpark with Pandas

From PySpark 3.2.0, PySpark supports pandas API on Spark which allows you to scale your pandas workload out.

Pandas-on-Spark DataFrame and Spark DataFrame are virtually interchangeable. More instructions in From/to pandas and PySpark DataFrames.

```
import pyspark.pandas as ps

psdf = ps.range(10)
sdf = psdf.to_spark().filter("id > 5")
sdf.show()
```

3.6.9 Client Commons

Client Configuration Guide

Logging

Advanced Features

Using Different Kyuubi Engines

Sharing and Isolation for Kyuubi Engines

Setting Time to Live for Kyuubi Engines

Enabling Kyuubi Engine Pool

Running Scala Snippets

Plan Only Execution Mode

Plan only execution mode currently only supports spark and flink engine.

Configure the kyuubi.operation.plan.only.mode parameter, the value can be 'parse', 'analyze', 'optimize', 'optimize_with_stats', 'physical', 'execution', or 'none', when it is 'none', indicate to the statement will be fully executed, otherwise only way without executing the query. Different engines currently support different modes, the spark engine supports all modes, and the flink engine supports 'parse', 'physical', and 'execution', other engines do not support plan-only mode currently, and supports application-level and session-level parameter settings, which are used in the following ways.

Application-level parameter setting

You can add parameters to the URL when establishing a JDBC connection, the parameter is kyuubi.operation.plan.only.mode=parse/analyze/optimize. JDBC URLs have the following format:

```
jdbc:hive2://<host>:<port>/<dbName>;<sessionVars>?kyuubi.operation.plan.only.mode=parse/
↪analyze/optimize/optimize_with_stats/physical/execution/none;<kyuubiConfs>#
↪<[spark|hive]Vars>
```

Refer to hive_jdbc doc for details of others parameters

Example:

Using beeline tool to connect to the local service, the Shell command is:

```
beeline -u 'jdbc:hive2://0.0.0.0:10009/default?kyuubi.operation.plan.only.mode=parse' -n
↪{user_name}
```

Running the following SQL:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id
```

The results are as follows:

```
# SQL:
0: jdbc:hive2://0.0.0.0:10009/default> SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id;

#Result:
+-----------------------------------------------------+
|                        plan                         |
+-----------------------------------------------------+
| 'Project [*]
+- 'Join LeftOuter, ('t1.id = 't2.id)
   :- 'UnresolvedRelation [t1], [], false
   +- 'UnresolvedRelation [t2], [], false
 |
+-----------------------------------------------------+
1 row selected (3.008 seconds)
0: jdbc:hive2://0.0.0.0:10009/default>
```

Session-level parameter setting

You can also set the kyuubi.operation.plan.only.mode parameter by executing the set command after the connection has been established

```
beeline -u 'jdbc:hive2://0.0.0.0:10009/default' -n {user_name}
```

Running the following SQL:

```
set kyuubi.operation.plan.only.mode=parse;
SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id
```

The results are as follows:

```
#set command:
0: jdbc:hive2://0.0.0.0:10009/default> set kyuubi.operation.plan.only.mode=parse;

#set command result:
+-----------------------------------+--------+
|               key                 | value  |
+-----------------------------------+--------+
| kyuubi.operation.plan.only.mode   | parse  |
+-----------------------------------+--------+
1 row selected (0.568 seconds)

#execute SQL:
0: jdbc:hive2://0.0.0.0:10009/default> SELECT * FROM t1 LEFT JOIN t2 ON t1.id = t2.id;

# SQL result:
+-----------------------------------------------------+
|                        plan                         |
+-----------------------------------------------------+
| 'Project [*]
+- 'Join LeftOuter, ('t1.id = 't2.id)
   :- 'UnresolvedRelation [t1], [], false
```

(continues on next page)

(continued from previous page)

```
   +- 'UnresolvedRelation [t2], [], false
 |
+----------------------------------------------------+
1 row selected (0.404 seconds)
0: jdbc:hive2://0.0.0.0:10009/default>
```

3.7 Extensions

Besides the base use case, Kyuubi also has some extension points for extending use cases. By defining plugin of your own or applying third-party ones, kyuubi allows to run your plugin's functionality at the specific point.

The extension points can be divided into server side extensions and engine side extensions.

Server side extensions are applied by kyuubi administrators to extend the ability of kyuubi servers.

Engine side extensions are applied to kyuubi engines, some of them can be managed by administrators, some of them can be applied by end-users dynamically at runtime.

3.7.1 Server Side Extensions

Server side extensions for injecting custom functionality to some of the modules at the kyuubi server. They are applied by kyuubi administrators to extend the ability of kyuubi servers.

Inject Session Conf with Custom Config Advisor

New in version 1.5.0.

Session Conf Advisor

Kyuubi supports inject session configs with custom config advisor. It is usually used to append or overwrite session configs dynamically, so administrators of Kyuubi can have an ability to control the user specified configs.

The steps of injecting session configs

1. create a custom class which implements the `org.apache.kyuubi.plugin.SessionConfAdvisor`.
2. compile and put the jar into `$KYUUBI_HOME/jars`
3. adding configuration at `kyuubi-defaults.conf`:

   ```
   kyuubi.session.conf.advisor=${classname}
   ```

The `org.apache.kyuubi.plugin.SessionConfAdvisor` has a zero-arg constructor, holds one method with user and session conf and returns a new conf map.

```
public interface SessionConfAdvisor {
  default Map<String, String> getConfOverlay(String user, Map<String, String>␣
↪sessionConf) {
    return Collections.EMPTY_MAP;
```

(continues on next page)

(continued from previous page)

```
  }
}
```

Note: The returned conf map will overwrite the original session conf.

Example

We have a custom class `CustomSessionConfAdvisor`:

```
@Override
public class CustomSessionConfAdvisor {
  Map<String, String> getConfOverlay(String user, Map<String, String> sessionConf) {
    if ("uly".equals(user)) {
      return Collections.singletonMap("spark.driver.memory", "1G");
    } else {
      return Collections.EMPTY_MAP;
    }
  }
}
```

If a user *uly* creates a connection with:

```
jdbc:hive2://localhost:10009/;hive.server2.proxy.user=uly;#spark.driver.memory=2G
```

The final Spark application will allocate `1G` rather than `2G` for the driver jvm.

Configure Kyuubi to use Custom EventHandler

Kyuubi provide event processing mechanism, it can help us to record some events. Beside the builtin `JsonLoggingEventHandler`, Kyuubi supports custom event handler. It is usually used to write Kyuubi events to some external systems. For example, Kafka, ElasticSearch, etc. The `org.apache.kyuubi.events.handler.CustomEventHandlerProvider` has a zero-arg constructor, it can help us to create a custom EventHandler.

```
package org.apache.kyuubi.events.handler

import org.apache.kyuubi.config.KyuubiConf
import org.apache.kyuubi.events.KyuubiEvent

/**
 * Custom EventHandler provider. We can implement it to provide a custom EventHandler.
 * The implementation will be loaded by ``Service Provider Interface``.
 */
trait CustomEventHandlerProvider {

  /**
   * The create method is called to create a custom event handler
   * when this implementation is loaded.
   *
   * @param kyuubiConf The conf can be used to read some configs.
```

(continues on next page)

(continued from previous page)

```
   * @return A custom handler to handle KyuubiEvent.
   */
   def create(kyuubiConf: KyuubiConf): EventHandler[KyuubiEvent]
}
```

Build A Custom EventHandler

To create custom EventHandlerProvider class derived from the above interface, we need to:

- Referencing the library

```
<dependency>
   <groupId>org.apache.kyuubi</groupId>
   <artifactId>kyuubi-event_2.12</artifactId>
   <version>1.7.0-incubating</version>
   <scope>provided</scope>
</dependency>
```

- Implement `org.apache.kyuubi.events.handler.CustomEventHandlerProvider`
- Adding a file named `org.apache.kyuubi.events.handler.CustomEventHandlerProvider` in the src/main/resources/META-INF/services folder of project, its content is the custom class name.

Enable Custom EventHandler

To enable the custom EventHandler, we need to

- **Put the jar package to `$KYUUBI_HOME/jars` directory to make it visible for** the classpath of the kyuubi server.
- Configure the following properties to `$KYUUBI_HOME/conf/kyuubi-defaults.conf` on each node where kyuubi server is installed. If you need use other event handler, it can be appended after the `CUSTOM`.
- Restart all the kyuubi server instances.

Manage Applications against Extra Cluster Managers

New in version 1.6.0.

Caution: unstable

Inside Kyuubi, the Kyuubi server uses the `ApplicationManager` module to manage all applications launched by itself, including different kinds of Kyuubi engines and self-contained applications.

The `ApplicationManager` leverages methods provided by application operation implementations derived from `org.apache.kyuubi.engine.ApplicationOperation` to monitor the status of those applications and kill abnormal applications in case they get orphaned and may introduce more methods in the future.

An `ApplicationOperation` implementation is usually built upon clients or APIs provided by cluster managers, such as Hadoop YARN, Kubernetes, etc.

For now, Kyuubi has already supported several built-in application operations:

- `JpsApplicationOperation`: an operation that can manage apps with a local process, e.g. a local mode spark application
- `YarnApplicationOperation`: an operation that can manage apps with a Hadoop Yarn cluster, e.g. a spark on yarn application
- `KubernetesApplicationOperation`: an operation that can manage apps with a k8s cluster, e.g. a spark on k8s application

Besides those built-in ones, Kyuubi also supports loading custom `ApplicationOperation` through the Java *ServiceLoader* (SPI) for extra cluster managers.

The rest of this article will show you the specifications and steps to build and enable a custom operation.

```
trait ApplicationOperation {

  /**
   * Step for initializing the instance.
   */
  def initialize(conf: KyuubiConf): Unit

  /**
   * Step to clean up the instance
   */
  def stop(): Unit

  /**
   * Called before other method to do a quick skip
   *
   * @param clusterManager the underlying cluster manager or just local␣
↪instance
   */
  def isSupported(clusterManager: Option[String]): Boolean

  /**
   * Kill the app/engine by the unique application tag
   *
   * @param tag the unique application tag for engine instance.
   *            For example,
   *            if the Hadoop Yarn is used, for spark applications,
   *            the tag will be preset via spark.yarn.tags
   * @return a message contains response describing how the kill process.
   *
   * @note For implementations, please suppress exceptions and always return␣
↪KillResponse
   */
  def killApplicationByTag(tag: String): KillResponse

  /**
   * Get the engine/application status by the unique application tag
   *
   * @param tag the unique application tag for engine instance.
   * @return [[ApplicationInfo]]
   */
  def getApplicationInfoByTag(tag: String): ApplicationInfo
}
```

```
/**
  * (killed or not, hint message)
  */
type KillResponse = (Boolean, String)
```

An `ApplicationInfo` is used to represented the application information, including application id, name, state, url address and error message.

```
object ApplicationState extends Enumeration {
  type ApplicationState = Value
  val PENDING, RUNNING, FINISHED, KILLED, FAILED, ZOMBIE, NOT_FOUND, UNKNOWN =␣
↪Value
}

case class ApplicationInfo(
    id: String,
    name: String,
    state: ApplicationState,
    url: Option[String] = None,
    error: Option[String] = None)
```

For application state mapping, you can reference the implementation of yarn:

```
def toApplicationState(state: YarnApplicationState): ApplicationState = state␣
↪match {
  case YarnApplicationState.NEW => ApplicationState.PENDING
  case YarnApplicationState.NEW_SAVING => ApplicationState.PENDING
  case YarnApplicationState.SUBMITTED => ApplicationState.PENDING
  case YarnApplicationState.ACCEPTED => ApplicationState.PENDING
  case YarnApplicationState.RUNNING => ApplicationState.RUNNING
  case YarnApplicationState.FINISHED => ApplicationState.FINISHED
  case YarnApplicationState.FAILED => ApplicationState.FAILED
  case YarnApplicationState.KILLED => ApplicationState.KILLED
  case _ =>
    warn(s"The yarn driver state: $state is not supported, " +
      "mark the application state as UNKNOWN.")
    ApplicationState.UNKNOWN
}
```

Build A Custom Application Operation

- reference kyuubi-server

```
<dependency>
   <groupId>org.apache.kyuubi</groupId>
   <artifactId>kyuubi-server_2.12</artifactId>
   <version>1.5.2-incubating</version>
   <scope>provided</scope>
</dependency>
```

- create a custom class which implements the `org.apache.kyuubi.engine.ApplicationOperation`.

- create a directory META-INF.services and a file with `org.apache.kyuubi.engine.ApplicationOperation`:

  ```
  META-INF.services/org.apache.kyuubi.engine.ApplicationOperation
  ```

  then add your fully-qualified name of custom application operation into the file.

Enable Custom Application Operation

Note: Kyuubi uses Java SPI to load the custom Application Operation

---

- compile and put the jar into `$KYUUBI_HOME/jars`

3.7.2 Engine Side Extensions

Engine side extensions are applied to kyuubi engines, some of them can be managed by administrators, some of them can be applied by end-users dynamically at runtime.

Extensions for Spark

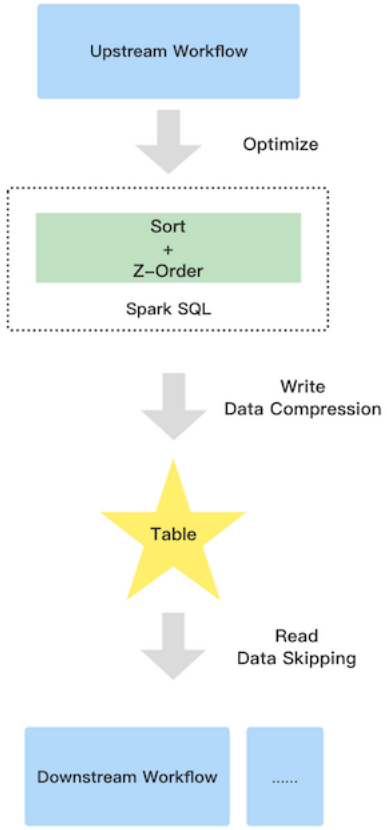
Z-Ordering Support

To improve query speed, Kyuubi supports Z-Ordering to optimize the layout of data stored in all kind of storage with various data format.

Please check our benchmark report here.

Introduction

The following picture shows the workflow of z-order.



It contains three parties:

- Upstream

Due to the extra sort, the upstream job will run a little slower than before

- Table

  Z-order has the good data clustering, so the compression ratio can be improved

- Downstream

  Improve the downstream read performance benefit from data skipping. Since the parquet and orc file support collect data statistic automatically when you write data e.g. minimum and maximum values, the good data clustering let the pushed down filter more efficient

Supported table format

Supported column data type

How to use

This feature is inside Kyuubi extension, so you should apply the extension to Spark by following steps.

- add extension jar: `copy $KYUUBI_HOME/extension/kyuubi-extension-spark-3-1* $SPARK_HOME/jars/`
- add config into `spark-defaults.conf`: `spark.sql.extensions=org.apache.kyuubi.sql.KyuubiSparkSQLExtension`

Due to the extension, z-order only works with Spark-3.1 and higher version.

Optimize history data

If you want to optimize the history data of a table, the `OPTIMIZE ...` syntax is good to go. Due to Spark SQL doesn't support read and overwrite same datasource table, the syntax can only support to optimize Hive table.

Syntax

```
OPTIMIZE table_name [WHERE predicate] ZORDER BY col_name1 [, ...]
```

Note that, the `predicate` only supports partition spec.

Examples

```
OPTIMIZE t1 ZORDER BY c3;

OPTIMIZE t1 ZORDER BY c1,c2;

OPTIMIZE t1 WHERE day = '2021-12-01' ZORDER BY c1,c2;
```

Optimize incremental data

Kyuubi supports optimize a table automatically for incremental data. e.g., time partitioned table. The only things you need to do is adding Kyuubi properties into the target table properties:

```
ALTER TABLE t1 SET TBLPROPERTIES('kyuubi.zorder.enabled'='true','kyuubi.zorder.cols'='c1,
↪c2');
```

- the key `kyuubi.zorder.enabled` decide if the table allows Kyuubi to optimize by z-order.
- the key `kyuubi.zorder.cols` decide which columns are used to optimize by z-order.

Kyuubi will detect the properties and optimize SQL using z-order during SQL compilation, so you can enjoy z-order with all writing table command like:

```
INSERT INTO TABLE t1 PARTITION() ...;

INSERT OVERWRITE TABLE t1 PARTITION() ...;

