Kyuubi

Release 1.3.0

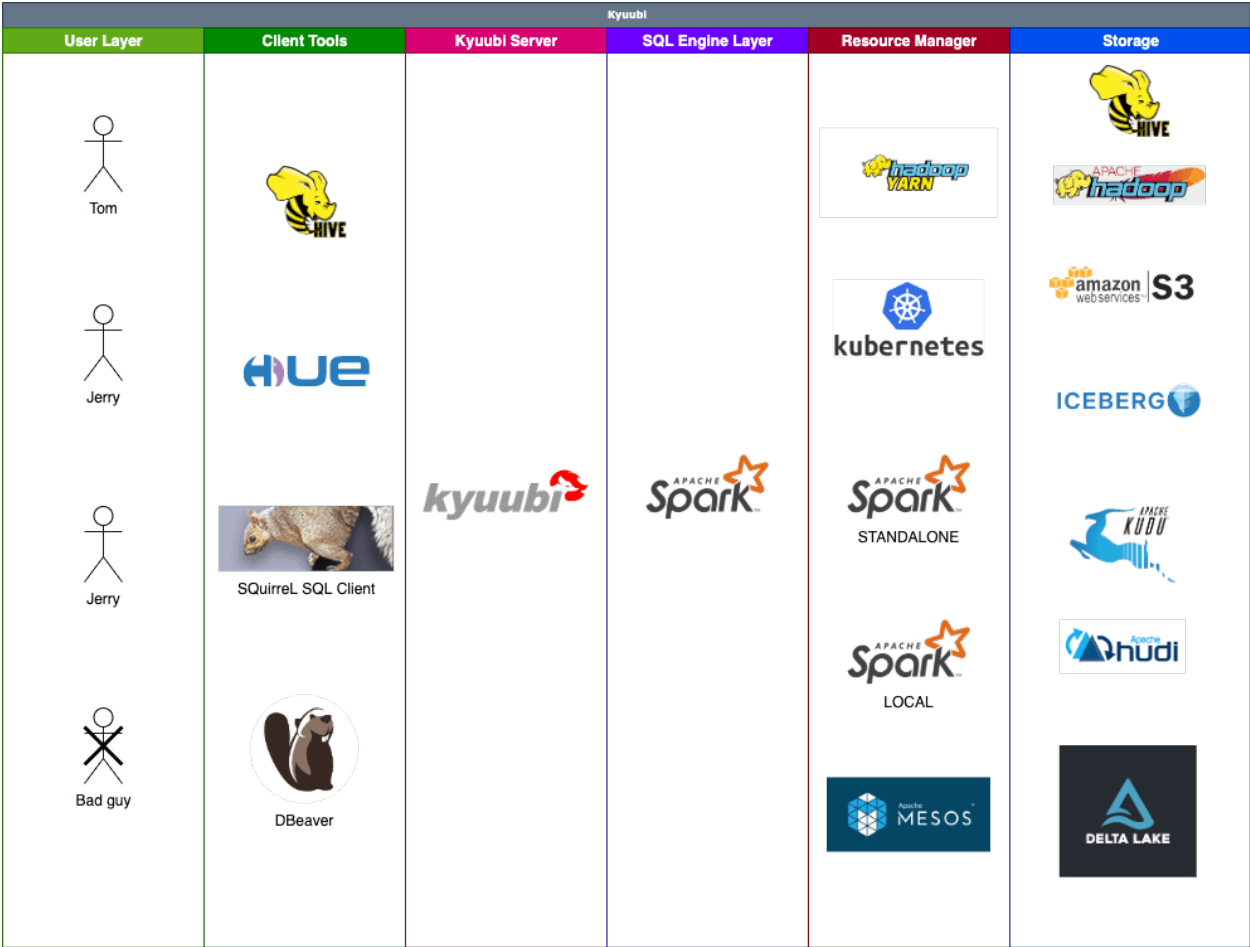**Kent Yao**

Nov 23, 2021

USAGE GUIDE

Kyuubi™ is a unified multi-tenant JDBC interface for large-scale data processing and analytics, built on top of Apache Spark™.

In general, the complete ecosystem of Kyuubi falls into the hierarchies shown in the above figure, with each layer loosely coupled to the other.

For example, you can use Kyuubi, Spark and Apache Iceberg to build and manage Data Lake with pure SQL for both data processing e.g. ETL, and analytics e.g. BI. All workloads can be done on one platform, using one copy of data, with one SQL interface.

Kyuubi provides the following features:

MULTI-TENANCY

Kyuubi supports the end-to-end multi-tenancy, and this is why we want to create this project despite that the Spark Thrift JDBC/ODBC server already exists.

1. Supports multi-client concurrency and authentication

2. Supports one Spark application per account(SPA).

3. Supports QUEUE/NAMESPACE Access Control Lists (ACL)

4. Supports metadata & data Access Control Lists

Users who have valid accounts could use all kinds of client tools, e.g. Hive Beeline, HUE, DBeaver, SQuirreL SQL Client, etc, to operate with Kyuubi server concurrently.

The SPA policy makes sure 1) a user account can only get computing resource with managed ACLs, e.g. Queue Access Control Lists, from cluster managers, e.g. Apache Hadoop YARN, Kubernetes (K8s) to create the Spark application; 2) a user account can only access data and metadata from a storage system, e.g. Apache Hadoop HDFS, with permissions.

EASE OF USE

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.

RUN ANYWHERE

Kyuubi can submit Spark applications to all supported cluster managers, including YARN, Mesos, Kubernetes, Standalone, and local.

The SPA policy also make it possible for you to launch different applications against different cluster managers.

FOUR

HIGH PERFORMANCE

Kyuubi is built on the Apache Spark, a lightning-fast unified analytics engine.

- **Concurrent execution**: multiple Spark applications work together
- **Quick response**: long-running Spark applications without startup cost
- **Optimal execution plan**: fully supports Spark SQL Catalyst Optimizer,

FIVE

AUTHENTICATION & AUTHORIZATION

With strong authentication and fine-grained column/row level authorization, Kyuubi keeps your system and data secure.

HIGH AVAILABILITY

Kyuubi provides both high availability and load balancing solutions based on Zookeeper.

6.1 Quick Start

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

6.1.1 Getting Started with Apache Kyuubi

Getting Kyuubi

Currently, Apache Kyuubi maintains all its releases on our official website. You can get the most recent stable release of Apache Kyuubi here:

Download

Requirements

These are essential components required for Kyuubi to startup. For quick start deployment, the only thing you need is `JAVA_HOME` and `SPARK_HOME` being correctly set. The Kyuubi release package you downloaded or built contains the rest prerequisites inside already.

Additionally, if you want to work with other Spark compatible systems or plugins, you only need to take care of them as using them with regular Spark applications. For example, you can run Spark SQL engines created by the Kyuubi on any cluster manager, including YARN, Kubernetes, Mesos, e.t.c… Or, you can manipulate data from different data sources with the Spark Datasource API, e.g. Delta Lake, Apache Hudi, Apache Iceberg, Apache Kudu and e.t.c…

Installation

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

```
tar zxf apache-kyuubi-1.3.1-incubating-bin.tgz
```

This will result in the creation of a subdirectory named `apache-kyuubi-1.3.1-incubating-bin` shown below,

```
apache-kyuubi-1.3.1-incubating-bin
├── DISCLAIMER
├── LICENSE
├── NOTICE
├── RELEASE
├── bin
├── conf
│   ├── kyuubi-defaults.conf.template
│   ├── kyuubi-env.sh.template
│   └── log4j.properties.template
├── docker
│   ├── Dockerfile
│   ├── kyuubi-configmap.yaml
│   ├── kyuubi-pod.yaml
│   └── kyuubi-service.yaml
├── extension
│   └── kyuubi-extension-spark-3-1_2.12-1.3.1-incubating.jar
├── externals
│   └── engines
├── jars
├── licenses
├── logs
├── pid
└── work
```

From top to bottom are:

- DISCLAIMER: the disclaimer made by Apache Kyuubi Community as a project still in ASF Incubator.