CREATE TABLE t1 AS SELECT ...;
```

Auxiliary Optimization Rules

Kyuubi provides SQL extension out of box. Due to the version compatibility with Apache Spark, currently we support Apache Spark branch-3.1 and later. And don't worry, Kyuubi will support the new Apache Spark version in the future. Thanks to the adaptive query execution framework (AQE), Kyuubi can do these optimizations.

Features

- merging small files automatically

  Small files is a long time issue with Apache Spark. Kyuubi can merge small files by adding an extra shuffle. Currently, Kyuubi supports handle small files with datasource table and hive table, and also Kyuubi support optimize dynamic partition insertion. For example, a common write query `INSERT INTO TABLE $table1 SELECT * FROM $table2`, Kyuubi will introduce an extra shuffle before write and then the small files will go away.

- insert shuffle node before Join to make AQE OptimizeSkewedJoin work

  In current implementation, Apache Spark can only optimize skewed join by the standard join which means a join must have two sort and shuffle node. However, in complex scenario this assuming will be broken easily. Kyuubi can guarantee the join is standard by adding an extra shuffle node before join. So that, OptimizeSkewedJoin can work better.

- stage level config isolation in AQE

  As we know, `spark.sql.adaptive.advisoryPartitionSizeInBytes` is a key config in Apache Spark AQE. It controls how big data size per-task should handle during shuffle, so we always use a 64MB or a smaller value to make parallelism enough. However, in general, we expect a file is big enough like 256MB or 512MB. Kyuubi can make the config isolation to solve the conflict so that we can make staging partition data size small and last partition data size big.

Usage

1. Check the matrix that if you are using the supported Spark version, and find the corresponding Kyuubi Spark SQL Extension jar
2. Get the Kyuubi Spark SQL Extension jar
   1. Each Kyuubi binary release tarball only contains one default version of Kyuubi Spark SQL Extension jar, if you are looking for such version, you can find it under `$KYUUBI_HOME/extension`
   2. All supported versions of Kyuubi Spark SQL Extension jar will be deployed to Maven Central
   3. If you like, you can compile Kyuubi Spark SQL Extension jar by yourself, please activate the corresponding Maven's profile on you compile command, i.e. you can get Kyuubi Spark SQL Extension jar for Spark 3.1 under `extensions/spark/kyuubi-extension-spark-3-1/target` when compile with `-Pspark-3.1`
3. Put the Kyuubi Spark SQL extension jar `kyuubi-extension-spark-*.jar` into `$SPARK_HOME/jars`

4. Enable `KyuubiSparkSQLExtension`, i.e. add a config into `$SPARK_HOME/conf/spark-defaults.conf`, `spark.sql.extensions=org.apache.kyuubi.sql.KyuubiSparkSQLExtension`

Now, you can enjoy the Kyuubi SQL Extension.

Additional Configurations

Kyuubi provides some configs to make these feature easy to use.

Auxiliary SQL Functions

Kyuubi provides several auxiliary SQL functions as supplement to Spark's Built-in Functions

Connectors for Spark SQL Query Engine

The Kyuubi Spark SQL Query Engine uses Spark DataSource APIs(V1/V2) to access data from different data sources.

By default, it provides accessibility to hive warehouses with various file formats supported, such as parquet, orc, json, etc.

Alsoit can easily integrate with other third-party libraries, such as Hudi, Iceberg, Delta Lake, Kudu, Flink Table Store, HBaseCassandra, etc.

We also provide sample data sources like TDC-DS, TPC-H for testing and benchmarking purpose.

Delta Lake

Delta lake is an open-source project that enables building a Lakehouse Architecture on top of existing storage systems such as S3, ADLS, GCS, and HDFS.

Tip: This article assumes that you have mastered the basic knowledge and operation of Delta Lake. For the knowledge about delta lake not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards delta lake which is more convenient, easy to understand, and easy to expand than directly using spark to manipulate delta lake.

Delta Lake Integration

To enable the integration of kyuubi spark sql engine and delta lake through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the delta lake *dependencies*
- Setting the spark extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with delta lake supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with kyuubi distributions
2. a copy of spark distribution
3. delta-core & delta-storage, which can be found in the Maven Central

In order to make the delta packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the delta packages into `$SPARK_HOME/jars` directly
2. Set `spark.jars=/path/to/delta-core,/path/to/delta-storage`

Warning: Please mind the compatibility of different Delta Lake and Spark versions, which can be confirmed on the page of delta release notes.

Configurations

To activate functionality of delta lake, we can set the following configurations:

```
spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
```

Delta Lake Operations

As for end-users, who only use a pure SQL interface, there aren't much differences between using a delta table and a regular hive table. Unless you are going to use some advanced features, but they are still SQL, just more syntax added.

Taking `CREATE A TABLE` as a example,

```
CREATE TABLE IF NOT EXISTS kyuubi_delta (
  id INT,
  name STRING,
  org STRING,
  url STRING,
  start TIMESTAMP
) USING DELTA;
```

Delta Lake with Microsoft Azure Blob Storage

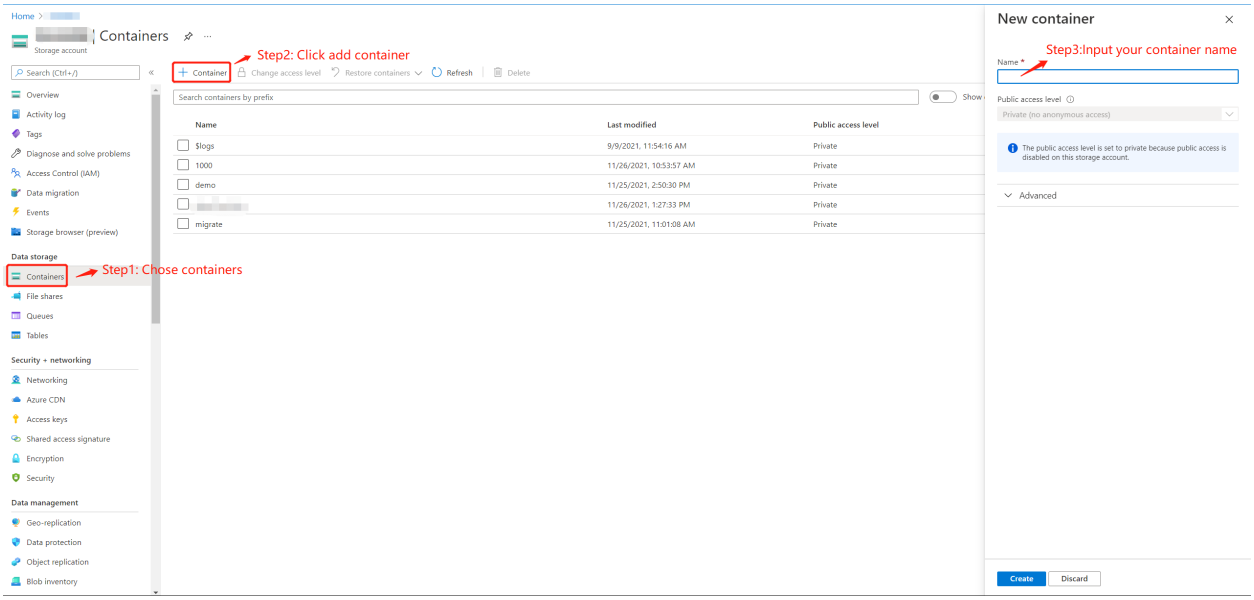
Registration And Configuration

Register An Account And Log In

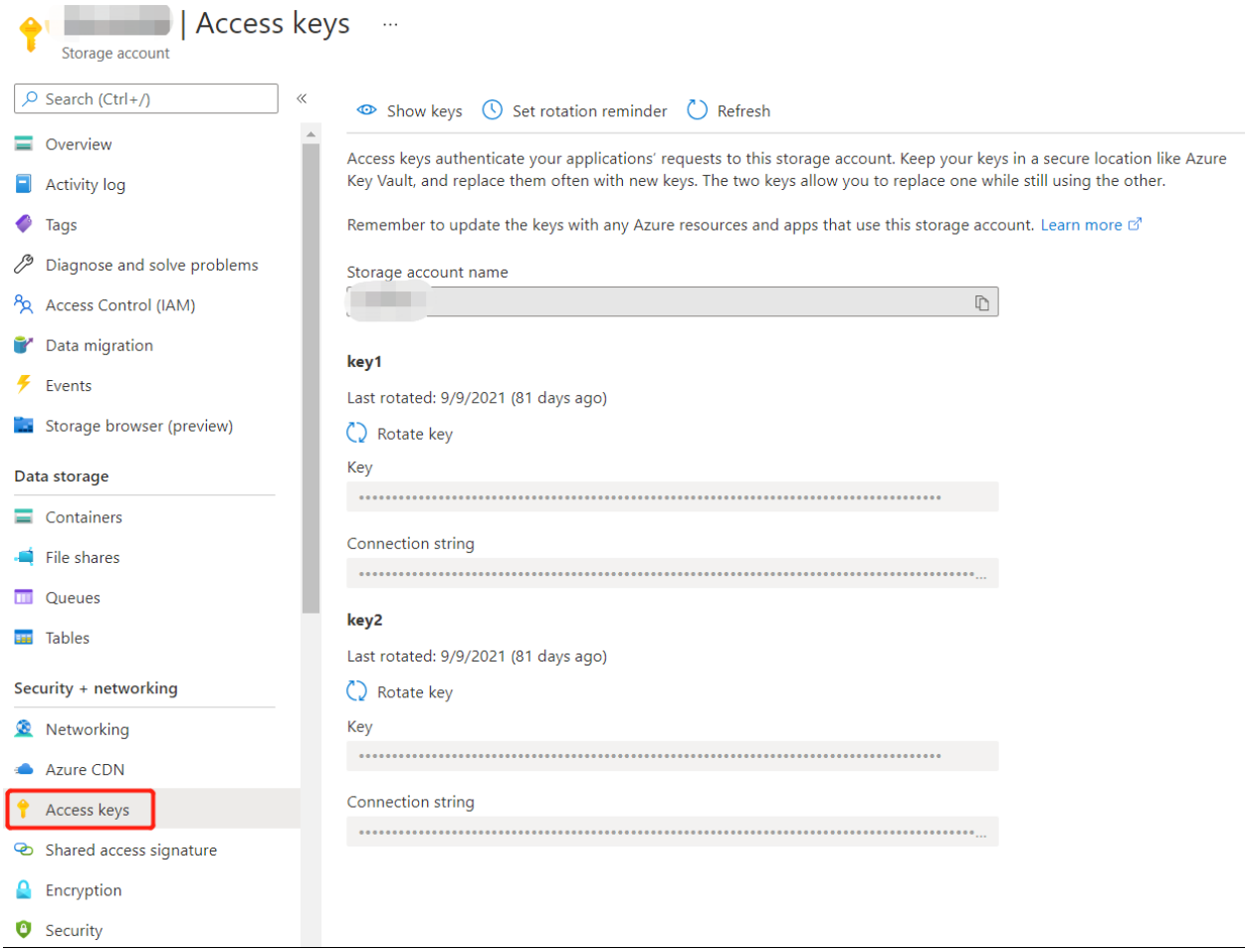
Regarding the Microsoft Azure account, please contact your organization or register an account as an individual. For details, please refer to the Microsoft Azure official website.

Create Storage Container

After logging in with your Microsoft Azure account, please follow the steps below to create a data storage container:



Get Access Key



Deploy Spark

Download Spark Package

Download spark package that matches your environment from spark official website. And then unpackage: `shell tar -xzvf spark-3.2.0-bin-hadoop3.2.tgz `

Config Spark

Enter the `$SPARK_HOME/conf` directory and execute:

```
cp spark-defaults.conf.template spark-defaults.conf
```

Add following configuration to spark-defaults.conf, please refer to your own local configuration for specific personalized configuration:

```
spark.master                        spark://<YOUR_HOST>:7077
spark.sql.extensions                io.delta.sql.DeltaSparkSessionExtension
spark.sql.catalog.spark_catalog     org.apache.spark.sql.delta.catalog.DeltaCatalog
```

Create a new file named `core-site.xml` under `$SPARK_HOME/conf` directory, and add following configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>
<property>
    <name>fs.AbstractFileSystem.wasb.Impl</name>
    <value>org.apache.hadoop.fs.azure.Wasb</value>
 </property>
 <property>
  <name>fs.azure.account.key.YOUR_AZURE_ACCOUNT.blob.core.windows.net</name>
  <value>YOUR_AZURE_ACCOUNT_ACCESS_KEY</value>
 </property>
 <property>
    <name>fs.azure.block.blob.with.compaction.dir</name>
    <value>/hbase/WALs,/tmp/myblobfiles</value>
 </property>
 <property>
    <name>fs.azure</name>
    <value>org.apache.hadoop.fs.azure.NativeAzureFileSystem</value>
 </property>
<property>
    <name>fs.azure.enable.append.support</name>
    <value>true</value>
 </property>
</configuration>
```

Copy Dependencies To Spark

Copy jar packages required by delta lake and microsoft azure to ./spark/jars directory:

```
wget https://repo1.maven.org/maven2/io/delta/delta-core_2.12/1.0.0/delta-core_2.12-1.0.0.
↪jar -O ./spark/jars/delta-core_2.12-1.0.0.jar
wget https://repo1.maven.org/maven2/com/microsoft/azure/azure-storage/8.6.6/azure-
↪storage-8.6.6.jar -O ./spark/jars/azure-storage-8.6.6.jar
wget https://repo1.maven.org/maven2/com/azure/azure-storage-blob/12.14.2/azure-storage-
↪blob-12.14.2.jar -O ./spark/jars/azure-storage-blob-12.14.2.jar
wget https://repo1.maven.org/maven2/org/apache/hadoop/hadoop-azure/3.1.1/hadoop-azure-3.
↪1.1.jar -O ./spark/jars/hadoop-azure-3.1.1.jar
```

Start Spark Standalone cluster

```
./spark/sbin/start-master.sh -h <YOUR_HOST> -p 7077 --webui-port 9090
./spark/sbin/start-worker.sh spark://<YOUR_HOST>:7077
```

Test The connectivity Of Spark And Delta Lake

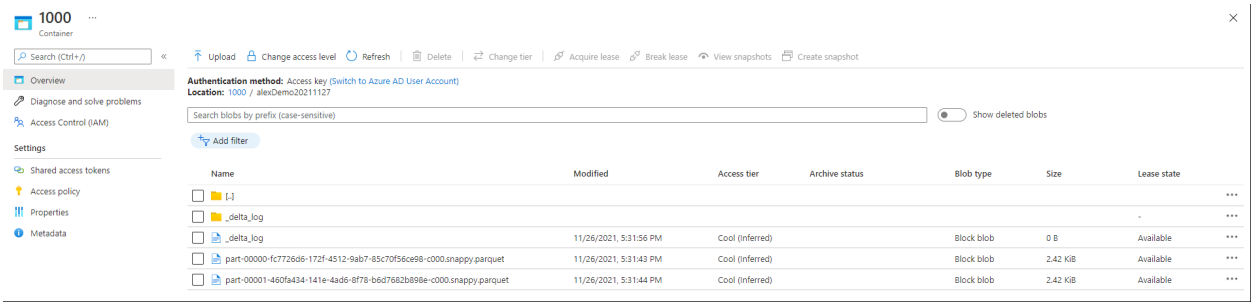
Start spark shell:

```
./bin/spark-shell
```

Generate a piece of random data and push them to delta lake:

```
scala> val data = spark.range(1000, 2000)
scala> data.write.format("delta").mode("overwrite").save("wasbs://<YOUR_CONTAINER_NAME>@
↪<YOUR_AZURE_ACCOUNT>.blob.core.windows.net/<YOUR_TABLE_NAME>")
```

After this, you can check your data on azure web UI. For example, my container name is 1000 and table name is alexDemo20211127:



You can also check data by reading back the data from delta lake:

```
scala> val df=spark.read.format("delta").load("wasbs://<YOUR_CONTAINER_NAME>@<YOUR_AZURE_
↪ACCOUNT>.blob.core.windows.net/<YOUR_TABLE_NAME>")
scala> df.show()
```

If there is no problem with the above, it proves that spark has been built with delta lake.

Deploy Kyuubi

Install Kyuubi

1.Download the latest version of kyuubi from kyuubi download page.

2.Unpackage

    tar -xzvf apache-kyuubi-1.7.0-bin.tgz

Config Kyuubi

Enter the ./kyuubi/conf directory

```
cp kyuubi-defaults.conf.template kyuubi-defaults.conf
vim kyuubi-defaults.conf
```

Add the following content:

Start Kyuubi

```
bin/kyuubi start
```

Check kyuubi log, in order to check kyuubi start status and find the jdbc connection url:

```
2021-11-26 17:49:50.235 INFO service.ThriftFrontendService: Starting and exposing JDBC␣
↪connection at: jdbc:hive2://HOST:10009/
2021-11-26 17:49:50.265 INFO client.ServiceDiscovery: Created a /kyuubi/
↪serviceUri=host:10009;version=1.3.1-incubating;sequence=0000000037 on ZooKeeper for␣
↪KyuubiServer uri: host:10009
2021-11-26 17:49:50.267 INFO server.KyuubiServer: Service[KyuubiServer] is started.
```

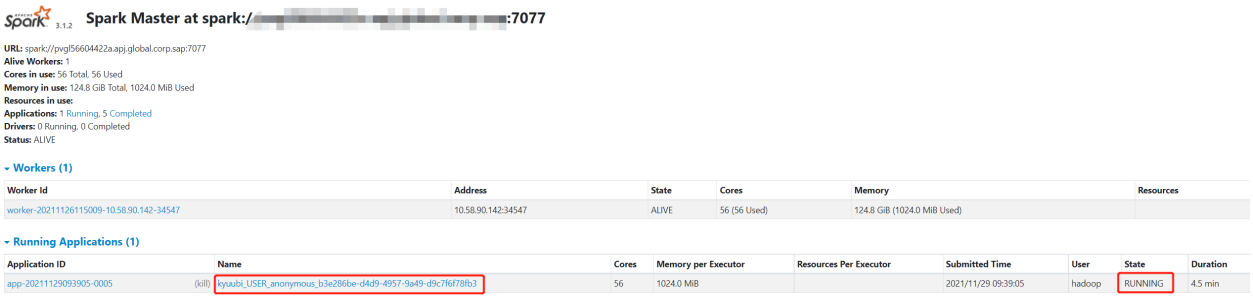
You can get the jdbc connection url by the log above.

Test The Connectivity Of Kyuubi And Delta Lake

Use `$KYUUBI_HOME/bin/beeline` tool,

```
./bin//beeline -u 'jdbc:hive2://<YOUR_HOST>:10009/'
```

At the same time, you can also check whether the engine is running on the spark UI:



When the engine started, it will expose a thrift endpoint and register itself into ZooKeeper, Kyuubi server can get the connection info from ZooKeeper and establish the connection to the engine. So, you can check the registration details in zookeeper path '/kyuubi_USER/anonymous'.

Dealing Delta Lake Data By Using Kyuubi Examples

Operate delta-lake data through SQL:

Create Table

Insert Data

Append Mode

```
INSERT INTO delta.`wasbs://1000@azure_account.blob.core.windows.net/alexDemo20211129` (
    date,
    eventId,
    eventType,
    data)
VALUES
    (now(),'001','test','Hello World!'),
    (now(),'002','test','Hello World!'),
    (now(),'003','test','Hello World!');
```

Result:

```
+-------------+----------+------------+---------------+
|    date     | eventId  | eventType  |     data      |
+-------------+----------+------------+---------------+
| 2021-11-29  | 001      | test       | Hello World!  |
| 2021-11-29  | 003      | test       | Hello World!  |
| 2021-11-29  | 002      | test       | Hello World!  |
+-------------+----------+------------+---------------+
```

Overwrite Mode

```
INSERT OVERWRITE TABLE delta.`wasbs://1000@azure_account.blob.core.windows.net/
↪alexDemo20211129`(
    date,
    eventId,
    eventType,
    data)
VALUES
(now(),'001','test','hello kyuubi'),
(now(),'002','test','hello kyuubi');
```

Result:

Delete Table Data

Result:

```
+-------------+----------+------------+----------------+
|    date     | eventId  | eventType  |     data       |
+-------------+----------+------------+----------------+
| 2021-11-29  | 001      | test       | hello kyuubi   |
+-------------+----------+------------+----------------+
```

Update table data

```
UPDATE
    delta.`wasbs://1000@azure_account.blob.core.windows.net/alexDemo20211129`
SET data = 'This is a test for update data.'
WHERE eventId = 001;
```

Result:

```
+-------------+----------+------------+-----------------------------------+
|    date     | eventId  | eventType  |               data                |
+-------------+----------+------------+-----------------------------------+
| 2021-11-29  | 001      | test       | This is a test for update data.   |
+-------------+----------+------------+-----------------------------------+
```

Select table data

```
SELECT *
FROM
    delta.`wasbs://1000@azure_account.blob.core.windows.net/alexDemo20211129`;
```

Result:

```
+-------------+----------+------------+-----------------------------------+
|    date     | eventId  | eventType  |               data                |
+-------------+----------+------------+-----------------------------------+
| 2021-11-29  | 001      | test       | This is a test for update data.   |
+-------------+----------+------------+-----------------------------------+
```

Hudi

Apache Hudi (pronounced "hoodie") is the next generation streaming data lake platform. Apache Hudi brings core warehouse and database functionality directly to a data lake.