- 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 Spark applications.

- externals

- – engines: contains all kinds of SQL engines that we support, e.g. Apache Spark, Apache Flink(coming soon).

- 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.

Running Kyuubi

As mentioned above, for a quick start deployment, then only you need to be sure is that your java runtime environment and `SPARK_HOME` are correct.

Setup JAVA

You can either set it system-widely, e.g. in the `.bashrc` file.

```
java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)
```

Or, `export JAVA_HOME=/path/to/java` in the local os session.

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-11.0.5.jdk/Contents/Home
java -version
java version "11.0.5" 2019-10-15 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.5+10-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.5+10-LTS, mixed mode)
```

The recommended place to set `JAVA_HOME` is `$KYUUBI_HOME/conf/kyuubi-env.sh`, as the ways above are too flaky. The `JAVA_HOME` in `$KYUUBI_HOME/conf/kyuubi-env.sh` will take others' precedence.

Setup Spark

Similar to `JAVA_HOME`, you can also set `SPARK_HOME` in different ways. However, we recommend setting it in `$KYUUBI_HOME/conf/kyuubi-env.sh` too.

For example,

```
SPARK_HOME=~/Downloads/spark-3.2.0-bin-hadoop3.2
```

Starting Kyuubi

```
bin/kyuubi start
```

It will print all essential environment variables on the screen during the server starts, and you may check whether they are expected.

```
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:
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\ \ \ \ \
    \ \_\ \_\/`____ \ \_____/\ \_____/ \ \_,__/\ \_\
     \/_/\/_/`/___/> \/___/   \/___/    \/___/  \/_/
                /\___/
                \/__/
```

If all goes well, this will result in the creation of the Kyuubi server instance with a `PID` stored in `$KYUUBI_HOME/pid/kyuubi-<username>-org.apache.kyuubi.server.KyuubiServer.pid`

Then, you can get the JDBC connection URL at the end of the log file, e.g.

```
ThriftFrontendService: 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.

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
```

Using Hive Beeline

Kyuubi server is compatible with Apache Hive beeline, so you can use `$SPARK_HOME/bin/beeline` for testing.

Opening a Connection

The command below will tell the Kyuubi server to create a session with itself.

```
bin/beeline -u 'jdbc:hive2://localhost:10009/'
Connecting to jdbc:hive2://localhost:10009/
Connected to: Spark SQL (version 1.0.2)
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/>
```

In this case, the session will create for the user named 'anonymous'.

Kyuubi will create a Spark SQL engine application using `kyuubi-spark-sql-engine_2.12-<version>.jar`. It will cost awhile for the application to be ready before fully establishing the session. Otherwise, an existing application will be reused, and the time cost here is negligible.

Similarly, you can create a session for another user(or principal, subject, and maybe something else you defined), e.g. named `kentyao`,

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

The formerly created Spark application for user 'anonymous' will not be reused in this case, while a brand new application will be submitted for user 'kentyao' instead.

Then, you can see 3 processes running in your local environment, including one `KyuubiServer` instance and 2 `SparkSubmit` instances as the SQL engines.

```
75730 Jps
70843 KyuubiServer
72566 SparkSubmit
75356 SparkSubmit
```

Execute Statements

If the beeline session is successfully connected, then you can run any query supported by Spark SQL now. For example,

```
0: jdbc:hive2://10.242.189.214:2181/> select timestamp '2018-11-17';
2021-10-28 13:56:27.509 INFO operation.ExecuteStatement: Processing kent's
↪query[1f619182-20ad-4733-995b-a5e43b80d998]: INITIALIZED_STATE -> PENDING_STATE,
↪statement: select timestamp '2018-11-17'
2021-10-28 13:56:27.547 INFO operation.ExecuteStatement: Processing kent's
↪query[1f619182-20ad-4733-995b-a5e43b80d998]: PENDING_STATE -> RUNNING_STATE,
↪statement: select timestamp '2018-11-17'
2021-10-28 13:56:27.540 INFO operation.ExecuteStatement: Processing kent's
↪query[a46ca504-fe3a-4dfb-be1e-19770af8ac4c]: INITIALIZED_STATE -> PENDING_STATE,
↪statement: select timestamp '2018-11-17'
2021-10-28 13:56:27.541 INFO operation.ExecuteStatement: Processing kent's
↪query[a46ca504-fe3a-4dfb-be1e-19770af8ac4c]: PENDING_STATE -> RUNNING_STATE,
↪statement: select timestamp '2018-11-17'
```

(continues on next page)

```
2021-10-28 13:56:27.543 INFO operation.ExecuteStatement:
          Spark application name: kyuubi_USER_kent_7ad055d0-3eca-4b78-87e8-
→94b22f3bade9
                application ID: local-1635400506190
                application web UI: http://10.242.189.214:56774
                master: local[*]
                deploy mode: client
                version: 3.2.0
          Start time: 2021-10-28T13:55:05.528
          User: kent
2021-10-28 13:56:27.604 INFO operation.ExecuteStatement: Processing kent's␣
→query[a46ca504-fe3a-4dfb-be1e-19770af8ac4c]: RUNNING_STATE -> RUNNING_STATE,␣
→statement: select timestamp '2018-11-17'
2021-10-28 13:56:27.627 INFO codegen.CodeGenerator: Code generated in 6.696179 ms
2021-10-28 13:56:27.635 INFO spark.SparkContext: Starting job: collect at␣
→ExecuteStatement.scala:97
2021-10-28 13:56:27.639 INFO kyuubi.SQLOperationListener: Query [a46ca504-fe3a-4dfb-
→be1e-19770af8ac4c]: Job 3 started with 1 stages, 1 active jobs running
2021-10-28 13:56:27.639 INFO kyuubi.SQLOperationListener: Query [a46ca504-fe3a-4dfb-
→be1e-19770af8ac4c]: Stage 3 started with 1 tasks, 1 active stages running
2021-10-28 13:56:27.651 INFO scheduler.DAGScheduler: Job 3 finished: collect at␣
→ExecuteStatement.scala:97, took 0.016234 s
2021-10-28 13:56:27.653 INFO kyuubi.SQLOperationListener: Finished stage: Stage(3, 0);
→ Name: 'collect at ExecuteStatement.scala:97'; Status: succeeded; numTasks: 1;␣
→Took: 13 msec
2021-10-28 13:56:27.663 INFO scheduler.StatsReportListener: task runtime:(count: 1,␣
→mean: 8.000000, stdev: 0.000000, max: 8.000000, min: 8.000000)
2021-10-28 13:56:27.664 INFO scheduler.StatsReportListener:        0%          5␣
→%        10%       25%       50%       75%       90%       95%       100%
2021-10-28 13:56:27.664 INFO scheduler.StatsReportListener:       8.0 ms       8.0␣
→ms       8.0 ms       8.0 ms       8.0 ms       8.0 ms       8.0␣
→ms       8.0 ms
2021-10-28 13:56:27.665 INFO scheduler.StatsReportListener: shuffle bytes␣
→written:(count: 1, mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-28 13:56:27.665 INFO scheduler.StatsReportListener:        0%          5␣
→%        10%       25%       50%       75%       90%       95%       100%
2021-10-28 13:56:27.665 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
2021-10-28 13:56:27.666 INFO scheduler.StatsReportListener: fetch wait time:(count: 1,
→ mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-28 13:56:27.666 INFO scheduler.StatsReportListener:        0%          5␣
→%        10%       25%       50%       75%       90%       95%       100%
2021-10-28 13:56:27.666 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
2021-10-28 13:56:27.667 INFO scheduler.StatsReportListener: remote bytes read:(count:␣
→1, mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-28 13:56:27.667 INFO scheduler.StatsReportListener:        0%          5␣
→%        10%       25%       50%       75%       90%       95%       100%
2021-10-28 13:56:27.667 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
2021-10-28 13:56:27.668 INFO scheduler.StatsReportListener: task result size:(count:␣
→1, mean: 1402.000000, stdev: 0.000000, max: 1402.000000, min: 1402.000000)
2021-10-28 13:56:27.668 INFO scheduler.StatsReportListener:        0%          5␣
→%        10%       25%       50%       75%       90%       95%       100%
```