Tip: This article assumes that you have mastered the basic knowledge and operation of Hudi. For the knowledge about Hudi not mentioned in this article, you can obtain it from its Official Documentation.

By using Kyuubi, we can run SQL queries towards Hudi which is more convenient, easy to understand, and easy to expand than directly using Spark to manipulate Hudi.

Hudi Integration

To enable the integration of kyuubi spark sql engine and Hudi through Catalog APIs, you need to:

- Referencing the Hudi *dependencies*
- Setting the Spark extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with Hudi supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of spark distribution
3. hudi-spark<spark.version>-bundle_<scala.version>-<hudi.version>.jar (example: hudi-spark3.2-bundle_2.12-0.11.1.jar), which can be found in the Maven Central

In order to make the Hudi packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the Hudi packages into `$SPARK_HOME/jars` directly
2. Set `spark.jars=/path/to/hudi-spark-bundle`

Configurations

To activate functionality of Hudi, we can set the following configurations:

Hudi Operations

Taking `Create Table` as a example,

```
CREATE TABLE hudi_cow_nonpcf_tbl (
  uuid INT,
  name STRING,
  price DOUBLE
) USING HUDI;
```

Taking `Query Data` as a example,

```
SELECT * FROM hudi_cow_nonpcf_tbl WHERE id < 20;
```

Taking `Insert Data` as a example,

```
INSERT INTO hudi_cow_nonpcf_tbl SELECT 1, 'a1', 20;
```

Taking `Update Data` as a example,

```
UPDATE hudi_cow_nonpcf_tbl SET name = 'foo', price = price * 2 WHERE id = 1;
```

Taking `Delete Data` as a example,

```
DELETE FROM hudi_cow_nonpcf_tbl WHERE uuid = 1;
```

Iceberg

Apache Iceberg is an open table format for huge analytic datasets. Iceberg adds tables to compute engines including Spark, Trino, PrestoDB, Flink, Hive and Impala using a high-performance table format that works just like a SQL table.

Tip: This article assumes that you have mastered the basic knowledge and operation of Iceberg. For the knowledge about Iceberg not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Iceberg which is more convenient, easy to understand, and easy to expand than directly using spark to manipulate Iceberg.

Iceberg Integration

To enable the integration of kyuubi spark sql engine and Iceberg through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the Iceberg *dependencies*
- Setting the spark extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with Iceberg supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of spark distribution
3. iceberg-spark-runtime-<spark.version>_<scala.version>-<iceberg.version>.jar (example: iceberg-spark-runtime-3.2_2.12-0.14.0.jar), which can be found in the Maven Central

In order to make the Iceberg packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the Iceberg packages into `$SPARK_HOME/jars` directly
2. Set `spark.jars=/path/to/iceberg-spark-runtime`

Warning: Please mind the compatibility of different Iceberg and Spark versions, which can be confirmed on the page of Iceberg multi engine support.

Configurations

To activate functionality of Iceberg, we can set the following configurations:

```
spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkCatalog
spark.sql.catalog.spark_catalog.type=hive
spark.sql.catalog.spark_catalog.uri=thrift://metastore-host:port
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

Iceberg Operations

Taking `CREATE TABLE` as a example,

```
CREATE TABLE foo (
  id bigint COMMENT 'unique id',
  data string)
USING iceberg;
```

Taking `SELECT` as a example,

```
SELECT * FROM foo;
```

Taking `INSERT` as a example,

```
INSERT INTO foo VALUES (1, 'a'), (2, 'b'), (3, 'c');
```

Taking `UPDATE` as a example, Spark 3.1 added support for UPDATE queries that update matching rows in tables.

```
UPDATE foo SET data = 'd', id = 4 WHERE id >= 3 and id < 4;
```

Taking `DELETE FROM` as a example, Spark 3 added support for DELETE FROM queries to remove data from tables.

```
DELETE FROM foo WHERE id >= 1 and id < 2;
```

Taking `MERGE INTO` as a example,

```
MERGE INTO target_table t
USING source_table s
ON t.id = s.id
WHEN MATCHED AND s.opType = 'delete' THEN DELETE
WHEN MATCHED AND s.opType = 'update' THEN UPDATE SET id = s.id, data = s.data
WHEN NOT MATCHED AND s.opType = 'insert' THEN INSERT (id, data) VALUES (s.id, s.data);
```

Kudu

What is Apache Kudu

> A new addition to the open source Apache Hadoop ecosystem, Apache Kudu completes Hadoop's storage layer to enable fast analytics on fast data.

When you are reading this documentation, we suppose that you are not necessary to be familiar with Apache Kudu. But at least, you have one running Kudu cluster which is able to be connected for you. And it is even better for you to understand what Apache Kudu is capable with.

Anything missing on this page about Apache Kudu background knowledge, you can refer to its official website.

Why Kyuubi on Kudu

Basically, Kyuubi can take place of HiveServer2 as a multi tenant ad-hoc SQL on Hadoop solution, with the advantages of speed and power coming from Spark SQL. You can run SQL queries towards both data source and Hive tables whose data is secured only with computing resources you are authorized.

> Spark SQL supports operating on a variety of data sources through the DataFrame interface. A DataFrame can be operated on using relational transformations and can also be used to create a temporary view. Registering a DataFrame as a temporary view allows you to run SQL queries over its data. This section describes the general methods for loading and saving data using the Spark Data Sources and then goes into specific options that are available for the built-in data sources.

In Kyuubi, we can register Kudu tables and other data source tables as Spark temporary views to enable federated union queries across Hive, Kudu, and other data sources.

Kudu Integration with Apache Spark

Before integrating Kyuubi with Kudu, we strongly suggest that you integrate and test Spark with Kudu first. You may find the guide from Kudu's online documentation – Kudu Integration with Spark

Kudu Integration with Kyuubi

Install Kudu Spark Dependency

Confirm your Kudu cluster version and download the corresponding kudu spark dependency library, such as org.apache.kudu:kudu-spark3_2.12-1.14.0 to $SPARK_HOME/jars.

Start Kyuubi

Now, you can start Kyuubi server with this kudu embedded Spark distribution.

Start Beeline Or Other Client You Prefer

```
bin/beeline -u 'jdbc:hive2://<host>:<port>/;principal=<if kerberized>;#spark.yarn.
↪queue=kyuubi_test'
```

Register Kudu table as Spark Temporary view

```
CREATE TEMPORARY VIEW kudutest
USING kudu
options (
  kudu.master "ip1:port1,ip2:port2,...",
  kudu.table "kudu::test.testtbl")
```

```
0: jdbc:hive2://spark5.jd.163.org:10009/> show tables;
19/07/09 15:28:03 INFO ExecuteStatementInClientMode: Running query 'show tables' with␣
↪1104328b-515c-4f8b-8a68-1c0b202bc9ed
```

(continues on next page)

(continued from previous page)

```
19/07/09 15:28:03 INFO KyuubiSparkUtil$: Application application_1560304876299_3805060␣
↪has been activated
19/07/09 15:28:03 INFO ExecuteStatementInClientMode: Executing query in incremental mode,
↪ running 1 jobs before optimization
19/07/09 15:28:03 INFO ExecuteStatementInClientMode: Executing query in incremental mode,
↪ running 1 jobs without optimization
19/07/09 15:28:03 INFO DAGScheduler: Asked to cancel job group 1104328b-515c-4f8b-8a68-
↪1c0b202bc9ed
+-----------+-----------------------------+--------------+--+
| database  |          tableName          | isTemporary  |
+-----------+-----------------------------+--------------+--+
| kyuubi    | hive_tbl                    | false        |
|           | kudutest                    | true         |
+-----------+-----------------------------+--------------+--+
2 rows selected (0.29 seconds)
```

Query Kudu Table

```
0: jdbc:hive2://spark5.jd.163.org:10009/> select * from kudutest;
19/07/09 15:25:17 INFO ExecuteStatementInClientMode: Running query 'select * from␣
↪kudutest' with ac3e8553-0d79-4c57-add1-7d3ffe34ba16
19/07/09 15:25:17 INFO KyuubiSparkUtil$: Application application_1560304876299_3805060␣
↪has been activated
19/07/09 15:25:17 INFO ExecuteStatementInClientMode: Executing query in incremental mode,
↪ running 3 jobs before optimization
19/07/09 15:25:17 INFO ExecuteStatementInClientMode: Executing query in incremental mode,
↪ running 3 jobs without optimization
19/07/09 15:25:17 INFO DAGScheduler: Asked to cancel job group ac3e8553-0d79-4c57-add1-
↪7d3ffe34ba16
+---------+---------------+----------------+--+
| userid  | sharesetting  | notifysetting  |
+---------+---------------+----------------+--+
| 1       | 1             | 1              |
| 5       | 5             | 5              |
| 2       | 2             | 2              |
| 3       | 3             | 3              |
| 4       | 4             | 4              |
+---------+---------------+----------------+--+
5 rows selected (1.083 seconds)
```

Join Kudu table with Hive table

```
0: jdbc:hive2://spark5.jd.163.org:10009/> select t1.*, t2.* from hive_tbl t1 join␣
↪kudutest t2 on t1.userid=t2.userid+1;
19/07/09 15:31:01 INFO ExecuteStatementInClientMode: Running query 'select t1.*, t2.*␣
↪from hive_tbl t1 join kudutest t2 on t1.userid=t2.userid+1' with 6982fa5c-29fa-49be-
↪a5bf-54c935bbad18
19/07/09 15:31:01 INFO KyuubiSparkUtil$: Application application_1560304876299_3805060␣
↪has been activated
```

(continues on next page)

(continued from previous page)

```
<omitted lines.... >
19/07/09 15:31:01 INFO DAGScheduler: Asked to cancel job group 6982fa5c-29fa-49be-a5bf-
↪54c935bbad18
+---------+---------------+----------------+---------+---------------+----------------+--
↪+
| userid  | sharesetting  | notifysetting  | userid  | sharesetting  | notifysetting  |
+---------+---------------+----------------+---------+---------------+----------------+--
↪+
| 2       | 2             | 2              | 1       | 1             | 1              |
| 3       | 3             | 3              | 2       | 2             | 2              |
| 4       | 4             | 4              | 3       | 3             | 3              |
+---------+---------------+----------------+---------+---------------+----------------+--
↪+
3 rows selected (1.63 seconds)
```

Insert to Kudu table

You should notice that only `INSERT INTO` is supported by Kudu, `OVERWRITE` data is not supported

```
0: jdbc:hive2://spark5.jd.163.org:10009/> insert overwrite table kudutest select *  from␣
↪hive_tbl;
19/07/09 15:35:29 INFO ExecuteStatementInClientMode: Running query 'insert overwrite␣
↪table kudutest select *  from hive_tbl' with 1afdb791-1aa7-4ceb-8ba8-ff53c17615d1
19/07/09 15:35:29 INFO KyuubiSparkUtil$: Application application_1560304876299_3805060␣
↪has been activated
19/07/09 15:35:30 ERROR ExecuteStatementInClientMode:
Error executing query as bdms_hzyaoqin,
insert overwrite table kudutest select *  from hive_tbl
Current operation state RUNNING,
java.lang.UnsupportedOperationException: overwrite is not yet supported
        at org.apache.kudu.spark.kudu.KuduRelation.insert(DefaultSource.scala:424)
        at org.apache.spark.sql.execution.datasources.InsertIntoDataSourceCommand.
↪run(InsertIntoDataSourceCommand.scala:42)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.sideEffectResult
↪$lzycompute(commands.scala:70)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.
↪sideEffectResult(commands.scala:68)
        at org.apache.spark.sql.execution.command.ExecutedCommandExec.
↪executeCollect(commands.scala:79)
        at org.apache.spark.sql.Dataset$$anonfun$6.apply(Dataset.scala:190)
        at org.apache.spark.sql.Dataset$$anonfun$6.apply(Dataset.scala:190)
        at org.apache.spark.sql.Dataset$$anonfun$52.apply(Dataset.scala:3259)
        at org.apache.spark.sql.execution.SQLExecution$.withNewExecutionId(SQLExecution.
↪scala:77)
        at org.apache.spark.sql.Dataset.withAction(Dataset.scala:3258)
        at org.apache.spark.sql.Dataset.<init>(Dataset.scala:190)
        at org.apache.spark.sql.Dataset$.ofRows(Dataset.scala:75)
        at org.apache.spark.sql.SparkSQLUtils$.toDataFrame(SparkSQLUtils.scala:39)
        at org.apache.kyuubi.operation.statement.ExecuteStatementInClientMode.
↪execute(ExecuteStatementInClientMode.scala:152)
        at org.apache.kyuubi.operation.statement.ExecuteStatementOperation$$anon$1$$anon
↪$2.run(ExecuteStatementOperation.scala:74)
```

(continues on next page)

(continued from previous page)

```
        at org.apache.kyuubi.operation.statement.ExecuteStatementOperation$$anon$1$$anon
↪$2.run(ExecuteStatementOperation.scala:70)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.
↪java:1698)
        at org.apache.kyuubi.operation.statement.ExecuteStatementOperation$$anon$1.
↪run(ExecuteStatementOperation.scala:70)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.
↪java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.
↪java:617)
        at java.lang.Thread.run(Thread.java:745)


19/07/09 15:35:30 INFO DAGScheduler: Asked to cancel job group 1afdb791-1aa7-4ceb-8ba8-
↪ff53c17615d1
```

```
0: jdbc:hive2://spark5.jd.163.org:10009/> insert into table kudutest select * from hive_
↪tbl;
19/07/09 15:36:26 INFO ExecuteStatementInClientMode: Running query 'insert into table␣
↪kudutest select *  from hive_tbl' with f7460400-0564-4f98-93b6-ad76e579e7af
19/07/09 15:36:26 INFO KyuubiSparkUtil$: Application application_1560304876299_3805060␣
↪has been activated
<omitted lines ...>
19/07/09 15:36:27 INFO DAGScheduler: ResultStage 36 (foreachPartition at KuduContext.
↪scala:332) finished in 0.322 s
19/07/09 15:36:27 INFO DAGScheduler: Job 36 finished: foreachPartition at KuduContext.
↪scala:332, took 0.324586 s
19/07/09 15:36:27 INFO KuduContext: completed upsert ops: duration histogram: 33.
↪333333333333336%: 2ms, 66.66666666666667%: 64ms, 100.0%: 102ms, 100.0%: 102ms
19/07/09 15:36:27 INFO ExecuteStatementInClientMode: Executing query in incremental mode,
↪ running 1 jobs before optimization
19/07/09 15:36:27 INFO ExecuteStatementInClientMode: Executing query in incremental mode,
↪ running 1 jobs without optimization
19/07/09 15:36:27 INFO DAGScheduler: Asked to cancel job group f7460400-0564-4f98-93b6-
↪ad76e579e7af
+---------+--+
| Result  |
+---------+--+
+---------+--+
No rows selected (0.611 seconds)
```

References

https://kudu.apache.org/ https://kudu.apache.org/docs/developing.html#_kudu_integration_with_spark https://github.com/apache/kyuubi https://spark.apache.org/docs/latest/sql-data-sources.html

Hive

The Kyuubi Hive Connector is a datasource for both reading and writing Hive table, It is implemented based on Spark DataSource V2, and supports concatenating multiple Hive metastore at the same time.

This connector can be used to federate queries of multiple hives warehouse in a single Spark cluster.

Hive Connector Integration

To enable the integration of kyuubi spark sql engine and Hive connector through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the Hive connector *dependencies*
- Setting the spark extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with Hive connector supported consists of

1. kyuubi-spark-connector-hive_2.12-1.7.0, the hive connector jar deployed with Kyuubi distributions
2. a copy of spark distribution

In order to make the Hive connector packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the Kyuubi Hive connector packages into `$SPARK_HOME/jars` directly
2. Set `spark.jars=/path/to/kyuubi-hive-connector`

Configurations

To activate functionality of Kyuubi Hive connector, we can set the following configurations:

```
spark.sql.catalog.hive_catalog     org.apache.kyuubi.spark.connector.hive.
↪HiveTableCatalog
spark.sql.catalog.hive_catalog.spark.sql.hive.metastore.version     hive-metastore-
↪version
spark.sql.catalog.hive_catalog.hive.metastore.uris     thrift://metastore-host:port
spark.sql.catalog.hive_catalog.hive.metastore.port     port
spark.sql.catalog.hive_catalog.spark.sql.hive.metastore.jars     path
spark.sql.catalog.hive_catalog.spark.sql.hive.metastore.jars.path     file:///opt/hive1/
↪lib/*.jar
```

Tip: For details about the multi-version Hive configuration, see the related multi-version Hive configurations supported by Apache Spark.

Hive Connector Operations

Taking `CREATE NAMESPACE` as a example,

```
CREATE NAMESPACE ns;
```

Taking `CREATE TABLE` as a example,

```
CREATE TABLE hive_catalog.ns.foo (
  id bigint COMMENT 'unique id',
  data string)
USING parquet;
```

Taking `SELECT` as a example,

```
SELECT * FROM hive_catalog.ns.foo;
```

Taking `INSERT` as a example,

```
INSERT INTO hive_catalog.ns.foo VALUES (1, 'a'), (2, 'b'), (3, 'c');
```

Taking `DROP TABLE` as a example,

```
DROP TABLE hive_catalog.ns.foo;
```

Taking `DROP NAMESPACE` as a example,

```
DROP NAMESPACE hive_catalog.ns;
```

Flink Table Store

Flink Table Store is a unified storage to build dynamic tables for both streaming and batch processing in Flink, supporting high-speed data ingestion and timely data query.

Tip: This article assumes that you have mastered the basic knowledge and operation of Flink Table Store. For the knowledge about Flink Table Store not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Flink Table Store which is more convenient, easy to understand, and easy to expand than directly using spark to manipulate Flink Table Store.

Flink Table Store Integration

To enable the integration of kyuubi spark sql engine and Flink Table Store through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the Flink Table Store *dependencies*
- Setting the spark extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with Flink Table Store supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of spark distribution
3. flink-table-store-spark-<version>.jar (example: flink-table-store-spark-0.2.jar), which can be found in the Maven Central

In order to make the Flink Table Store packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the Flink Table Store packages into `$SPARK_HOME/jars` directly
2. Set `spark.jars=/path/to/flink-table-store-spark`

Warning: Please mind the compatibility of different Flink Table Store and Spark versions, which can be confirmed on the page of Flink Table Store multi engine support.

Configurations

To activate functionality of Flink Table Store, we can set the following configurations:

```
spark.sql.catalog.tablestore=org.apache.flink.table.store.spark.SparkCatalog
spark.sql.catalog.tablestore.warehouse=file:/tmp/warehouse
```

Flink Table Store Operations

Flink Table Store supports reading table store tables through Spark. A common scenario is to write data with Flink and read data with Spark. You can follow this document Flink Table Store Quick Start to write data to a table store table and then use kyuubi spark sql engine to query the table with the following SQL `SELECT` statement.

```
select * from table_store.default.word_count;
```

TiDB

TiDB is an open-source NewSQL database that supports Hybrid Transactional and Analytical Processing (HTAP) workloads.

TiSpark is a thin layer built for running Apache Spark on top of TiDB/TiKV to answer complex OLAP queries. It enjoys the merits of both the Spark platform and the distributed clusters of TiKV while seamlessly integrated to TiDB to provide one-stop HTAP solutions for online transactions and analyses.

Tip: This article assumes that you have mastered the basic knowledge and operation of TiDB and TiSpark. For the knowledge not mentioned in this article, you can obtain it from TiDB Official Documentation.

By using kyuubi, we can run SQL queries towards TiDB/TiKV which is more convenient, easy to understand, and easy to expand than directly using spark to manipulate TiDB/TiKV.

TiDB Integration

To enable the integration of kyuubi spark sql engine and TiDB through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the TiSpark *dependencies*
- Setting the spark extension and catalog *configurations*

Dependencies

The classpath of kyuubi spark sql engine with TiDB supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of spark distribution
3. tispark-assembly-<spark.version>_<scala.version>-<tispark.version>.jar (example: tispark-assembly-3.2_2.12-3.0.1.jar), which can be found in the Maven Central

In order to make the TiSpark packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the TiSpark packages into `$SPARK_HOME/jars` directly
2. Set `spark.jars=/path/to/tispark-assembly`

Warning: Please mind the compatibility of different TiDB, TiSpark and Spark versions, which can be confirmed on the page of TiSpark Environment setup.

Configurations

To activate functionality of TiSpark, we can set the following configurations:

```
spark.tispark.pd.addresses $pd_host:$pd_port
spark.sql.extensions org.apache.spark.sql.TiExtensions
spark.sql.catalog.tidb_catalog  org.apache.spark.sql.catalyst.catalog.TiCatalog
spark.sql.catalog.tidb_catalog.pd.addresses $pd_host:$pd_port
```

The *spark.tispark.pd.addresses* and *spark.sql.catalog.tidb_catalog.pd.addresses* configurations allow you to put in multiple PD servers. Specify the port number for each of them.

For example, when you have multiple PD servers on *10.16.20.1,10.16.20.2,10.16.20.3* with the port *2379*, put it as *10.16.20.1:2379,10.16.20.2:2379,10.16.20.3:2379*.

TiDB Operations

Taking `SELECT` as a example,

```
SELECT * FROM foo;
```

Taking `DELETE FROM` as a example, Spark 3 added support for DELETE FROM queries to remove data from tables.

```
DELETE FROM foo WHERE id >= 1 and id < 2;
```

Note: As for now (TiSpark 3.0.1), TiSpark does not support `CREATE TABLE`, `INSERT INTO/OVERWRITE` operations through Apache Spark Datasource V2 and Catalog APIs.

TPC-DS

The TPC-DS is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance.

Tip: This article assumes that you have mastered the basic knowledge and operation of *TPC-DS*. For the knowledge about TPC-DS not mentioned in this article, you can obtain it from its Official Documentation.

This connector can be used to test the capabilities and query syntax of Spark without configuring access to an external data source. When you query a TPC-DS table, the connector generates the data on the fly using a deterministic algorithm.

Goto Try Kyuubi to explore TPC-DS data instantly!

TPC-DS Integration

To enable the integration of kyuubi spark sql engine and TPC-DS through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the TPC-DS connector *dependencies*
- Setting the spark catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with TPC-DS supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of spark distribution
3. kyuubi-spark-connector-tpcds-1.7.0_2.12.jar, which can be found in the Maven Central

In order to make the TPC-DS connector package visible for the runtime classpath of engines, we can use one of these methods:

1. Put the TPC-DS connector package into `$SPARK_HOME/jars` directly
2. Set spark.jars=kyuubi-spark-connector-tpcds-1.7.0_2.12.jar

Configurations

To add TPC-DS tables as a catalog, we can set the following configurations in `$SPARK_HOME/conf/spark-defaults.conf`:

```
# (required) Register a catalog named `tpcds` for the spark engine.
spark.sql.catalog.tpcds=org.apache.kyuubi.spark.connector.tpcds.TPCDSCatalog

# (optional) Excluded database list from the catalog, all available databases are:
#            sf0, tiny, sf1, sf10, sf30, sf100, sf300, sf1000, sf3000, sf10000, sf30000,␣
↪sf100000.
spark.sql.catalog.tpcds.excludeDatabases=sf10000,sf30000

# (optional) When true, use CHAR/VARCHAR, otherwise use STRING. It affects output of the␣
↪table schema,
#            e.g. `SHOW CREATE TABLE <table>`, `DESC <table>`.
spark.sql.catalog.tpcds.useAnsiStringType=false

# (optional) TPCDS changed table schemas in v2.6.0, turn off this option to use old␣
↪table schemas.
#            See detail at: https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-
↪ds_v3.2.0.pdf
spark.sql.catalog.tpcds.useTableSchema_2_6=true

# (optional) Maximum bytes per task, consider reducing it if you want higher parallelism.
spark.sql.catalog.tpcds.read.maxPartitionBytes=128m
```

TPC-DS Operations

Listing databases under *tpcds* catalog.

```
SHOW DATABASES IN tpcds;
```

Listing tables under *tpcds.sf1* database.

```
SHOW TABLES IN tpcds.sf1;
```

Switch current database to *tpcds.sf1* and run a query against it.

```
USE tpcds.sf1;
SELECT * FROM orders;
```

TPC-H

The TPC-H is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance.

Tip: This article assumes that you have mastered the basic knowledge and operation of *TPC-H*. For the knowledge about TPC-H not mentioned in this article, you can obtain it from its Official Documentation.

This connector can be used to test the capabilities and query syntax of Spark without configuring access to an external data source. When you query a TPC-H table, the connector generates the data on the fly using a deterministic algorithm.

Goto Try Kyuubi to explore TPC-H data instantly!

TPC-H Integration

To enable the integration of kyuubi spark sql engine and TPC-H through Apache Spark Datasource V2 and Catalog APIs, you need to:

- Referencing the TPC-H connector *dependencies*
- Setting the spark catalog *configurations*

Dependencies

The **classpath** of kyuubi spark sql engine with TPC-H supported consists of

1. kyuubi-spark-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of spark distribution
3. kyuubi-spark-connector-tpch-1.7.0_2.12.jar, which can be found in the Maven Central

In order to make the TPC-H connector package visible for the runtime classpath of engines, we can use one of these methods:

1. Put the TPC-H connector package into `$SPARK_HOME/jars` directly
2. Set spark.jars=kyuubi-spark-connector-tpch-1.7.0_2.12.jar

Configurations

To add TPC-H tables as a catalog, we can set the following configurations in `$SPARK_HOME/conf/spark-defaults.conf`:

```
# (required) Register a catalog named `tpch` for the spark engine.
spark.sql.catalog.tpch=org.apache.kyuubi.spark.connector.tpch.TPCHCatalog

# (optional) Excluded database list from the catalog, all available databases are:
#            sf0, tiny, sf1, sf10, sf30, sf100, sf300, sf1000, sf3000, sf10000, sf30000,␣
↪sf100000.
spark.sql.catalog.tpch.excludeDatabases=sf10000,sf30000

# (optional) When true, use CHAR/VARCHAR, otherwise use STRING. It affects output of the␣
↪table schema,
#            e.g. `SHOW CREATE TABLE <table>`, `DESC <table>`.
spark.sql.catalog.tpch.useAnsiStringType=false

# (optional) Maximum bytes per task, consider reducing it if you want higher parallelism.
spark.sql.catalog.tpch.read.maxPartitionBytes=128m
```

TPC-H Operations

Listing databases under *tpch* catalog.

```
SHOW DATABASES IN tpch;
```

Listing tables under *tpch.sf1* database.

```
SHOW TABLES IN tpch.sf1;
```

Switch current database to *tpch.sf1* and run a query against it.

```
USE tpch.sf1;
SELECT * FROM orders;
```

SQL Lineage Support

When we execute a SQL statement, the computing engine will return to a result set, each column in the result set may originate from different tables, and these different tables depend on other tables, which may have been calculated by functions, aggregation, etc. The source table is related to the result set, which is the SQL Lineage.

Introduction

The current lineage parsing functionality is implemented as a plugin by extending Spark's `QueryExecutionListener`.

1. The `SparkListenerSQLExecutionEnd` event is triggered after the SQL execution is finished and captured by the `QueryExecuctionListener`, where the SQL lineage parsing process is performed on the successfully executed SQL.
2. Will write the parsed lineage information to the log file in JSON format.

Example

When the following SQL is executed:

```
## table
create table test_table0(a string, b string)

## query
select a as col0, b as col1 from test_table0
```

The lineage of this SQL:

```
{
   "inputTables": ["default.test_table0"],
   "outputTables": [],
   "columnLineage": [{
      "column": "col0",
      "originalColumns": ["default.test_table0.a"]
   }, {
      "column": "col1",
```

(continues on next page)

(continued from previous page)

```
      "originalColumns": ["default.test_table0.b"]
   }]
}
```

Lineage specific identification

- `__count__`. Means that the column is an `count(*)` aggregate expression and cannot extract the specific column. Lineage of the column like `default.test_table0.__count__`.
- `__local__`. Means that the lineage of the table is a `LocalRelation` and not the real table, like `__local__.a`

SQL type support

Currently supported column lineage for spark's `Command` and `Query` type:

Query

- `Select`

Command

- `AlterViewAsCommand`
- `AppendData`
- `CreateDataSourceTableAsSelectCommand`
- `CreateHiveTableAsSelectCommand`
- `CreateTableAsSelect`
- `CreateViewCommand`
- `InsertIntoDataSourceCommand`
- `InsertIntoDataSourceDirCommand`
- `InsertIntoHadoopFsRelationCommand`
- `InsertIntoHiveDirCommand`
- `InsertIntoHiveTable`
- `MergeIntoTable`
- `OptimizedCreateHiveTableAsSelectCommand`
- `OverwriteByExpression`
- `OverwritePartitionsDynamic`
- `ReplaceTableAsSelect`
- `SaveIntoDataSourceCommand`

Building

Build with Apache Maven

Kyuubi Spark Lineage Listener Extension is built using Apache Maven. To build it, `cd` to the root direct of kyuubi project and run:

```
build/mvn clean package -pl :kyuubi-spark-lineage_2.12 -DskipTests
```

After a while, if everything goes well, you will get the plugin finally in two parts:

- The main plugin jar, which is under `./extensions/spark/kyuubi-spark-lineage/target/kyuubi-spark-lineage_${scala.binary.version}-${project.version}.jar`
- The least transitive dependencies needed, which are under `./extensions/spark/kyuubi-spark-lineage/target/scala-${scala.binary.version}/jars`

Build against Different Apache Spark Versions

The maven option `spark.version` is used for specifying Spark version to compile with and generate corresponding transitive dependencies. By default, it is always built with the latest `spark.version` defined in kyuubi project main pom file. Sometimes, it may be incompatible with other Spark distributions, then you may need to build the plugin on your own targeting the Spark version you use.

For example,

```
build/mvn clean package -pl :kyuubi-spark-lineage_2.12 -DskipTests -Dspark.version=3.1.2
```

The available `spark.version`s are shown in the following table.

Currently, Spark released with Scala 2.12 are supported.

Test with ScalaTest Maven plugin

If you omit `-DskipTests` option in the command above, you will also get all unit tests run.

```
build/mvn clean package -pl :kyuubi-spark-lineage_2.12
```

If any bug occurs and you want to debug the plugin yourself, you can configure `-DdebugForkedProcess=true` and `-DdebuggerPort=5005`(optional).

```
build/mvn clean package -pl :kyuubi-spark-lineage_2.12 -DdebugForkedProcess=true
```

The tests will suspend at startup and wait for a remote debugger to attach to the configured port.

We will appreciate if you can share the bug or the fix to the Kyuubi community.

Installing

With the `kyuubi-spark-lineage_*.jar` and its transitive dependencies available for spark runtime classpath, such as

- Copied to `$SPARK_HOME/jars`, or
- Specified to `spark.jars` configuration

Configure

Settings for Spark Listener Extensions

Add `org.apache.kyuubi.plugin.lineage.SparkOperationLineageQueryExecutionListener` to the spark configuration `spark.sql.queryExecutionListeners`.

```
spark.sql.queryExecutionListeners=org.apache.kyuubi.plugin.lineage.
↪SparkOperationLineageQueryExecutionListener
```

Settings for Lineage Logger and Path

Lineage Logger Path

The location of all the engine operation lineage events go for the builtin JSON logger. We first need set `kyuubi.engine.event.loggers` to `JSON`. All operation lineage events will be written in the unified event json logger path, which be setting with `kyuubi.engine.event.json.log.path`. We can get the lineage logger from the `operation_lineage` dir in the `kyuubi.engine.event.json.log.path`.

Hive Dialect Support

Hive Dialect plugin aims to provide Hive Dialect support to Spark's JDBC source. It will auto registered to Spark and applied to JDBC sources with url prefix of `jdbc:hive2://` or `jdbc:kyuubi://`.

Hive Dialect helps to solve failures access Kyuubi. It fails and unexpected results when querying data from Kyuubi as JDBC source with Hive JDBC Driver or Kyuubi Hive JDBC Driver in Spark, as Spark JDBC provides no Hive Dialect support out of box and quoting columns and other identifiers in ANSI as "table.column" rather than in HiveSQL style as `table`.`column`.

Features

- quote identifier in Hive SQL style

  eg. Quote `table.column` in `table`.`column`

Usage

1. Get the Kyuubi Hive Dialect Extension jar
    1. compile the extension by executing `build/mvn clean package -pl :kyuubi-extension-spark-jdbc-dialect_2.12 -DskipTests`
    2. get the extension jar under `extensions/spark/kyuubi-extension-spark-jdbc-dialect/target`
    3. If you like, you can compile the extension jar with the corresponding Maven's profile on you compile command, i.e. you can get extension jar for Spark 3.2 by compiling with `-Pspark-3.1`
2. Put the Kyuubi Hive Dialect Extension jar `kyuubi-extension-spark-jdbc-dialect_-*.jar` into `$SPARK_HOME/jars`
3. Enable `KyuubiSparkJdbcDialectExtension`, by setting `spark.sql.extensions=org.apache.spark.sql.dialect.KyuubiSparkJdbcDialectExtension`, i.e.
    - add a config into `$SPARK_HOME/conf/spark-defaults.conf`
    - or add setting config in SparkSession builder

Extensions for Flink

Connectors For Flink SQL Query Engine

Flink Table Store

Flink Table Store is a unified storage to build dynamic tables for both streaming and batch processing in Flink, supporting high-speed data ingestion and timely data query.

Tip: This article assumes that you have mastered the basic knowledge and operation of Flink Table Store. For the knowledge about Flink Table Store not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Flink Table Store which is more convenient, easy to understand, and easy to expand than directly using flink to manipulate Flink Table Store.

Flink Table Store Integration

To enable the integration of kyuubi flink sql engine and Flink Table Store, you need to:

- Referencing the Flink Table Store *dependencies*

Dependencies

The **classpath** of kyuubi flink sql engine with Flink Table Store supported consists of

1. kyuubi-flink-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of flink distribution
3. flink-table-store-dist-<version>.jar (example: flink-table-store-dist-0.2.jar), which can be found in the Maven Central

In order to make the Flink Table Store packages visible for the runtime classpath of engines, we can use these methods:

1. Put the Flink Table Store packages into `$FLINK_HOME/lib` directly
2. Setting the HADOOP_CLASSPATH environment variable or copy the Pre-bundled Hadoop Jar to flink/lib.

Warning: Please mind the compatibility of different Flink Table Store and Flink versions, which can be confirmed on the page of Flink Table Store multi engine support.

Flink Table Store Operations

Taking `CREATE CATALOG` as a example,

```
CREATE CATALOG my_catalog WITH (
  'type'='table-store',
  'warehouse'='hdfs://nn:8020/warehouse/path' -- or 'file:///tmp/foo/bar'
);

USE CATALOG my_catalog;
```

Taking `CREATE TABLE` as a example,

```
CREATE TABLE MyTable (
  user_id BIGINT,
  item_id BIGINT,
  behavior STRING,
  dt STRING,
  PRIMARY KEY (dt, user_id) NOT ENFORCED
) PARTITIONED BY (dt) WITH (
  'bucket' = '4'
);
```

Taking `Query Table` as a example,

```
SET 'execution.runtime-mode' = 'batch';
SELECT * FROM orders WHERE catalog_id=1025;
```

Taking `Streaming Query` as a example,