```
2021-10-28 13:56:27.669 INFO scheduler.StatsReportListener:          1402.0␣
↪B          1402.0 B          1402.0 B          1402.0 B          1402.0 B          1402.0␣
↪B          1402.0 B          1402.0 B          1402.0 B
2021-10-28 13:56:27.669 INFO codegen.CodeGenerator: Code generated in 8.815996 ms
2021-10-28 13:56:27.672 INFO scheduler.StatsReportListener: executor (non-fetch) time␣
↪pct: (count: 1, mean: 12.500000, stdev: 0.000000, max: 12.500000, min: 12.500000)
2021-10-28 13:56:27.672 INFO scheduler.StatsReportListener:          0%          5␣
↪%          10%          25%          50%          75%          90%          95%          100%
2021-10-28 13:56:27.672 INFO scheduler.StatsReportListener:          13 %          13␣
↪%          13 %          13 %          13 %          13 %          13 %          13 %          13␣
↪%
2021-10-28 13:56:27.673 INFO scheduler.StatsReportListener: fetch wait time pct:␣
↪(count: 1, mean: 0.000000, stdev: 0.000000, max: 0.000000, min: 0.000000)
2021-10-28 13:56:27.673 INFO scheduler.StatsReportListener:          0%          5␣
↪%          10%          25%          50%          75%          90%          95%          100%
2021-10-28 13:56:27.673 INFO scheduler.StatsReportListener:          0 %          0␣
↪%          0 %          0 %          0 %          0 %          0 %          0 %          0␣
↪%
2021-10-28 13:56:27.674 INFO scheduler.StatsReportListener: other time pct: (count: 1,␣
↪ mean: 87.500000, stdev: 0.000000, max: 87.500000, min: 87.500000)
2021-10-28 13:56:27.674 INFO scheduler.StatsReportListener:          0%          5␣
↪%          10%          25%          50%          75%          90%          95%          100%
2021-10-28 13:56:27.674 INFO scheduler.StatsReportListener:          88 %          88␣
↪%          88 %          88 %          88 %          88 %          88 %          88 %          88␣
↪%
2021-10-28 13:56:27.674 INFO kyuubi.SQLOperationListener: Query [a46ca504-fe3a-4dfb-
↪be1e-19770af8ac4c]: Job 3 succeeded, 0 active jobs running
2021-10-28 13:56:27.744 INFO operation.ExecuteStatement: Processing kent's␣
↪query[a46ca504-fe3a-4dfb-be1e-19770af8ac4c]: RUNNING_STATE -> FINISHED_STATE,␣
↪statement: select timestamp '2018-11-17', time taken: 0.202 seconds
2021-10-28 13:56:27.784 INFO operation.ExecuteStatement: Query[1f619182-20ad-4733-
↪995b-a5e43b80d998] in FINISHED_STATE
2021-10-28 13:56:27.784 INFO operation.ExecuteStatement: Processing kent's␣
↪query[1f619182-20ad-4733-995b-a5e43b80d998]: RUNNING_STATE -> FINISHED_STATE,␣
↪statement: select timestamp '2018-11-17', time taken: 0.237 seconds
+---------------------------------+
| TIMESTAMP '2018-11-17 00:00:00' |
+---------------------------------+
| 2018-11-17 00:00:00.0           |
+---------------------------------+
1 row selected (0.404 seconds)
```

As shown in the above case, you can retrieve all the operation logs, the result schema, and the result to your client-side in the beeline console.

Additionally, some useful information about the background Spark SQL application associated with this connection is also printed in the operation log. For example, you can get the Spark web UI from the log for debugging or tuning.

Closing a Connection

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

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

Stopping Kyuubi

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

```
bin/kyuubi stop
```

And then, you will see the KyuubiServer waving goodbye to you.

```
Stopping org.apache.kyuubi.server.KyuubiServer
   __    __                            __
  /\ \  /\ \                          /\ \         __
  \ \ \ \/'/'   __    __    __    __   __\ \ \____/\_\
   \ \ \ , <   /\ \/\ \/\ \ \/\ \ \/\ \ \\ \ '__`\/\ \
    \ \ \\`\\ \ \_\ \ \ \ \_\ \ \ \ \_\ \\ \ \L\ \ \ \
     \ \_\ \_\/`____ \ \____/\ \____/ \ \_,__/\ \_\
      \/_/\/_/`/___/> \/___/  \/___/   \/___/  \/_/
                 /\___/
                 \/__/
Bye!
```

The `KyuubiServer` instance will be stopped immediately while the SQL engine's application will still be alive for a while.

If you start Kyuubi again before the SQL engine application terminates itself, it will reconnect to the newly created `KyuubiServer` instance.

6.1.2 Getting Started With Kyuubi on kubernetes

Running kyuubi with helm

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

Get helm and Install

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

Get Kyuubi Started

[Optional] Create namespace on kubernetes

```
cretate ns kyuubi
```

Get kyuubi started

```
helm install kyuubi-helm ${KYUUBI_HOME}/docker/helm -n ${namespace_name}
```

It will print variables and the way to get kyuubi expose ip and port.

```
NAME: kyuubi-helm
LAST DEPLOYED: Wed Oct 20 15:22:47 2021
NAMESPACE: kyuubi
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Get kyuubi expose URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace kyuubi -o jsonpath="{.spec.ports[0].
↪nodePort}" services kyuubi-helm-nodeport)
  export NODE_IP=$(kubectl get nodes --namespace kyuubi -o jsonpath="{.items[0].
↪status.addresses[0].address}")
  echo $NODE_IP:$NODE_PORT
```

Using hive beeline

Using Hive Beeline to opening a connection.

Remove kyuubi

```
helm uninstall kyuubi-helm -n ${namespace_name}
```

Edit server config

Modify `values.yaml` under `${KYUUBI_HOME}/docker/helm`:

```yaml
# Kyuubi server numbers
replicaCount: 2

image:
  repository: apache/kyuubi
  pullPolicy: Always
  # Overrides the image tag whose default is the chart appVersion.
  tag: "master-snapshot"

server:
  bind:
    host: 0.0.0.0
    port: 10009
  conf:
    mountPath: /opt/kyuubi/conf

service:
  type: NodePort
  # The default port limit of kubernetes is 30000-32767
  # to change:
  #   vim kube-apiserver.yaml (usually under path: /etc/kubernetes/manifests/)
  #   add or change line 'service-node-port-range=1-32767' under kube-apiserver
  port: 30009
```

Get server log

List all server pods:

```
kubectl get po -n ${namespace_name}
```

The server pods will print:

```
NAME                              READY   STATUS    RESTARTS   AGE
kyuubi-server-585d8944c5-m7j5s    1/1     Running   0          30m
kyuubi-server-32sdsa1245-2d2sj    1/1     Running   0          30m
```

then, use pod name to get logs:

```
kubectl -n ${namespace_name} logs kyuubi-server-585d8944c5-m7j5s
```

6.1.3 Getting Started With Kyuubi and 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 our Kyuubi.

Get to know more About DBeaver.

Preparation

Get DBeaver and Install

Please go to Download DBeaver page to get and install an appropriate release version for yourself.

Get Kyuubi Started

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

```
Welcome to
     __   __                                   __
    /\ \ /\ \                        /\ \              __
    \ \ \ \/'/'    __   __   __   __   __   __\ \ \  \____/\_\
     \ \ \ , <   /\ \/\ \/\ \/\ \/\ \/\ \ \ \ \ '__`\/\ \
      \ \ \\`\ \ \ \_\ \ \ \ \_\ \ \ \ \_\ \ \ \ \L\ \ \ \
       \ \_\ \_\/`____ \ \____/\ \____/ \ \_,__/\ \_\
        \/_/\/_/`/___/> \/___/  \/___/   \/___/  \/_/
                   /\___/
                   \/__/
```

For debugging purpose, you can `tail -f` to track the server log like:

```
$ tail -f /Users/kentyao/Downloads/kyuubi/kyuubi-1.3.0-incubating-bin/logs/kyuubi-
↪kentyao-org.apache.kyuubi.server.KyuubiServer-hulk.local.out
2021-01-16 03:27:35.449 INFO server.NIOServerCnxnFactory: Accepted socket connection␣
↪from /127.0.0.1:65320
2021-01-16 03:27:35.453 INFO server.ZooKeeperServer: Client attempting to establish␣
↪new session at /127.0.0.1:65320
2021-01-16 03:27:35.455 INFO persistence.FileTxnLog: Creating new log file: log.1
2021-01-16 03:27:35.491 INFO server.ZooKeeperServer: Established session␣
↪0x177078469840000 with negotiated timeout 60000 for client /127.0.0.1:65320
2021-01-16 03:27:35.492 INFO zookeeper.ClientCnxn: Session establishment complete on␣
↪server 127.0.0.1/127.0.0.1:2181, sessionid = 0x177078469840000, negotiated timeout␣
↪= 60000
```

```
2021-01-16 03:27:35.494 INFO state.ConnectionStateManager: State change: CONNECTED
2021-01-16 03:27:35.495 INFO client.ServiceDiscovery: Zookeeper client connection␣
↪state changed to: CONNECTED
2021-01-16 03:27:36.516 INFO client.ServiceDiscovery: Created a /kyuubi/
↪serviceUri=localhost:10009;version=1.0.2;sequence=0000000000 on ZooKeeper for␣
↪KyuubiServer uri: localhost:10009
2021-01-16 03:27:36.516 INFO client.ServiceDiscovery: Service[ServiceDiscovery] is␣
↪started.
2021-01-16 03:27:36.516 INFO server.KyuubiServer: Service[KyuubiServer] is started.
```

Configurations

Start DBeaver