```
SET 'execution.runtime-mode' = 'streaming';
SELECT * FROM MyTable /*+ OPTIONS ('log.scan'='latest') */;
```

Taking `Rescale Bucket` as a example,

```
ALTER TABLE my_table SET ('bucket' = '4');
INSERT OVERWRITE my_table PARTITION (dt = '2022-01-01');
```

Hudi

Apache Hudi (pronounced "hoodie") is the next generation streaming data lake platform. Apache Hudi brings core warehouse and database functionality directly to a data lake.

Tip: This article assumes that you have mastered the basic knowledge and operation of Hudi. For the knowledge about Hudi not mentioned in this article, you can obtain it from its Official Documentation.

By using Kyuubi, we can run SQL queries towards Hudi which is more convenient, easy to understand, and easy to expand than directly using flink to manipulate Hudi.

Hudi Integration

To enable the integration of kyuubi flink sql engine and Hudi through Catalog APIs, you need to:

- Referencing the Hudi *dependencies*

Dependencies

The **classpath** of kyuubi flink sql engine with Hudi supported consists of

1. kyuubi-flink-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of flink distribution
3. hudi-flink<flink.version>-bundle_<scala.version>-<hudi.version>.jar (example: hudi-flink1.14-bundle_2.12-0.11.1.jar), which can be found in the Maven Central

In order to make the Hudi packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the Hudi packages into `$flink_HOME/lib` directly
2. Set `pipeline.jars=/path/to/hudi-flink-bundle`

Hudi Operations

Taking `Create Table` as a example,

```
CREATE TABLE t1 (
  id INT PRIMARY KEY NOT ENFORCED,
  name STRING,
  price DOUBLE
) WITH (
  'connector' = 'hudi',
  'path' = 's3://bucket-name/hudi/',
  'table.type' = 'MERGE_ON_READ' -- this creates a MERGE_ON_READ table, by default is␣
↪COPY_ON_WRITE
);
```

Taking `Query Data` as a example,

```
SELECT * FROM t1;
```

Taking `Insert and Update Data` as a example,

```
INSERT INTO t1 VALUES (1, 'Lucas' , 2.71828);
```

Taking `Streaming Query` as a example,

```
CREATE TABLE t1 (
  uuid VARCHAR(20) PRIMARY KEY NOT ENFORCED,
  name VARCHAR(10),
  age INT,
  ts TIMESTAMP(3),
  `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
  'connector' = 'hudi',
  'path' = '${path}',
  'table.type' = 'MERGE_ON_READ',
  'read.streaming.enabled' = 'true',  -- this option enable the streaming read
  'read.start-commit' = '20210316134557', -- specifies the start commit instant time
  'read.streaming.check-interval' = '4' -- specifies the check interval for finding new␣
↪source commits, default 60s.
);

-- Then query the table in stream mode
SELECT * FROM t1;
```

Taking `Delete Data`,

The streaming query can implicitly auto delete data. When consuming data in streaming query, Hudi Flink source can also accepts the change logs from the underneath data source, it can then applies the UPDATE and DELETE by per-row level.

Iceberg

Apache Iceberg is an open table format for huge analytic datasets. Iceberg adds tables to compute engines including Spark, Trino, PrestoDB, Flink, Hive and Impala using a high-performance table format that works just like a SQL table.

Tip: This article assumes that you have mastered the basic knowledge and operation of Iceberg. For the knowledge about Iceberg not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Iceberg which is more convenient, easy to understand, and easy to expand than directly using flink to manipulate Iceberg.

Iceberg Integration

To enable the integration of kyuubi flink sql engine and Iceberg through Catalog APIs, you need to:

- Referencing the Iceberg *dependencies*

Dependencies

The **classpath** of kyuubi flink sql engine with Iceberg supported consists of

1. kyuubi-flink-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of flink distribution
3. iceberg-flink-runtime-<flink.version>-<iceberg.version>.jar (example: iceberg-flink-runtime-1.14-0.14.0.jar), which can be found in the Maven Central

In order to make the Iceberg packages visible for the runtime classpath of engines, we can use one of these methods:

1. Put the Iceberg packages into `$FLINK_HOME/lib` directly
2. Set `pipeline.jars=/path/to/iceberg-flink-runtime`

Warning: Please mind the compatibility of different Iceberg and Flink versions, which can be confirmed on the page of Iceberg multi engine support.

Iceberg Operations

Taking `CREATE CATALOG` as a example,

```
CREATE CATALOG hive_catalog WITH (
  'type'='iceberg',
  'catalog-type'='hive',
  'uri'='thrift://localhost:9083',
  'warehouse'='hdfs://nn:8020/warehouse/path'
);
USE CATALOG hive_catalog;
```

Taking `CREATE DATABASE` as a example,

```
CREATE DATABASE iceberg_db;
USE iceberg_db;
```

Taking `CREATE TABLE` as a example,

```
CREATE TABLE `hive_catalog`.`default`.`sample` (
  id BIGINT COMMENT 'unique id',
  data STRING
);
```

Taking `Batch Read` as a example,

```
SET execution.runtime-mode = batch;
SELECT * FROM sample;
```

Taking `Streaming Read` as a example,

```
SET execution.runtime-mode = streaming;
SELECT * FROM sample /*+ OPTIONS('streaming'='true', 'monitor-interval'='1s')*/ ;
```

Taking `INSERT INTO` as a example,

```
INSERT INTO `hive_catalog`.`default`.`sample` VALUES (1, 'a');
INSERT INTO `hive_catalog`.`default`.`sample` SELECT id, data from other_kafka_table;
```

Taking `INSERT OVERWRITE` as a example, Flink streaming job does not support INSERT OVERWRITE.

```
INSERT OVERWRITE `hive_catalog`.`default`.`sample` VALUES (1, 'a');
INSERT OVERWRITE `hive_catalog`.`default`.`sample` PARTITION(data='a') SELECT 6;
```

Warning: This page is still in-progress.

Extensions for Hive

Connectors for Hive SQL Query Engine

Flink Table Store

Flink Table Store is a unified storage to build dynamic tables for both streaming and batch processing in Flink, supporting high-speed data ingestion and timely data query.

Tip: This article assumes that you have mastered the basic knowledge and operation of Flink Table Store. For the knowledge about Flink Table Store not mentioned in this article, you can obtain it from its Official Documentation.

By using Kyuubi, we can run SQL queries towards Flink Table Store which is more convenient, easy to understand, and easy to expand than directly using Hive to manipulate Flink Table Store.

Flink Table Store Integration

To enable the integration of kyuubi flink sql engine and Flink Table Store, you need to:

- Referencing the Flink Table Store *dependencies*
- Setting the environment variable *configurations*

Dependencies

The **classpath** of kyuubi hive sql engine with Iceberg supported consists of

1. kyuubi-hive-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of hive distribution
3. flink-table-store-hive-connector-<flink-table-store.version>_<hive.binary.version>.jar (example: flink-table-store-hive-connector-0.4.0_3.1.jar), which can be found in the Installation Table Store in Hive

In order to make the Hive packages visible for the runtime classpath of engines, we can use one of these methods:

1. You can create an auxlib folder under the root directory of Hive, and copy flink-table-store-hive-connector-0.4.0_3.1.jar into auxlib.
2. Execute ADD JAR statement in the Kyuubi to add dependencies to Hive's auxiliary classpath. For example:

```
ADD JAR /path/to/flink-table-store-hive-connector-0.4.0_3.1.jar;
```

Warning: The second method is not recommended. If you're using the MR execution engine and running a join statement, you may be faced with the exception `org.apache.hive.com.esotericsoftware.kryo.kryoexception: unable to find class.`

Warning: Please mind the compatibility of different Flink Table Store and Hive versions, which can be confirmed on the page of Flink Table Store multi engine support.

Configurations

If you are using HDFS, make sure that the environment variable HADOOP_HOME or HADOOP_CONF_DIR is set.

Flink Table Store Operations

Flink Table Store only supports only reading table store tables through Hive. A common scenario is to write data with Flink and read data with Hive. You can follow this document Flink Table Store Quick Start to write data to a table store table and then use Kyuubi Hive SQL engine to query the table with the following SQL `SELECT` statement.

Taking `Query Data` as an example,

```
SELECT a, b FROM test_table ORDER BY a;
```

Taking `Query External Table` as an example,

```
CREATE EXTERNAL TABLE external_test_table
STORED BY 'org.apache.flink.table.store.hive.TableStoreHiveStorageHandler'
LOCATION '/path/to/table/store/warehouse/default.db/test_table';

SELECT a, b FROM test_table ORDER BY a;
```

Iceberg

Apache Iceberg is an open table format for huge analytic datasets. Iceberg adds tables to compute engines including Spark, Trino, PrestoDB, Flink, Hive and Impala using a high-performance table format that works just like a SQL table.

Tip: This article assumes that you have mastered the basic knowledge and operation of Iceberg. For the knowledge about Iceberg not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Iceberg which is more convenient, easy to understand, and easy to expand than directly using hive to manipulate Iceberg.

Iceberg Integration

To enable the integration of kyuubi hive sql engine and Iceberg, you need to:

- Referencing the Iceberg *dependencies*
- Setting the hive extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi hive sql engine with Iceberg supported consists of

1. kyuubi-hive-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of hive distribution
3. iceberg-hive-runtime-<hive.version>_<scala.version>-<iceberg.version>.jar (example: iceberg-hive-runtime-3.2_2.12-0.14.0.jar), which can be found in the Maven Central

In order to make the Iceberg packages visible for the runtime classpath of engines, we can use the method:

1. Execute ADD JAR statement in the Kyuubi to add dependencies to Hive's auxiliary classpath. For example:

```
ADD JAR /path/to/iceberg-hive-runtime.jar;
```

Warning: Please mind the compatibility of different Iceberg and Hive versions, which can be confirmed on the page of Iceberg multi engine support.

Configurations

To activate functionality of Iceberg, we can set the following configurations: Set `iceberg.engine.hive.enabled=true` in the Hadoop configuration. For example, setting this in the hive-site.xml

Iceberg Operations

Taking `CREATE TABLE` as a example,

```
CREATE TABLE x (i int) STORED BY 'org.apache.iceberg.mr.hive.HiveIcebergStorageHandler';
CREATE EXTERNAL TABLE x (i int) STORED BY 'org.apache.iceberg.mr.hive.
↪HiveIcebergStorageHandler';
```

Taking `SELECT` as a example,

```
SELECT * FROM foo;
```

Taking `INSERT` as a example,

```
INSERT INTO foo VALUES (1, 'a'), (2, 'b'), (3, 'c');
INSERT OVERWRITE TABLE target SELECT * FROM source;
```

Taking `ALTER` as a example,

```
ALTER TABLE t SET TBLPROPERTIES('...'='...');
```

Warning: This page is still in-progress.

Extensions for Trino

Warning: This page is still in-progress.

Connectors For Trino SQL Engine

Flink Table Store

Flink Table Store is a unified storage to build dynamic tables for both streaming and batch processing in Flink, supporting high-speed data ingestion and timely data query.

Tip: This article assumes that you have mastered the basic knowledge and operation of Flink Table Store. For the knowledge about Flink Table Store not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Flink Table Store which is more convenient, easy to understand, and easy to expand than directly using trino to manipulate Flink Table Store.

Flink Table Store Integration

To enable the integration of kyuubi trino sql engine and Flink Table Store, you need to:

- Referencing the Flink Table Store *dependencies*
- Setting the trino extension and catalog *configurations*

Dependencies

The **classpath** of kyuubi trino sql engine with Flink Table Store supported consists of

1. kyuubi-trino-sql-engine-1.7.0_2.12.jar, the engine jar deployed with Kyuubi distributions
2. a copy of trino distribution
3. flink-table-store-trino-<version>.jar (example: flink-table-store-trino-0.2.jar), which code can be found in the Source Code
4. flink-shaded-hadoop-2-uber-2.8.3-10.0.jar, which code can be found in the Pre-bundled Hadoop 2.8.3

In order to make the Flink Table Store packages visible for the runtime classpath of engines, we can use these methods:

1. Build the flink-table-store-trino-<version>.jar by reference to Flink Table Store Trino README
2. Put the flink-table-store-trino-<version>.jar and flink-shaded-hadoop-2-uber-2.8.3-10.0.jar packages into `$TRINO_SERVER_HOME/plugin/tablestore` directly

Warning: Please mind the compatibility of different Flink Table Store and Trino versions, which can be confirmed on the page of Flink Table Store multi engine support.

Configurations

To activate functionality of Flink Table Store, we can set the following configurations:

Catalogs are registered by creating a catalog properties file in the $TRINO_SERVER_HOME/etc/catalog directory. For example, create $TRINO_SERVER_HOME/etc/catalog/tablestore.properties with the following contents to mount the tablestore connector as the tablestore catalog:

```
connector.name=tablestore
warehouse=file:///tmp/warehouse
```

Flink Table Store Operations

Flink Table Store supports reading table store tables through Trino. A common scenario is to write data with Flink and read data with Trino. You can follow this document Flink Table Store Quick Start to write data to a table store table and then use kyuubi trino sql engine to query the table with the following SQL `SELECT` statement.

```
SELECT * FROM tablestore.default.t1
```

Hudi

Apache Hudi (pronounced "hoodie") is the next generation streaming data lake platform. Apache Hudi brings core warehouse and database functionality directly to a data lake.

Tip: This article assumes that you have mastered the basic knowledge and operation of Hudi. For the knowledge about Hudi not mentioned in this article, you can obtain it from its Official Documentation.

By using Kyuubi, we can run SQL queries towards Hudi which is more convenient, easy to understand, and easy to expand than directly using Trino to manipulate Hudi.

Hudi Integration

To enable the integration of Kyuubi Trino SQL engine and Hudi, you need to:

- Setting the Trino extension and catalog *configurations*

Configurations

Catalogs are registered by creating a file of catalog properties in the *$TRINO_SERVER_HOME/etc/catalog* directory. For example, we can create a *$TRINO_SERVER_HOME/etc/catalog/hudi.properties* with the following contents to mount the Hudi connector as a Hudi catalog:

```
connector.name=hudi
hive.metastore.uri=thrift://example.net:9083
```

Note: You need to replace $TRINO_SERVER_HOME above to your Trino server home path like */opt/trino-server-406*.

More configuration properties can be found in the Hudi connector in Trino document.

Tip: Trino version 398 or higher, it is recommended to use the Hudi connector. You don't need to install any dependencies in version 398 or higher.

Hudi Operations

The globally available and read operation statements are supported in Trino. These statements can be found in Trino SQL Support. Currently, Trino cannot write data to a Hudi table. A common scenario is to write data with Spark/Flink and read data with Trino. You can use the Kyuubi Trino SQL engine to query the table with the following SQL `SELECT` statement.

Taking `Query Data` as a example,

```
USE example.example_schema;

SELECT symbol, max(ts)
FROM stock_ticks_cow
GROUP BY symbol
HAVING symbol = 'GOOG';
```

Iceberg

Apache Iceberg is an open table format for huge analytic datasets. Iceberg adds tables to compute engines including Spark, Trino, PrestoDB, Flink, Hive and Impala using a high-performance table format that works just like a SQL table.

Tip: This article assumes that you have mastered the basic knowledge and operation of Iceberg. For the knowledge about Iceberg not mentioned in this article, you can obtain it from its Official Documentation.

By using kyuubi, we can run SQL queries towards Iceberg which is more convenient, easy to understand, and easy to expand than directly using Trino to manipulate Iceberg.

Iceberg Integration

To enable the integration of kyuubi trino sql engine and Iceberg through Catalog APIs, you need to:

- Setting the Trino extension and catalog *Configurations*

Configurations

To activate functionality of Iceberg, we can set the following configurations:

```
connector.name=iceberg
hive.metastore.uri=thrift://localhost:9083
```

Iceberg Operations

Taking `CREATE TABLE` as a example,

```
CREATE TABLE orders (
  orderkey bigint,
  orderstatus varchar,
  totalprice double,
  orderdate date
) WITH (
  format = 'ORC'
);
```

Taking `SELECT` as a example,

```
SELECT * FROM new_orders;
```

Taking `INSERT` as a example,

```
INSERT INTO cities VALUES (1, 'San Francisco');
```

Taking `UPDATE` as a example,

```
UPDATE purchases SET status = 'OVERDUE' WHERE ship_date IS NULL;
```

Taking `DELETE FROM` as a example,