If you have successfully installed DBeaver, just hit the button to launch it.

Select a database

Substantially, this step is to choose a JDBC Driver type to use later. We can choose Apache Hive or Apache Spark to set up a driver for Kyuubi, because they are compatible with the same client.
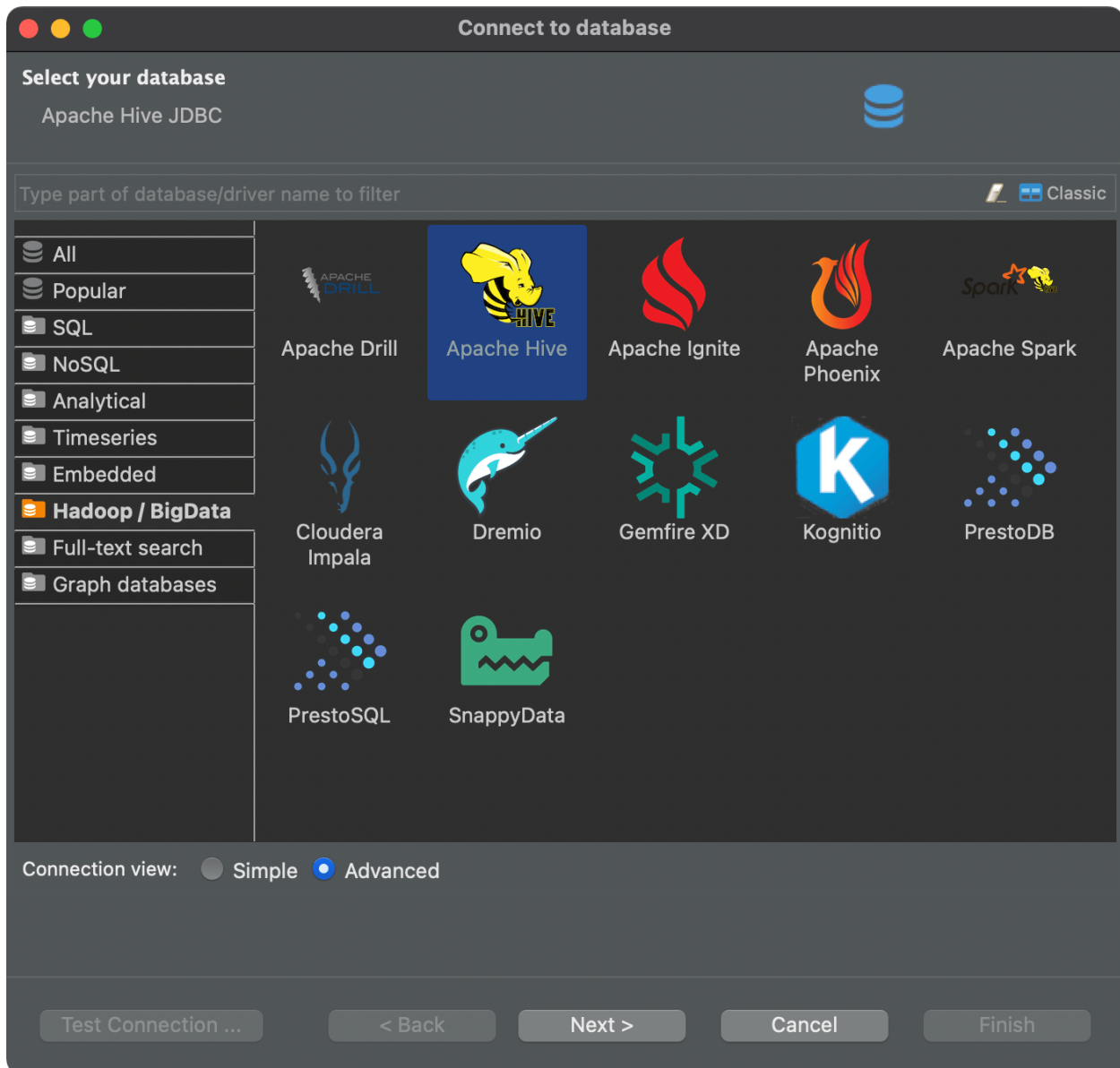
Tips: zoom up if the pic looks small

Click next. . .

Edit the Driver

We can set libraries that include the `org.apache.hive.jdbc.HiveDriver` and all of its dependencies.

Tips: zoom
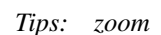
up if the pic looks small

Download/Update it. . . or,

Tips: zoom
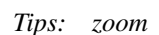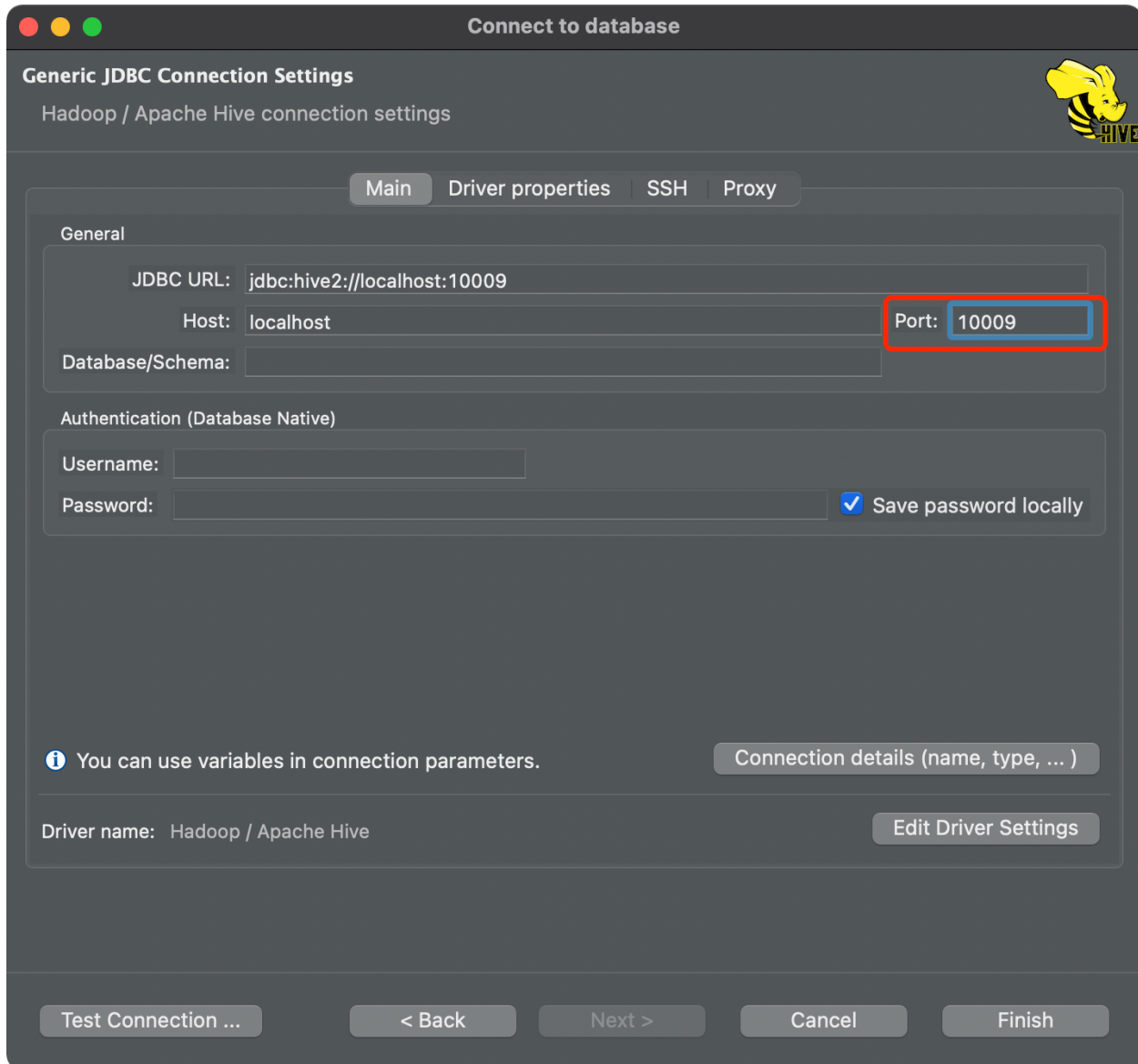
up if the pic looks small

We can configure it by adding a local folder which contains these jars.

Generic JDBC Connection Settings

To connect to Kyuubi, we should configure the right host and port that starts the server. By default, Kyuubi starts on port `10009` on your localhost.

Tips: zoom up if the pic looks small

Other settings

We also can name a recognizable title for this connection.

Tips: zoom up if the pic looks small
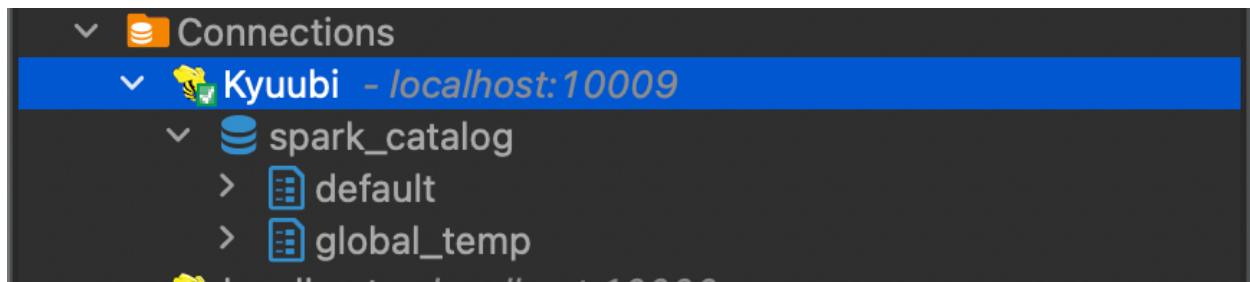
Interacting With Kyuubi server

Connections

First, we need to active the connection with Kyuubi server we created in the above steps.

Correspondingly, the server will help us start an engine, and we will be able to see a log like below,