```
DELETE FROM lineitem WHERE shipmode = 'AIR';
```

3.8 Connectors

This section describes the connectors available for different kyuubi engines to access data from various data sources.

Note: Is your connector missing? Report an feature request or help us document it.

3.9 Overview

3.9.1 Kyuubi Architecture

Introduction

Kyuubi is a high-performance universal JDBC and SQL execution engine. The goal of Kyuubi is to facilitate users to handle big data like ordinary data.

It provides a standardized JDBC interface with easy-to-use data access in big data scenarios. End-users can focus on developing their business systems and mining data value without being aware of the underlying big data platform (compute engines, storage services, metadata management, etc.).

Kyuubi relies on Apache Spark to provide high-performance data query capabilities, and every improvement in the engine's capabilities can help Kyuubi's performance make a qualitative leap. Besides, Kyuubi improves ad-hoc responsiveness through the way of engine caching, and enhances concurrency through horizontal scaling and load balancing.

It provides complete authentication and authentication services to ensure data and metadata security.

It provides robust high availability and load-balancing to help you guarantee the SLA commitment.

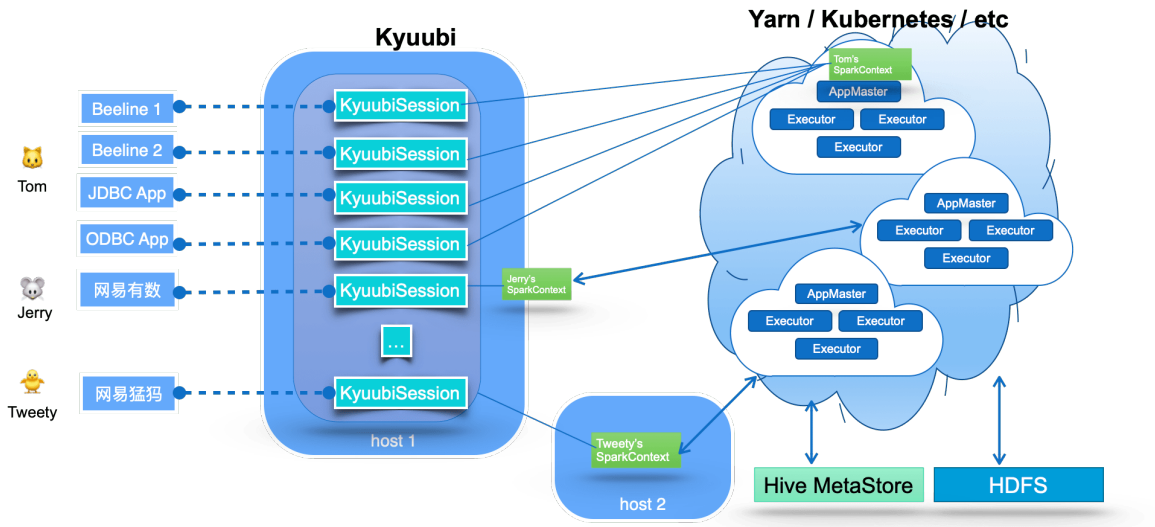
It provides a two-level elastic resource management architecture to effectively improve resource utilization while covering the performance and response requirements of all scenarios, including interactive, or batch processing and point queries or full table scans.

It embraces Spark and builds an ecosystem on top of it, which allows Kyuubi to expand its existing ecosystem and introduce new features quickly, such as cloud-native support and `Data Lake/Lake House` support.

Kyuubi's vision is to build on top of Apache Spark and Data Lake technologies to unify the portal and become an ideal data lake management platform. It can support data processing e.g. ETL, and analytics e.g. BI, in a pure SQL way. All workloads can be done on one platform, using one copy of data, with one SQL interface.

Architecture Overview

The fundamental technical architecture of the Kyuubi system is shown in the following diagram.



The middle part of the diagram shows the Kyuubi server's main component, which handles the clients' connection and execution requests shown in the left part of the image. Within Kyuubi, these connection requests are maintained as the Kyuubi session's, and execution requests are supported as the Kyuubi Operation's which are bound to the corresponding sessions.

The creation of a `Kyuubi Session` can be divided into two cases: lightweight and heavyweight. Most session creations are lightweight and user-unaware. The only heavyweight case is when there is no `SparkContext` instantiated or cached in the user's shared domain, which usually happens when the user is connecting for the first time or has not connected for a long time. This one-time cost session maintenance model can meet most of the ad-hoc fast response requirements.

Kyuubi maintains connections to `SparkConext` in a loosely coupled fashion. These `SparkContexts` can be Spark programs created locally in client deploy mode by this service instance, or in Yarn or Kubernetes clusters in cluster deploy mode. In highly available mode, these `SparkConexts` can also be created by other Kyuubi instances on different machines and shared by this instance.

These `SparkConexts` instances are essentially remote query execution engine programs hosted by Kyuubi services. These programs are implemented on Spark SQL and compile, optimize, and execute SQL statements end-to-end and the necessary interaction with the metadata (e.g. Hive Metastore) and storage (e.g. HDFS) services, maximizing the power of Spark SQL. They can manage their lifecycle, cache and recycle themselves, and are not affected by failover on the Kyuubi server.

Next, let us share some of the key design concepts of Kyuubi.

Unified Interface

Kyuubi implements the Hive Service RPC module, which provides the same way of accessing data as HiveServer2 and Spark Thrift Server. On the client side, you can build fantastic business reports, BI applications, or even ETL jobs only via the Hive JDBC module.

You only need to be familiar with Structured Query Language (SQL) and Java Database Connectivity (JDBC) to handle massive data. It helps you focus on the design and implementation of your business system.

- SQL is the standard language for accessing relational databases and very popular in big data eco too. It turns out that everybody knows SQL.
- JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.
- There are plenty of free or commercial JDBC tools out there.

Runtime Resource Resiliency

The most significant difference between Kyuubi and Spark Thrift Server(STS) is that STS is a single Spark application. For example, if it runs on an Apache Hadoop Yarn cluster, this application is also a single Yarn application that can only exist in a specific fixed queue of the Yarn cluster after it is created. Kyuubi supports the submission of multiple Spark applications.

Yarn loses its role as a resource manager for resource management and does not play the corresponding role of resource isolation and sharing. When users from the client have different resource queue permissions, STS will not be able to handle it in this case.

For data access, a single Spark application has only one user globally, a.k.a. `sparkUser`, and we have to grant it a superuser-like role to allow it to perform data access to different client users, which is a too insecure practice in production environments.

Kyuubi creates different Spark applications based on the connection requests from the client, and these applications can be placed in different shared domains for other connection requests to share.

Kyuubi does not occupy any resources from the Cluster Manager(e.g. Yarn) during startup and will give all resources back if there is not any active session interacting with a `SparkContext`.

Spark also provides Dynamic Resource Allocation to dynamically adjust the resources your application occupies based on the workload. It means that your application may give resources back to the cluster if they are no longer used and request them again later when there is demand. This feature is handy if multiple applications share resources in your Spark cluster.

With these features, Kyuubi provides a two-level elastic resource management architecture to improve resource utilization effectively.

For example,

```
./beeline -u "jdbc:hive2://kyuubi.org:10009/;\
hive.server2.proxy.user=tom#\
spark.yarn.queue=thequeue;\
spark.dynamicAllocation.enabled=true;\
spark.dynamicAllocation.maxExecutors=500;\
spark.shuffle.service.enabled=true;\
spark.executor.cores=3;\
spark.executor.memory=10g"
```

If the user named `tom` opens a connection like above, Kyuubi will try to create a Spark SQL engine application with [3, 500] executors (3 cores, 10g mem each) in the queue named `thequeue` in the Yarn cluster.

On the one hand, because tom enables Spark's dynamic resource request feature, Spark will efficiently request and recycle executors within the program based on the SQL operations scale and the available resources in the queue. On the other hand, when Kyuubi finds that the application has been idle for too long, it will also recycle its application.

High Availability & Load Balance

For an enterprise service, the Service Level Agreement(SLA) commitment must be very high. And the concurrency needs to be sufficiently robust to support the entire enterprise's requests. As a single Spark application and without high availability, Spark Thrift Server can hardly meet the SLA and concurrency requirement. When there are large query requests, there are potential bottlenecks in metadata service access, scheduling and memory pressure of Spark Driver, or the application's overall computational resource constraints.

Kyuubi provides high availability and load balancing solutions based on Zookeeper, as shown in the following diagram.

Let us try to break it down from top to bottom based on the above diagram.

1. At the top of the diagram is the client layer. A client can find multiple registered instances of Kyuubi instance (k.i.) from the namespace in the service discovery layer and then choose to connect. Kyuubi instances registered to the same namespace provide the ability to load balance each other.
2. The selected Kyuubi instance will pick an available engine instance (e.i.) from the engine-namespace in the service discovery layer to establish a connection. If no available instance is found, it will create a new one, wait for the engine to finish registering, and then proceed to connect.
3. If the same person requests a new connection, the connection will be set up to the same or another Kyuubi instance, but the engine instance will be reused.
4. For connections from different users, the step 2 and 3 will be repeated. This is because in the service discovery layer, the namespaces used to store the address of the engine instances are isolated based on the user(by default), and different users cannot access other's instances across the namespace.

Authentication & Authorization

In a secure cluster, services should be able to identify and authenticate callers. As the fact that the user claims does not necessarily mean this is true. The authentication process of Kyuubi is used to verify the user identity that a client used to talk to the Kyuubi server. Once done, a trusted connection will be set up between the client and server if they are successful; otherwise, they will be rejected.

The authenticated client user will also be the user that creates the associate engine instance, then authorizations for database objects or storage could be applied. We also create a Submarine: Spark Security external plugin to achieve fined-grained SQL standard-based authorization.

Conclusions

Kyuubi is a unified multi-tenant JDBC interface for large-scale data processing and analytics, built on top of Apache Spark™. It extends the Spark Thrift Server's scenarios in enterprise applications, the most important of which is multi-tenancy support.

3.9.2 Kyuubi v.s. HiveServer2

Introduction

HiveServer2 is a service that enables clients to execute Hive QL queries on Hive supporting multi-client concurrency and authentication. Kyuubi enables clients to execute Spark SQL queries directly on Spark supporting multi-client concurrency and authentication too.

They are both designed to provide better support for open API clients like JDBC and ODBC to manage and analyze BigData.

Hive on Spark

The purpose of Hive on Spark is to add Spark as a third execution backend, parallel to MR and Tez. Comparing to Hive on MR, it's use the Spark DAG will help improve the performance of Hive queries, especially those have multiple reducer stages.

Differences Between Kyuubi and HiveServer2

-

| Kyuubi | HiveServer2 | |———————–|———————————————————–|———————————————————–| | ** Language ** | Spark SQL | Hive QL | | ** Optimizer ** | Spark SQL Catalyst | Hive Optimizer | | ** Engine ** | up to Spark 3.x | MapReduce/up to Spark 2.3/Tez | | ** Performance ** | High | Low | | ** Compatibility with Spark ** | Good | Bad(need to rebuild on a specific version) | | ** Data Types ** | Spark Data Types | Hive Data Types |

Performance

References

1. HiveServer2 Overview

3.9.3 Kyuubi v.s. Spark Thrift JDBC/ODBC Server (STS)

Introductions

The Apache Spark Thrift JDBC/ODBC Server is a Thrift service implemented by the Apache Spark community based on HiveServer2. Designed to be seamlessly compatible with HiveServer2, it provides Spark SQL capabilities to end-users in a pure SQL way through a JDBC interface. This "out-of-the-box" model minimizes the barriers and costs for users to use Spark.

Kyuubi and Spark are aligned in this goal. On top of that, Kyuubi has made enhancements in multi-tenant support, service availability, service concurrency capability, data security, and other aspects.

Barriers to common Spark job usage

In this part, the most fundamental one is how we define a `Spark User`. Generally speaking, a Spark user is a guy that calls Spark APIs directly, but from Kyuubi and Spark ThriftServer's perspective, the direct API calls occur on the server-side, then a Spark user indirectly interacts with Spark's backend through the more common JDBC specification and protocols. With JDBC and SQL, Kyuubi and Spark ThriftServer make users experience the same way that interacts with most of the world's popular modern DBMSes.

Using Spark APIs directly is flexible for programmers with a bigdata background but may not be friendly for everyone.

High Barrier

Users need a certain programming framework to use Spark through the Scala/Java/Python interfaces provided by Spark. Also, users need to have a good background in big data. For example, users need to know which platform their application will be submitted to, YARN, Kubernetes, or others. They also need to be aware of the resource consumption of their jobs, for example, executor numbers, memory for each executor. If they use too many resources, will it affect other critical tasks? Otherwise, will the cluster's resources be idle and wasted? It is also hard for users to set up thousands of Spark configurations properly. Key features like *Dynamic Resource Allocation*, Speculation might be hard to benefit all with a one-time setup. And new features like *Adaptive Query Execution* could come a long way from the first release involved of Spark to finally get applied to end-users.

Insecurity

Users can access metadata and data by means of code, and data security cannot be guaranteed. All client configurations need to be handed over to the user directly or indirectly. These configurations may contain sensitive information and let all the backend services be completely exposed to the users. For example, in terms of data security, the Submarine Spark Security Plugin provides SQL Standard ACL Management for Apache Spark SQL with Apache Ranger. But in the end, this kind of security feature is at most a "gentleman's agreement" in front of programmers who can write code to submit jobs via Spark code.

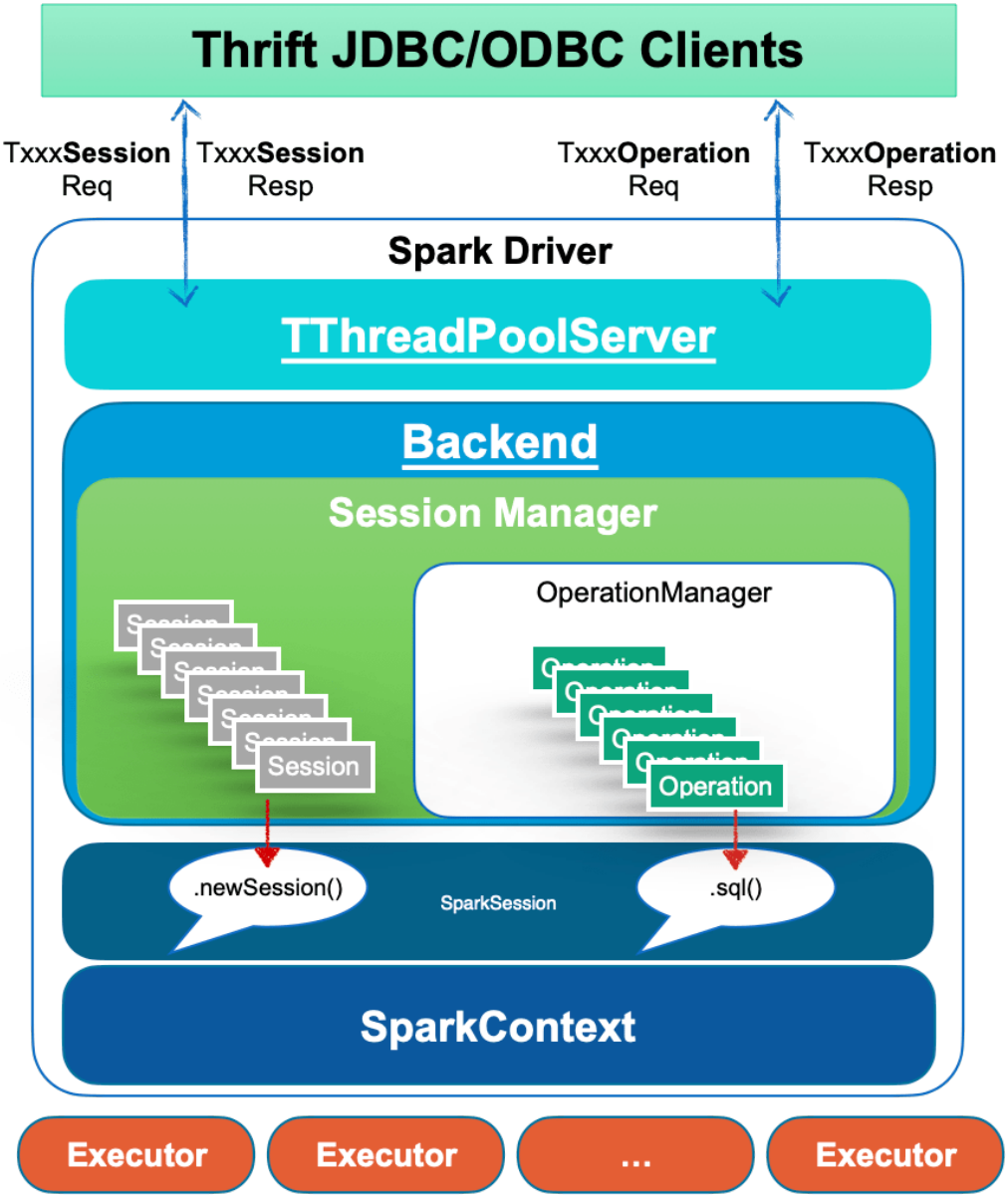
Compatibility

Client-side compatibility is difficult to guarantee. When a user's Spark job is finally scheduled to run on a cluster, it faces problems such as inconsistencies between the client environment and the cluster environment, conflicts between user job dependencies, Spark dependencies, and Hadoop cluster dependencies. When we upgrade the server-side staffs, such as Spark, Hive, and YARN, etc., it is also necessary to upgrade all of the user clients with transitive dependencies as much as possible, which may introduce a lot of unnecessary compatibility testing work, and it is hard to have complete test coverage.

Bootstrap latency

For long-running Spark applications, the bootstrap time is negligible compared to the total lifecycle, such as Spark Structured Streaming. In this case, the task scheduling and computing are fully thread-level with low latency and fast response. For short-term ones, the bootstrap time counts, such as the `SparkPi`. Relatively speaking, this process is very time-consuming, especially for some second-and minute-level computation tasks.

Spark ThriftServer is essentially a Spark application in a multi-threaded scenario. It pre-starts a distributed SQL engine consisting of a driver and multiple executors at runtime. At the SQL parsing layer, the service takes full advantage of the Spark SQL optimizer, and at the computation execution layer, since Spark ThriftServer is resident, there is no bootstrap overhead, and when *DRA* is not enabled, the entire SQL computation process is in pure threaded scheduling model with excellent performance.



The JDBC connections and operations are handled by the frontend thread pool as various requests. And the corresponding methods of the backend are called to bind to the `SparkSession` related interface. For example, the `DriverManager.getConnection` at the client-side will invoke `SparkSession.newSession()` at the server-side, and all queries from the client-side will be submitted to the backend thread pool asynchronously and executed by `SparkSession.sql(...)`.

First, in this mode, users can interact with Spark ThriftServer through simple SQL language and JDBC interface to implement their own business logic. The basic capacity planning of Spark ThriftServer, the consolidation of underlying services, and all the optimizations can all be made on the server-side. Some people may think that only using SQL does not meet all the business, that's true, but the service itself is targeting users that migrating from HiveServer2 for the reason of query speed. With UDF/UDAF support, Some complex logic can still be fulfilled, so basically, Spark ThriftServer is able to deal with most of the big data processing workloads.

Secondly, all the setups for backend services, such as YARN, HDFS, and Hive Metastore Server(HMS), are completed in Spark ThriftServer, so there is no need to hand over the configuration of the backend services to the end-users. This ensures data security to a certain extent. On top of that, the server generally has the ability to do authentication/authorization and other assurance to protect data security.

Finally, the JDBC interface protocol and C/S architecture under the server-side backward compatibility constraints basically ensure that there will be no client-side compatibility obstacles. Users only need to choose the appropriate version of the JDBC driver. The server-side upgrade will not cause interface incompatibility. As for the potential SQL compatibility problem in Spark version upgrade, it also exists when not using Spark ThriftServer, and is more challenging to solve. Moreover, in Spark ThriftServer mode, the server-side can do the full amount of SQL collection in advance, and the verification can be done before the upgrade.

Limitations of Spark ThriftServer

As we can see from the basic architecture of Spark ThriftServer above, it is essentially a single Spark application, and there are generally significant limitations to responding to thousands of client requests.

Driver Bottleneck

The Spark Driver has to both play the role of the scheduler of a Spark application and also the handler of thousands of connections and operations from the client-side. In this case, it is very likely to hit its bottleneck. The Hive metastore client on which the Spark analyzer depends for resolving all queries is one and only, so there will be more obvious concurrency issues when accessing the HMS.

Resource isolation issues

Over-sized Spark jobs encroach on too many of Spark ThriftServer resources, causing other jobs to delay or get stuck.

With Fair Scheduler Pools, Spark ThriftServer has the ability of resource isolation and sharing to a certain extent. It will send queries to a high-weight pool to get more executors for execution. In essence, resource isolation such as CPU/memory/IO should be something that resource managers like YARN and Kubernetes should do. Doing logical isolation at the computing layer is unlikely to work well, and this problem exists in the Apache Impala project as well, for example. And it is difficult to avoid the problem of HMS, HDFS single point access, especially in the scenario of reading and writing dynamic partition tables or handling queries with numerous `Union`s.

Multi-tenancy limitations

Spark ThriftServer itself should be a multi-tenant-enabled system, i.e., it can accept requests from different clients and users. However, from Spark's design point of view, Spark ThriftServer implemented in a single Spark application cannot fully support multi-tenancy because the entire application has only a globally unique username, including both the driver side, and the executor side. So, it has to access all users' data with a single tenant.

Spark ThriftServer occupies a single resource queue (YARN Queue / Kubernetes Namespace), making it difficult to control the resource pool's size available to each tenant in a fine-grained or elastic way from the perspective of resource isolation and sharing. No one would like to restart the server and stop it from serving to adjust some pool's weight or increase the total computing resources.

High Availability Limitations

The community edition of Spark ThriftServer does not support High Availability (HA). It is hard to imagine whether a server-side application without high availability can support the SLA commitment. It's not that difficult to apply an HA implementation to Spark ThriftServer, but tricky. For example, there is already a JIRA ticket: SPARK-11100 with a pull request attached, see [SPARK-11100]. There are generally two ways for the HA implementation of Spark ThriftServer, namely `Active/Standby` and `LoadBalancing`.

The Active/Standby mode consists of active Spark ThriftServer and several standby servers. When the active crashes or hangs, the standby nodes trigger the leader selection to become the new active one to take over. The problems here are undeniable: There is only one active node runtime, so the concurrency capability is limited. When a failover occurs due to hardware and software failure, all current connections and running jobs will fail. This kind of failover is expensive for client-side users. The clients will retry simultaneously, so it's hard for the new elected active server to handle the coming flood of client retries. It's very likely to crash again. The Standby node causes serious waste of cluster resources, whether Spark dynamic resource allocation is enabled or not. A more appropriate approach to solve the server-side single-point problem is to add LoadBalancing support of your own. So that when client requests increase, we can expand Spark ThriftServer horizontally. However, this model also has some limitations. Each Spark ThriftServer is stateful with transient data or functionalities, such as some global temporary views, UDFs, etc., which cannot be shared between two servers. And it's expensive to expand with computing resources together.

UDF Issues

For operations like `ADD JAR ...` or `CREATE TEMPORARY FUNCTION ... USING...`, classes or jars might conflict in the Spark ThriftServer. And there is no such way for deleting when conflicts. Besides, since UDFs are loaded directly into the Spark ThriftServer, if they contain some unintentional or malicious logic, such as calling `System.exit(-1)`, which may kill the service directly, or some operations that affect the server behavior globally like Kerberos authentication.

Kyuubi VS Spark Thrift Server

The HiveServer2 is also introduced here for a more comprehensive comparison.

Consistent Interfaces

Kyuubi, Spark Thrift Server, and HiveServer2 are identical in terms of interfaces and protocols. Therefore, from the user's point of view, the way of use is unchanged. Compared with HiveServer2, the most significant advantage of the first two should be the performance improvement.

From the perspective of SQL syntax compatibility, Kyuubi and Spark Thrift Server are fully compatible with Spark SQL as they are completely delegated to the Spark SQL Catalyst layer. Spark SQL also fully supports Hive QL collections, with only a few enumerable SQL behaviors and syntax differences.

Multi-tenant Architecture

`From wikipedia`: The term "software multitenancy" refers to a software architecture in which a single instance of the software runs on a server and serves multiple tenants. Systems designed in such a manner are often called shared (in contrast to dedicated or isolated).

Kyuubi, Spark ThriftServer, and HiveServer2 have been designed for a typical multi-tenant architecture scenario.

Firstly, we need to consider how to 1) make safer and more efficient use of these compute resources based on resource isolation and 2) how to give users enough control over their own resources.

HiveServer2 is supposed to be the most flexible one. Each SQL is programmed into several Spark applications for execution, and the resource queue, memory, and others can be set before execution. But this approach leads to extremely high Spark bootstrap latency and can not efficiently utilize resources.

Spark ThriftServer goes in the opposite direction because there is only one Spark application. It is impossible to adjust the queue, memory, and other resource-related configs from the user side interface as it is already pre-started. Queries can be sent to pre-set `Fair Scheduler Pools` for running in isolation. The `Fair Scheduler Pools` can only provide low isolation within a Spark application and be configured before Spark ThriftServer starts.

Kyuubi has neutralized these aspects with two other system implementations. Kyuubi applies the multi-tenant feature based on the concept of Kyuubi Engines, where an Engine is a Spark application.

In Kyuubi's system, the Engines are isolated according to tenants. The tenant, a.k.a. user, is unified and end-to-end unique through a JDBC connection. Kyuubi server will identify and authenticate the user and then retrieve or create an Engine belonging to this particular user. This user will be used as the submitter for Engine, and it must have authority to use the resources from YARN, Kubernetes, or just Local machine, e.t.c. Inside an Engine, the Engine's user, a.k.a. `Spark User`, will also be the same. When an Engine runs queries received from the JDBC connection, the Engine's user must also have rights to access the data. Besides, if it needs access to metadata during this process, then we can also add a fine-grained SQL standard ACL management on the metadata layer now with Submarine Spark Security Plugin.

The Engines have their lifecycle, which is related to the `kyuubi.engine.share.level` specified via client configurations. For example, if set to `CONNECTION`, then the corresponding Engine will be created for each JDBC connection and terminates itself when we close the connection. For another example, if set to `USER`, the corresponding Engine is cached and shared with all JDBC connections from the same user, even through different Kyuubi servers in HA mode. The Engine will eventually timeout after all the sessions are closed.

As we need to create Engines, on the one hand, we can configure all the Spark configurations during startup. On the other hand, it does bring the Spark application bootstraps overhead here, but overall, it is just a one-time cost. All queries or connections of the Engine's user will share this application. The more queries it runs, the lower the bootstraps overhead is.

High Availability Capabilities

The HA issues in Spark ThriftServer have already been covered in the previous section so that we won't go over them here again. In Kyuubi, we provide HA in the way of `LoadBlancing`. Kyuubi is lightweight, as it does not create any Engine when it starts. It's cheap to add Kyuubi HA nodes, so horizontal scaling is not overly burdensome.

Client Concurrency

The compilation and optimization for queries in both HiveServer2 and Spark ThriftServer are done on the server-side. In contrast, Kyuubi will do these at the Engine-side. It is instrumental in reducing the workload of the server and improving client concurrency. For task scheduling that belongs to the compute phase also happens at Kyuubi's Engine side. It is not as heavy as the Spark ThriftServer, where there is an intense competition between the client concurrency and the task scheduling. In principle, the more executors there are, or the more significant the amount of data processed, the more pressure on the server-side.

Service Stability

The intense competition between the client concurrency and the task scheduling increases GC issues and OOM risks of Spark ThriftServer. Kyuubi has no problem in this area due to the separation of the server and engines. The UDF risks cannot harm the stability of the service either. As if a user loads and calls an invalid UDF, which only damages its own Engine and will not affect other users or the Kyuubi server.

Summary

Kyuubi extends the use of Spark ThriftServer in a multi-tenant model based on a unified interface and relies on the concept of multi-tenancy to interact with cluster managers to finally gain the ability of resources sharing/isolation and data security. The loosely coupled architecture of Kyuubi Server and Engine greatly improves the concurrency and service stability of the service itself.

3.10 Develop Tools

3.10.1 Building Kyuubi

Building Kyuubi with Apache Maven

Kyuubi is built based on Apache Maven,

```
./build/mvn clean package -DskipTests
```

This results in the creation of all sub-modules of Kyuubi project without running any unit test.

If you want to test it manually, you can start Kyuubi directly from the Kyuubi project root by running

```
bin/kyuubi start
```

Building a Submodule Individually

For instance, you can build the Kyuubi Common module using:

```
build/mvn clean package -pl kyuubi-common -DskipTests
```

Building Submodules Individually

For instance, you can build the Kyuubi Common module using:

```
build/mvn clean package -pl kyuubi-common,kyuubi-ha -DskipTests
```

Skipping Some modules

For instance, you can build the Kyuubi modules without Kyuubi Codecov and Assembly modules using:

```
mvn clean install -pl '!dev/kyuubi-codecov,!kyuubi-assembly' -DskipTests
```

Building Kyuubi against Different Apache Spark versions

Since v1.1.0, Kyuubi support building with different Spark profiles,

Building with Apache dlcdn site

By default, we use `https://archive.apache.org/dist/` to download the built-in release packages of engines, such as Spark or Flink. But sometimes, you may find it hard to reach, or the download speed is too slow, then you can define the `apache.archive.dist` by `-Pmirror-cdn` to accelerate to download speed. For example,

```
build/mvn clean package -Pmirror-cdn
```

The profile migrates your download repo to the Apache officially suggested site - https://dlcdn.apache.org. Note that, this site only holds the latest versions of Apache releases. You may fail if the specific version defined by `spark.version` or `flink.version` is overdue.

Building with the `fast` profile

The `fast` profile helps to significantly reduce build time, which is useful for development or compilation validation, by skipping running the tests, code style checks, building scaladoc, enforcer rules and downloading engine archives used for tests.

```
build/mvn clean package -Pfast
```

3.10.2 Building a Runnable Distribution

To create a Kyuubi distribution like those distributed by Kyuubi Release Page, and that is laid out to be runnable, use `./build/dist` in the project root directory.

For more information on usage, run `./build/dist --help`

```
./build/dist - Tool for making binary distributions of Kyuubi

Usage:
+-----------------------------------------------------------------------------------------
↪--------------+
| ./build/dist [--name <custom_name>] [--tgz] [--flink-provided] [--spark-provided] [--
↪hive-provided]  |
```

(continues on next page)

(continued from previous page)

```
|               [--mvn <maven_executable>] <maven build options>                                                 ␣
↪               |
+-----------------------------------------------------------------------------------------------------------
↪--------------+
name:            -  custom binary name, using project version if undefined
tgz:             -  whether to make a whole bundled package
flink-provided:  -  whether to make a package without Flink binary
spark-provided:  -  whether to make a package without Spark binary
hive-provided:   -  whether to make a package without Hive binary
mvn:             -  external maven executable location
```

For instance,

```
./build/dist --name custom-name --tgz
```

This results in a Kyuubi distribution named `apache-kyuubi-{version}-bin-custom-name.tgz` for you.

If you are planing to deploy Kyuubi where `spark`/`flink`/`hive` is provided, in other word, it's not required to bundle spark/flink/hive binary, use

```
./build/dist --tgz --spark-provided --flink-provided --hive-provided
```

Then you will get a Kyuubi distribution without spark/flink/hive binary named `apache-kyuubi-{version}-bin.tgz`.

3.10.3 Building Kyuubi Documentation

Follow the steps below and learn how to build the Kyuubi documentation as the one you are watching now.

Install & Activate `virtualenv`

Firstly, install `virtualenv`, this is optional but recommended as it is useful to create an independent environment to resolve dependency issues for building the documentation.

```
pip install virtualenv
```

Switch to the `docs` root directory.

```
cd $KYUUBI_SOURCE_PATH/docs
```

Create a virtual environment named 'kyuubi' or anything you like using `virtualenv` if it's not existing.

```
virtualenv kyuubi
```

Activate it,

```
source ./kyuubi/bin/activate
```

Install all dependencies

Install all dependencies enumerated in the `requirements.txt`.

```
pip install -r requirements.txt
```

Create Documentation

Make sure you are in the `$KYUUBI_SOURCE_PATH/docs` directory.

linux & macos

```
make html
```

windows

```
make.bat html
```

If the build process succeed, the HTML pages are in `$KYUUBI_SOURCE_PATH/docs/_build/html`.

View Locally

Open the `$KYUUBI_SOURCE_PATH/docs/_build/html/index.html` file in your favorite web browser.

3.10.4 Running Tests

Kyuubi can be tested based on Apache Maven and the ScalaTest Maven Plugin, please refer to the ScalaTest documentation,

Running Tests Fully

The following is an example of a command to run all the tests:

```
./build/mvn clean install
```

Running Tests for a Module

```
./build/mvn clean install -pl kyuubi-common
```

Running Tests for a Single Test

When developing locally, it's convenient to run one single test, or a couple of tests, rather than all.

With Maven, you can use the -DwildcardSuites flag to run individual Scala tests:

```
./build/mvn clean install -Dtest=none -DwildcardSuites=org.apache.kyuubi.service.
↪FrontendServiceSuite
```

If you want to make a single test that need to integrate with kyuubi-spark-sql-engine module, please build the package for kyuubi-spark-sql-engine module at first.

You can leverage the ready-made tool for creating a binary distribution.

```
./build/dist
```

3.10.5 Debugging Kyuubi

You can use the Java Debug Wire Protocol to debug Kyuubi with your favorite IDE tool, e.g. IntelliJ IDEA.

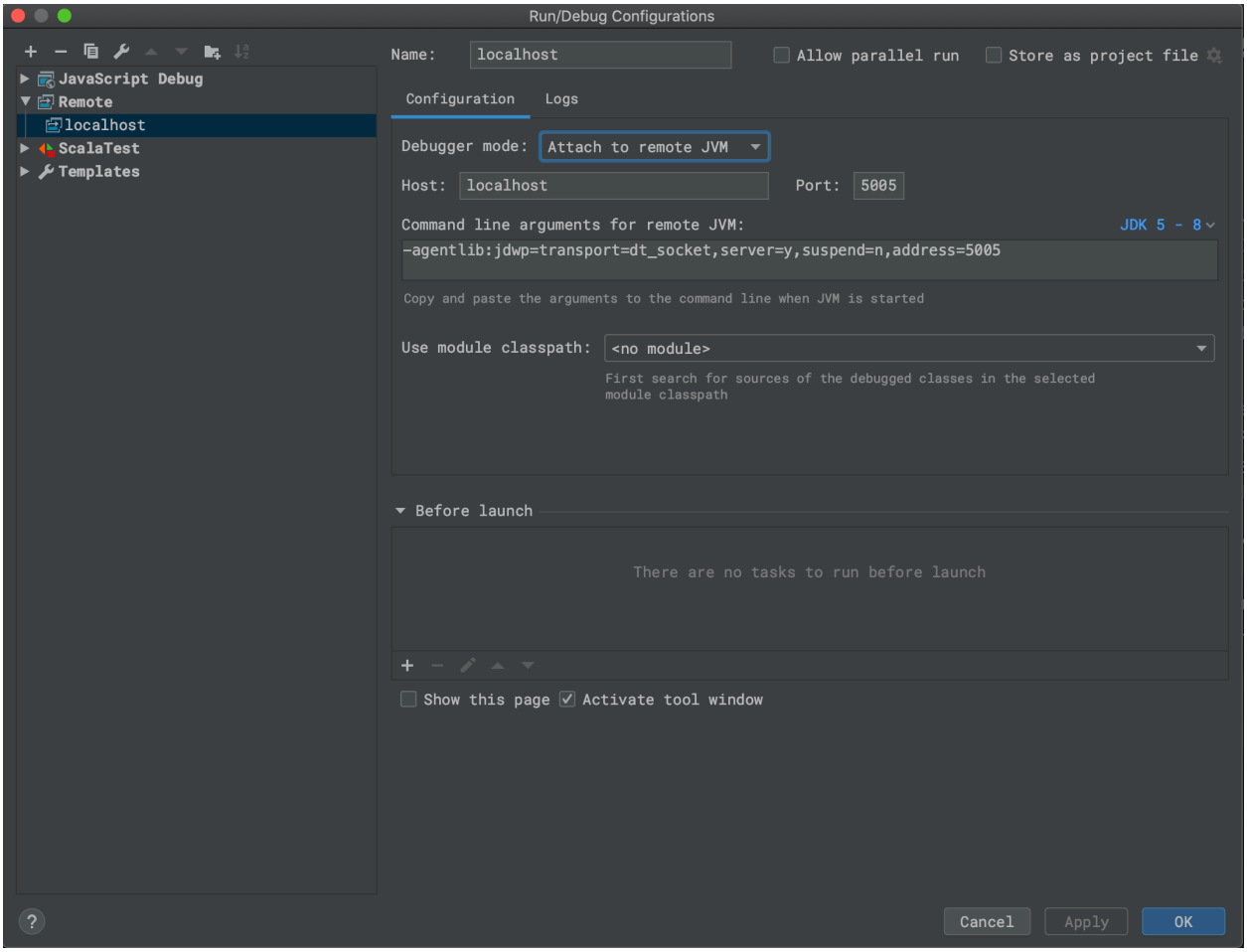
Debugging Server

We can configure the JDWP agent in `KYUUBI_JAVA_OPTS` for debugging.

For example,

```
KYUUBI_JAVA_OPTS=-agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005 \
bin/kyuubi start
```

In the IDE, you set the corresponding parameters(host&port) in debug configurations, for example,



Debugging Engine

We can configure the Kyuubi properties to enable debugging engine.

Flink Engine

```
kyuubi.engine.flink.java.options -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

Trino Engine

```
kyuubi.engine.trino.java.options -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

Hive Engine

```
kyuubi.engine.hive.java.options -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

Debugging Apps

Spark Engine

- Spark Driver

```
spark.driver.extraJavaOptions   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

- Spark Executor

```
spark.executor.extraJavaOptions   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

Flink Engine

- Flink Processes

```
env.java.opts   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005
```

- Flink JobManager

```
env.java.opts.jobmanager   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

- Flink TaskManager

```
env.java.opts.taskmanager   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

- Flink HistoryServer

```
env.java.opts.historyserver   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,
↪address=5005
```

- Flink Client

```
env.java.opts.client   -agentlib:jdwp=transport=dt_socket,server=y,suspend=y,address=5005
```

3.10.6 Developer Tools

Update Project Version

```
build/mvn versions:set -DgenerateBackupPoms=false
```

Update Document Version

Whenever project version updates, please also update the document version at `docs/conf.py` to target the upcoming release.