```
2021-01-16 14:33:56.050 INFO session.KyuubiSessionImpl: Launching SQL engine:
```
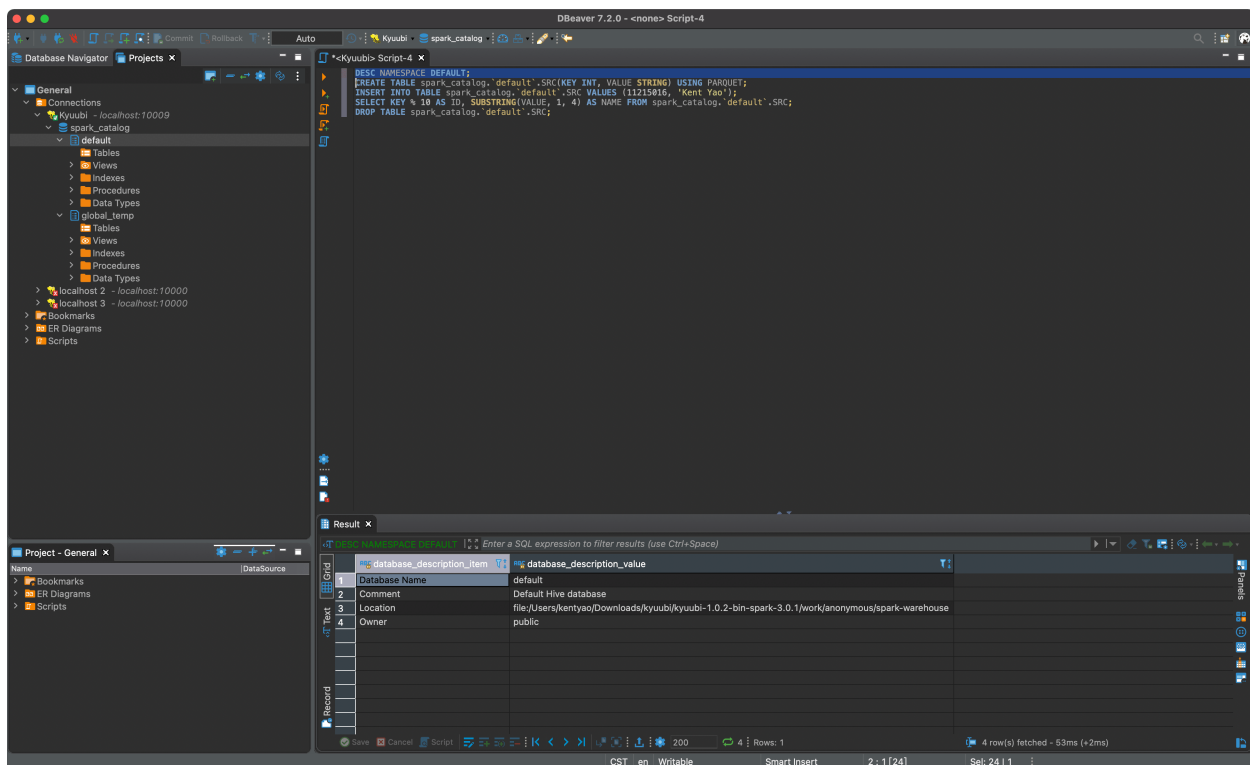
Once the connection is set up, we shall be able to see the default catalog, databases(namespaces) as below.
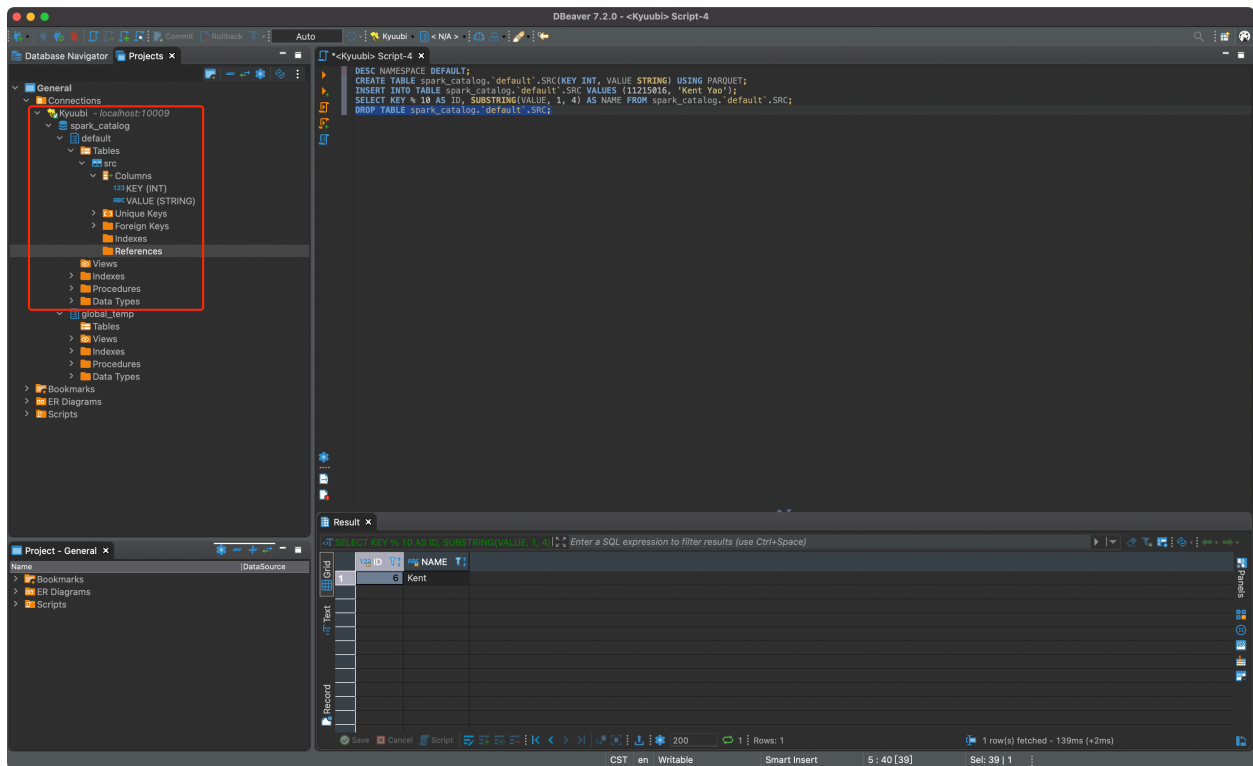
Operations

Now, we can use the SQL editor to write queries to interact with Kyuubi server through the connection.
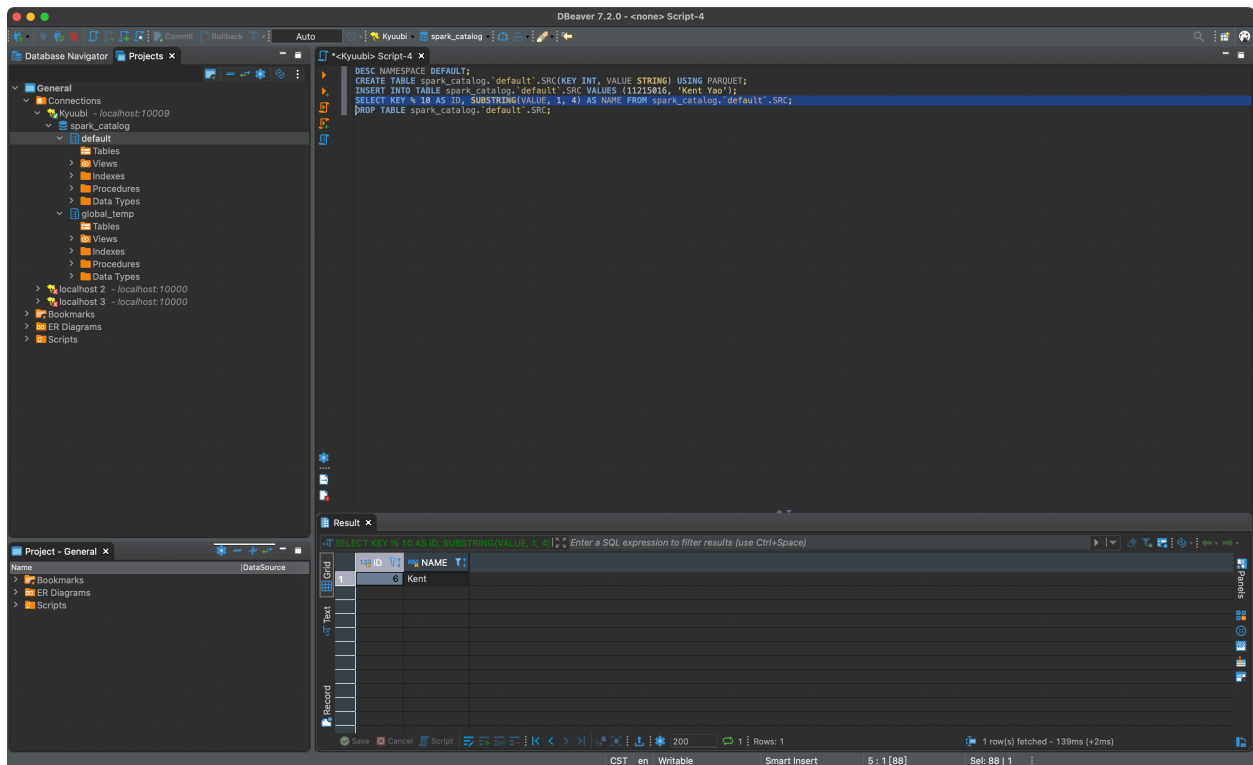
```
DESC NAMESPACE DEFAULT;
```



```
CREATE TABLE spark_catalog.`default`.SRC(KEY INT, VALUE STRING) USING PARQUET;
INSERT INTO TABLE spark_catalog.`default`.SRC VALUES (11215016, 'Kent Yao');
```

Tips: *zoom up if the pic looks small*

```
SELECT KEY % 10 AS ID, SUBSTRING(VALUE, 1, 4) AS NAME FROM spark_catalog.`default`.
↪SRC;
```
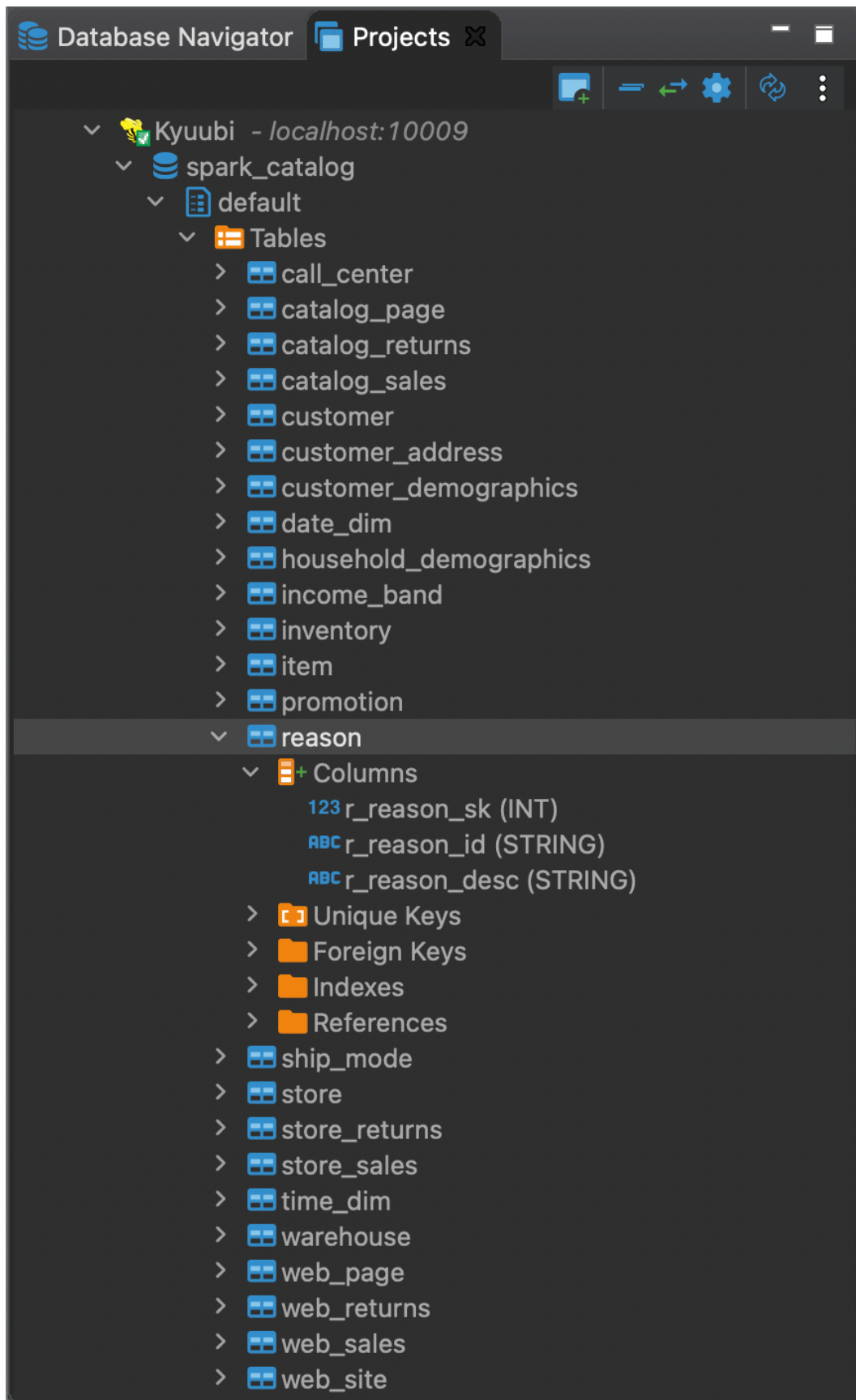

Tips: *zoom up if the pic looks small*

```
DROP TABLE spark_catalog.`default`.SRC;
```
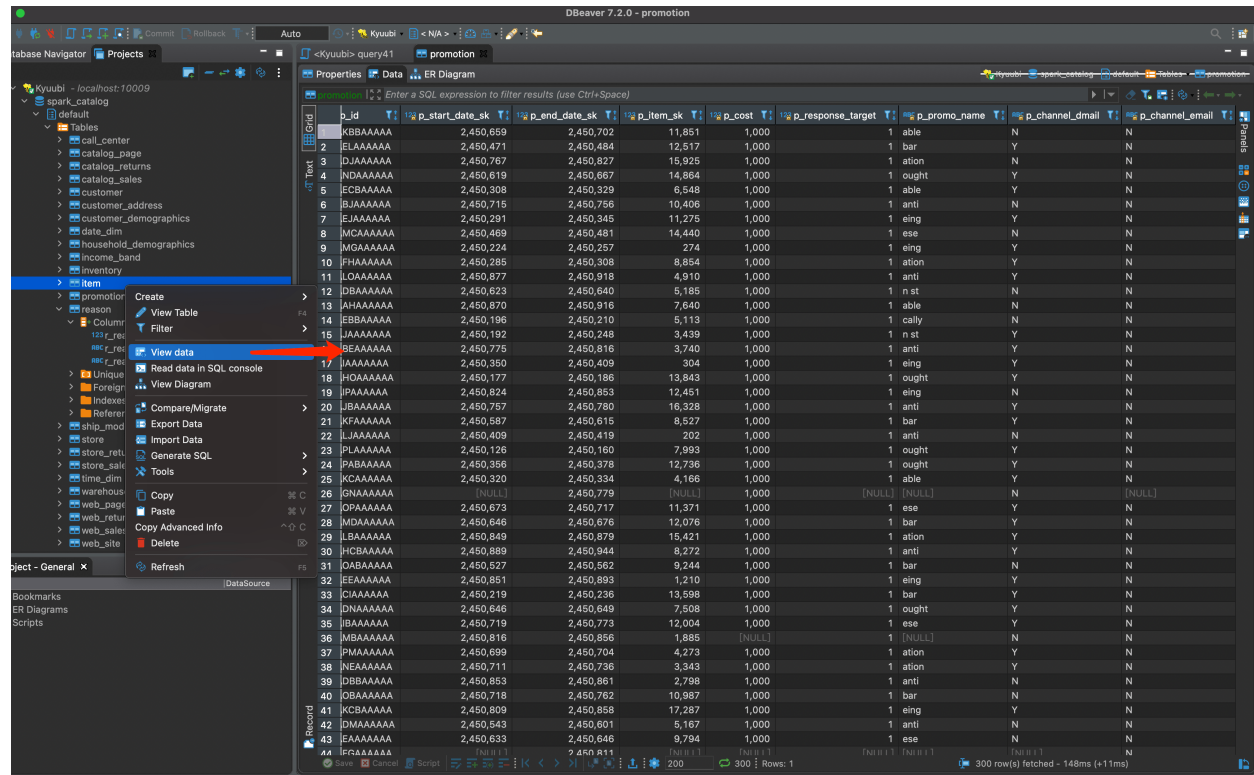
One more case with TPCDS

After we create the TPCDS table in Kyuubi server side, we are able to get all the database objects, including catalogs, databases, tables, and columns e.t.c.

Tips: zoom up if

the pic looks small

Also, we can use the shortcut key to operating metadata and data, for example.

Tips: zoom up if the pic looks small

And we can write simple or complex SQL to manipulate data, for example, here is the query 41 generated by TPCDS `dsqgen` tool.

```sql
SELECT DISTINCT (i_product_name)
FROM item i1
WHERE i_manufact_id BETWEEN 738 AND 738 + 40
  AND (SELECT count(*) AS item_cnt
FROM item
WHERE (i_manufact = i1.i_manufact AND
  ((i_category = 'Women' AND
    (i_color = 'powder' OR i_color = 'khaki') AND
    (i_units = 'Ounce' OR i_units = 'Oz') AND
    (i_size = 'medium' OR i_size = 'extra large')
  ) OR
    (i_category = 'Women' AND
      (i_color = 'brown' OR i_color = 'honeydew') AND
      (i_units = 'Bunch' OR i_units = 'Ton') AND
      (i_size = 'N/A' OR i_size = 'small')
    ) OR
    (i_category = 'Men' AND
      (i_color = 'floral' OR i_color = 'deep') AND
      (i_units = 'N/A' OR i_units = 'Dozen') AND
      (i_size = 'petite' OR i_size = 'large')
    ) OR
    (i_category = 'Men' AND
      (i_color = 'light' OR i_color = 'cornflower') AND
```

(continues on next page)