For example,

```
release = '1.2.0'
```

Update Dependency List

Kyuubi uses the `dev/dependencyList` file to indicate what upstream dependencies will actually go to the server-side classpath.

For Pull requests, a linter for dependency check will be automatically executed in GitHub Actions.

You can run `build/dependency.sh` locally first to detect the potential dependency change first.

If the changes look expected, run `build/dependency.sh --replace` to update `dev/dependencyList` in your Pull request.

Format All Code

Kyuubi uses Spotless with google-java-format and Scalafmt to format the Java and Scala code.

You can run `dev/reformat` to format all Java and Scala code.

Append descriptions of new configurations to settings.md

Kyuubi uses settings.md to explain available configurations.

You can run `KYUUBI_UPDATE=1 build/mvn clean test -pl kyuubi-server -am -Pflink-provided,spark-provided,hive-provided -DwildcardSuites=org.apache.kyuubi.config.AllKyuubiConfiguration` to append descriptions of new configurations to settings.md.

3.10.7 IntelliJ IDEA Setup Guide

Copyright Profile

Every file needs to include the Apache license as a header. This can be automated in IntelliJ by adding a Copyright profile:

1. Go to "Settings/Preferences" → "Editor" → "Copyright" → "Copyright Profiles".
2. Add a new profile and name it "Apache".
3. Add the following text as the copyright text:

```
Licensed to the Apache Software Foundation (ASF) under one
or more contributor license agreements.  See the NOTICE file
distributed with this work for additional information
regarding copyright ownership.  The ASF licenses this file
to you under the Apache License, Version 2.0 (the
"License"); you may not use this file except in compliance
with the License.  You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

4. Go to "Editor" → "Copyright" and choose the "Apache" profile as the default profile for this project.
5. Click "Apply".

Required Plugins

Go to "Settings/Preferences" → "Plugins" and select the "Marketplace" tab. Search for the following plugins, install them, and restart the IDE if prompted:

- Scala

You will also need to install the google-java-format plugin. However, a specific version of this plugin is required. Download google-java-format v1.7.0.6 and install it as follows. Make sure to NEVER update this plugin.

1. Go to "Settings/Preferences" → "Plugins".
2. Click the gear icon and select "Install Plugin from Disk".
3. Navigate to the downloaded ZIP file and select it.

Formatter For Java

Kyuubi uses Spotless together with google-java-format to format the Java code.

It is recommended to automatically format your code by applying the following settings:

1. Go to "Settings/Preferences" → "Other Settings" → "google-java-format Settings".
2. Tick the checkbox to enable the plugin.
3. Change the code style to "Default Google Java style".
4. Go to "Settings/Preferences" → "Tools" → "Actions on Save".
5. Select "Reformat code".

If you use the IDEA version is 2021.1 and below, please replace the above steps 4 and 5 by using the Save Actions plugin.

Formatter For Scala

Enable Scalafmt as follows:

1. Go to "Settings/Preferences" → "Editor" → "Code Style" → "Scala"
2. Set "Formatter" to "Scalafmt"
3. Enable "Reformat on file save"

Checkstyle For Scala

Enable Scalastyle as follows:

1. Go to "Settings/Preferences" → "Editor" → "Inspections".
2. Search for "Scala style inspection" and enable it.

3.11 Community

3.11.1 Contributing to Apache Kyuubi

Thanks for your interest in the Apache Kyuubi project. Contributions are welcome and are greatly appreciated! Every little effort helps, and a credit will always be given.

This page provides some orientation and resources we have for you to get involved. It also offers recommendations on getting the best results when engaging with the community. We hope that this will be a pleasant first experience for you to return to continue contributing.

Get Involved

In the process of using Apache Kyuubi, if you have any questions, suggestions, or improvement ideas, you can participate in the Kyuubi community building through the following suggested channels.

- Join the Mailing Lists - the best way to keep up-to-date with the community.
- Issue Tracker - tracking bugs, ideas, plans, etc.
- GitHub Discussions - second to mailing list for anything else you want to share or ask
- Slack - chat with our community User && Developer anytime!

Contributing Guide

As a community-driven project. All bits of help are welcome.

Contributing code is excellent, but that's probably not the first place to start. There are many ways to make valuable contributions to the project and community.

You can make various types of contributions to Kyuubi, including the following but not limited to,

- Answer questions in the Mailing Lists
- Share your success stories with us
- Improve Documentation -
- Test latest releases -
- Improve test coverage -
- Report bugs and better help developers to reproduce
- Review changes
- Make a pull request
- Promote to others
- Click the star button if you like this project

Easter Eggs for Contributors

TBD, please be patient for the surprise.

IDE Setup Guide

IntelliJ IDEA Setup Guide

3.11.2 Collaborators

PMC Members and Committers

See full contributor list at contributors.

3.11.3 Kyuubi Release Guide

Introduction

The Apache Kyuubi project periodically declares and publishes releases. A release is one or more packages of the project artifact(s) that are approved for general public distribution and use. They may come with various degrees of caveat regarding their perceived quality and potential for change, such as "alpha", "beta", "incubating", "stable", etc.

The Kyuubi community treats releases with great importance. They are a public face of the project and most users interact with the project only through the releases. Releases are signed off by the entire Kyuubi community in a public vote.

Each release is executed by a Release Manager, who is selected among the Kyuubi committers. This document describes the process that the Release Manager follows to perform a release. Any changes to this process should be discussed and adopted on the dev mailing list.

Please remember that publishing software has legal consequences. This guide complements the foundation-wide Product Release Policy and Release Distribution Policy.

Overview

The release process consists of several steps:

1. Decide to release
2. Prepare for the release
3. Cut branch off for **feature** release
4. Build a release candidate
5. Vote on the release candidate
6. If necessary, fix any issues and go back to step 3.
7. Finalize the release
8. Promote the release
9. Remove the dist repo directories for deprecated release candidates
10. Publish docker image

Decide to release

Deciding to release and selecting a Release Manager is the first step of the release process. This is a consensus-based decision of the entire community.

Anybody can propose a release on the dev mailing list, giving a solid argument and nominating a committer as the Release Manager (including themselves). There's no formal process, no vote requirements, and no timing requirements. Any objections should be resolved by consensus before starting the release.

In general, the community prefers to have a rotating set of 1-2 Release Managers. Keeping a small core set of managers allows enough people to build expertise in this area and improve processes over time, without Release Managers needing to re-learn the processes for each release. That said, if you are a committer interested in serving the community in this way, please reach out to the community on the dev mailing list.

Checklist to proceed to the next step

1. Community agrees to release
2. Community selects a Release Manager

Prepare for the release

Before your first release, you should perform one-time configuration steps. This will set up your security keys for signing the release and access to various release repositories.

One-time setup instructions

ASF authentication

The environments `ASF_USERNAME` and `ASF_PASSWORD` have been used in several places and several times in the release process, you can either one-time set up them in `~/.bashrc` or `~/.zshrc`, or export them in terminal every time.

```
export ASF_USERNAME=<your apache username>
export ASF_PASSWORD=<your apache password>
```

Java Home

An available environment variable `JAVA_HOME`, you can do `echo $JAVA_HOME` to check it. Note that, the Java version should be 8.

Subversion

Besides on `git`, `svn` is also required for Apache release, please refer to https://www.apache.org/dev/version-control.html#https-svn for details.

GPG Key

You need to have a GPG key to sign the release artifacts. Please be aware of the ASF-wide release signing guidelines. If you don't have a GPG key associated with your Apache account, please create one according to the guidelines.

Determine your Apache GPG Key and Key ID, as follows:

```
gpg --list-keys --keyid-format SHORT
```

This will list your GPG keys. One of these should reflect your Apache account, for example:

```
pub   rsa4096 2021-08-30 [SC]
      8FC8075E1FDC303276C676EE8001952629BCC75D
uid           [ultimate] Cheng Pan <chengpan@apache.org>
sub   rsa4096 2021-08-30 [E]
```

> Note: To follow the Apache's release specification, all new RSA keys generated should be at least 4096 bits. Do not generate new DSA keys.

Here, the key ID is the 8-digit hex string in the pub line: `29BCC75D`.

To export the PGP public key, using:

```
gpg --armor --export 29BCC75D
```

If you have more than one gpg key, you can specify the default key as the following:

```
echo 'default-key <key-fpr>' > ~/.gnupg/gpg.conf
```

The last step is to update the KEYS file with your code signing key https://www.apache.org/dev/openpgp.html#export-public-key

```
svn checkout --depth=files "https://dist.apache.org/repos/dist/release/kyuubi" work/svn-
↪kyuubi

(gpg --list-sigs "${ASF_USERNAME}@apache.org" && gpg --export --armor "${ASF_USERNAME}
↪@apache.org") >> work/svn-kyuubi/KEYS

svn commit --username "${ASF_USERNAME}" --password "${ASF_PASSWORD}" --message "Update␣
↪KEYS" work/svn-kyuubi
```

In order to make yourself have the right permission to stage java artifacts in Apache Nexus staging repository, please submit your GPG public key to ubuntu server via

```
gpg --keyserver hkp://keyserver.ubuntu.com --send-keys ${PUBLIC_KEY} # send public key␣
↪to ubuntu server
gpg --keyserver hkp://keyserver.ubuntu.com --recv-keys ${PUBLIC_KEY} # verify
```

Cut branch if for feature release

Kyuubi use version pattern `{MAJOR_VERSION}.{MINOR_VERSION}.{PATCH_VERSION}[-{OPTIONAL_SUFFIX}]`, e.g. `1.7.0`. **Feature Release** means `MAJOR_VERSION` or `MINOR_VERSION` changed, and **Patch Release** means `PATCH_VERSION` changed.

The main step towards preparing a feature release is to create a release branch. This is done via standard Git branching mechanism and should be announced to the community once the branch is created.

> Note: If you are releasing a patch version, you can ignore this step.

The release branch pattern is `branch-{MAJOR_VERSION}.{MINOR_VERSION}`, e.g. `branch-1.7`.

After cutting release branch, don't forget bump version in `master` branch.

Build a release candidate

> Don't forget to switch to the release branch!

- Set environment variables.

```
export RELEASE_VERSION=<release version, e.g. 1.7.0>
export RELEASE_RC_NO=<RC number, e.g. 0>
export NEXT_VERSION=<e.g. 1.7.1>
```

- Bump version, and create a git tag for the release candidate.

Considering that other committers may merge PRs during your release period, you should accomplish the version change first, and then come back to the release candidate tag to continue the rest release process.

The tag pattern is `v${RELEASE_VERSION}-rc${RELEASE_RC_NO}`, e.g. `v1.7.0-rc0`

> NOTE: After all the voting passed, be sure to create a final tag with the pattern: `v${RELEASE_VERSION}`

```
# Bump to the release version
build/mvn versions:set -DgenerateBackupPoms=false -DnewVersion="${RELEASE_VERSION}"
git commit -am "[RELEASE] Bump ${RELEASE_VERSION}"

# Create tag
git tag v${RELEASE_VERSION}-rc${RELEASE_RC_NO}

# Prepare for the next development version
build/mvn versions:set -DgenerateBackupPoms=false -DnewVersion="${NEXT_VERSION}-SNAPSHOT"
git commit -am "[RELEASE] Bump ${NEXT_VERSION}-SNAPSHOT"

# Push branch to apache remote repo
git push apache

# Push tag to apache remote repo
git push apache v${RELEASE_VERSION}-rc${RELEASE_RC_NO}

# Go back to release candidate tag
git checkout v${RELEASE_VERSION}-rc${RELEASE_RC_NO}
```

- Package source and binary artifacts, and upload them to the Apache staging SVN repo. Publish jars to the Apache staging Maven repo.

```
build/release/release.sh publish
```

To make your release available in the staging repository, you must close the staging repo in the Apache Nexus. Until you close, you can re-run deploying to staging multiple times. But once closed, it will create a new staging repo. So ensure you close this, so that the next RC (if need be) is on a new repo. Once everything is good, close the staging repository on Apache Nexus.

- Generate a pre-release note from GitHub for the subsequent voting.

Goto the release page and click the "Draft a new release" button, then it would jump to a new page to prepare the release.

Filling in all the necessary information required by the form. And in the bottom of the form, choose the "This is a pre-release" checkbox. Finally, click the "Publish release" button to finish the step.

> Note: the pre-release note is used for voting purposes. It would be marked with a **Pre-release** tag. After all the voting works(dev and general) are finished, do not forget to inverse the "This is a pre-release" checkbox. The pre-release version comes from vx.y.z-rcN tags, and the final version should come from vx.y.z tags.

Vote on the release candidate

The release voting takes place on the Apache Kyuubi developers list.

- If possible, attach a draft of the release notes with the email.
- Recommend represent voting closing time in UTC format.
- Make sure the email is in text format and the links are correct.

  Note: you can generate the voting mail content for dev ML automatically via invoke the `build/release/script/dev_kyuubi_vote.sh` script.

Once the vote is done, you should also send out a summary email with the totals, with a subject that looks something like **[VOTE][RESULT] Release Apache Kyuubi ...**

Finalize the Release

Be Careful!

THIS STEP IS IRREVERSIBLE so make sure you selected the correct staging repository. Once you move the artifacts into the release folder, they cannot be removed.

After the vote passes, to upload the binaries to Apache mirrors, you move the binaries from dev directory (this should be where they are voted) to release directory. This "moving" is the only way you can add stuff to the actual release directory. (Note: only PMC members can move to release directory)

Move the subdirectory in "dev" to the corresponding directory in "release". If you've added your signing key to the KEYS file, also update the release copy.

```
build/release/release.sh finalize
```

Verify that the resources are present in https://www.apache.org/dist/kyuubi/. It may take a while for them to be visible. This will be mirrored throughout the Apache network.

For Maven Central Repository, you can Release from the Apache Nexus Repository Manager. Log in, open "Staging Repositories", find the one voted on, select and click "Release" and confirm. If successful, it should show up under https://repository.apache.org/content/repositories/releases/org/apache/kyuubi/ and the same under https://repository.apache.org/content/groups/maven-staging-group/org/apache/kyuubi/ (look for the correct release version). After some time this will be synced to Maven Central automatically.

Promote the release

Update Website

Fork and clone Apache Kyuubi website

1. Add a new markdown file in `src/zh/news/`, `src/en/news/`
2. Add a new markdown file in `src/zh/release/`, `src/en/release/`
3. Follow *Build Document* to build documents, then copy `apache/kyuubi`'s folder `docs/_build/html` to `apache/kyuubi-website`'s folder `content/docs/r{RELEASE_VERSION}`

Create an Announcement

Once everything is working, create an announcement on the website and then send an e-mail to the mailing list. You can generate the announcement via `build/release/script/announce.sh` automatically. The mailing list includes: `announce@apache.org`, `dev@kyuubi.apache.org`, `user@spark.apache.org`.

Note that, you must use the apache.org email to send announce to `announce@apache.org`.

Enjoy an adult beverage of your choice, and congratulations on making a Kyuubi release.

Remove the dist repo directories for deprecated release candidates

Remove the deprecated dist repo directories at last.

```
cd work/svn-dev
svn delete https://dist.apache.org/repos/dist/dev/kyuubi/{RELEASE_TAG} \
  --username "${ASF_USERNAME}" \
  --password "${ASF_PASSWORD}" \
  --message "Remove deprecated Apache Kyuubi ${RELEASE_TAG}"
```

Publish docker image

See steps in `https://github.com/apache/kyuubi-docker/blob/master/release/release_guide.md`

3.12 Appendixes

3.12.1 Terminologies

Kyuubi

Kyuubi is a unified multi-tenant JDBC interface for large-scale data processing and analytics, built on top of Apache Spark.

JDBC

> The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.
>
> JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.

Typically, there is a gap between business development and big data analytics. If the two are forcefully coupled, it would make the corresponding system difficult to operate and optimize. On the flip side, if decoupled, the values of both can be maximized. Business experts can stay focused on their own business development, while Big Data engineers can continuously optimize server-side performance and stability. Kyuubi combines the two seamlessly through an easy-to-use JDBC interface.

Apache Hive

> The Apache Hive ™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.

Kyuubi supports Hive JDBC driver, which helps you seamlessly migrate your slow queries from Hive to Spark SQL.

Apache Thrift

> The Apache Thrift software framework, for scalable cross-language services development, combines a software stack with a code generation engine to build services that work efficiently and seamlessly between C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages.

Server

Server is a daemon process that handles concurrent connection and query requests and converting these requests into various operations against the **query engines** to complete the responses to clients.

Aliases: Kyuubi Server / Kyuubi Instance / k.i.

ServerSpace

A ServerSpace is used to register servers and expose them together as a service layer to clients.

Engine

An engine handles all queries through Kyuubi servers. It is created in one Kyuubi server and can be shared with other Kyuubi servers by registering itself to an engine namespace. All its capabilities are mainly powered by Spark SQL.

Aliases: Query Engine / Engine Instance / e.i.

EngineSpace

An EngineSpace is internally used by servers to register and interact with engines.

Apache Spark

> Apache Spark™ is a unified analytics engine for large-scale data processing.

Multi Tenancy

Kyuubi guarantees end-to-end multi-tenant isolation and sharing in the following pipeline

```
Client --> Kyuubi --> Query Engine(Spark) --> Resource Manager --> Data Storage Layer
```

High Availability / Load Balance

As an enterprise service, SLA commitment is essential. Deploying Kyuubi in High Availability (HA) mode helps you guarantee that.

Apache Zookeeper

> Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination.

Apache Curator

> Apache Curator is a Java/JVM client library for Apache ZooKeeper, a distributed coordination service. It includes a high-level API framework and utilities to make using Apache ZooKeeper much easier and more reliable. It also includes recipes for common use cases and extensions such as service discovery and a Java 8 asynchronous DSL.

DataLake & LakeHouse

Kyuubi unifies DataLake & LakeHouse access in the simplest pure SQL way, meanwhile it's also the securest way with authentication and SQL standard authorization.

Apache Iceberg

> Apache Iceberg is an open table format for huge analytic datasets. Iceberg adds tables to Trino and Spark that use a high-performance format that works just like a SQL table.

Delta Lake

> Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark™ and big data workloads.

Apache Hudi

> Apache Hudi ingests & manages storage of large analytical datasets over DFS (hdfs or cloud stores).