```
            (i_units = 'Box' OR i_units = 'Pound') AND
            (i_size = 'medium' OR i_size = 'extra large')
      ))) OR
  (i_manufact = i1.i_manufact AND
    ((i_category = 'Women' AND
      (i_color = 'midnight' OR i_color = 'snow') AND
      (i_units = 'Pallet' OR i_units = 'Gross') AND
      (i_size = 'medium' OR i_size = 'extra large')
    ) OR
      (i_category = 'Women' AND
        (i_color = 'cyan' OR i_color = 'papaya') AND
        (i_units = 'Cup' OR i_units = 'Dram') AND
        (i_size = 'N/A' OR i_size = 'small')
      ) OR
      (i_category = 'Men' AND
        (i_color = 'orange' OR i_color = 'frosted') AND
        (i_units = 'Each' OR i_units = 'Tbl') AND
        (i_size = 'petite' OR i_size = 'large')
      ) OR
      (i_category = 'Men' AND
        (i_color = 'forest' OR i_color = 'ghost') AND
        (i_units = 'Lb' OR i_units = 'Bundle') AND
        (i_size = 'medium' OR i_size = 'extra large')
      )))) > 0
ORDER BY i_product_name
LIMIT 100
```
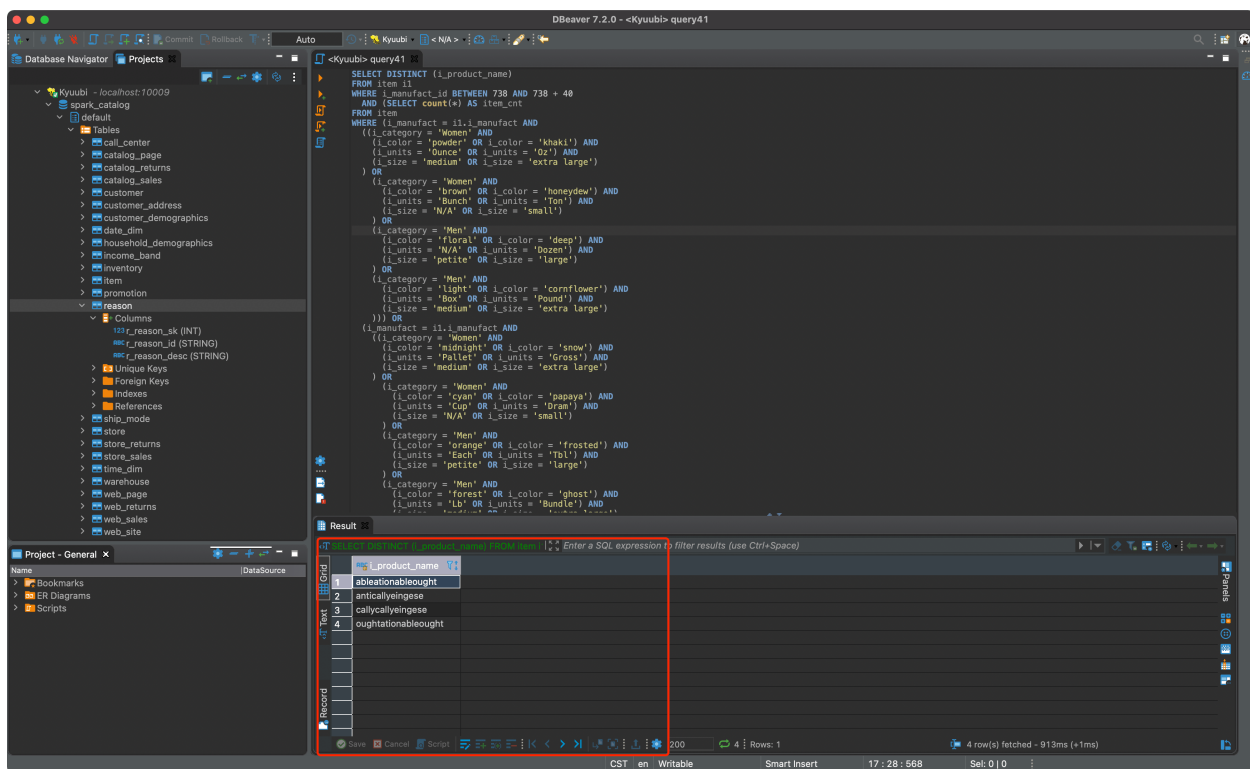

Tips: zoom up if the pic looks small

Epilogue

There are many other amazing features in both Kyuubi and DBeaver and here is just the tip of the iceberg. The rest is for you to discover.

6.1.4 Getting Started With Hive Beeline

Getting Beeline

```
$ bin/beeline --help
Usage: java org.apache.hive.cli.beeline.BeeLine
   -u <database url>               the JDBC URL to connect to
   -r                              reconnect to last saved connect url (in␣
→conjunction with !save)
   -n <username>                   the username to connect as
   -p <password>                   the password to connect as
   -d <driver class>               the driver class to use
   -i <init file>                  script file for initialization
   -e <query>                      query that should be executed
   -f <exec file>                  script file that should be executed
   -w (or) --password-file <password file>  the password file to read password from
   --hiveconf property=value       Use value for given property
   --hivevar name=value            hive variable name and value
                                   This is Hive specific settings in which variables
                                   can be set at session level and referenced in Hive
                                   commands or queries.
   --property-file=<property-file> the file to read connection properties (url,␣
→driver, user, password) from
   --color=[true/false]            control whether color is used for display
   --showHeader=[true/false]       show column names in query results
   --headerInterval=ROWS;          the interval between which heades are displayed
   --fastConnect=[true/false]      skip building table/column list for tab-completion
   --autoCommit=[true/false]       enable/disable automatic transaction commit
   --verbose=[true/false]          show verbose error messages and debug info
   --showWarnings=[true/false]     display connection warnings
   --showDbInPrompt=[true/false]   display the current database name in the prompt
   --showNestedErrs=[true/false]   display nested errors
   --numberFormat=[pattern]        format numbers using DecimalFormat pattern
   --force=[true/false]            continue running script even after errors
   --maxWidth=MAXWIDTH             the maximum width of the terminal
   --maxColumnWidth=MAXCOLWIDTH    the maximum width to use when displaying columns
   --silent=[true/false]           be more silent
   --autosave=[true/false]         automatically save preferences
   --outputformat=[table/vertical/csv2/tsv2/dsv/csv/tsv]  format mode for result␣
→display
                                   Note that csv, and tsv are deprecated - use csv2,␣
→tsv2 instead
```

```
  --incremental=[true/false]      Defaults to false. When set to false, the entire␣
↪result set
                                  is fetched and buffered before being displayed,␣
↪yielding optimal
                                  display column sizing. When set to true, result␣
↪rows are displayed
                                  immediately as they are fetched, yielding lower␣
↪latency and
                                  memory usage at the price of extra display column␣
↪padding.
                                  Setting --incremental=true is recommended if you␣
↪encounter an OutOfMemory
                                  on the client side (due to the fetched result set␣
↪size being large).
                                  Only applicable if --outputformat=table.
  --incrementalBufferRows=NUMROWS the number of rows to buffer when printing rows on␣
↪stdout,
                                  defaults to 1000; only applicable if --
↪incremental=true
                                  and --outputformat=table
  --truncateTable=[true/false]    truncate table column when it exceeds length
  --delimiterForDSV=DELIMITER     specify the delimiter for delimiter-separated␣
↪values output format (default: |)
  --isolation=LEVEL               set the transaction isolation level
  --nullemptystring=[true/false]  set to true to get historic behavior of printing␣
↪null as empty string
  --maxHistoryRows=MAXHISTORYROWS The maximum number of rows to store beeline␣
↪history.
  --help                          display this message

  Example:
  1. Connect using simple authentication to HiveServer2 on localhost:10000
  $ beeline -u jdbc:hive2://localhost:10000 username password

  2. Connect using simple authentication to HiveServer2 on hs.local:10000 using -n␣
↪for username and -p for password
  $ beeline -n username -p password -u jdbc:hive2://hs2.local:10012

  3. Connect using Kerberos authentication with hive/localhost@mydomain.com as␣
↪HiveServer2 principal
  $ beeline -u "jdbc:hive2://hs2.local:10013/default;principal=hive/
↪localhost@mydomain.com"

  4. Connect using SSL connection to HiveServer2 on localhost at 10000
  $ beeline "jdbc:hive2://localhost:10000/default;ssl=true;sslTrustStore=/usr/local/
↪truststore;trustStorePassword=mytruststorepassword"

  5. Connect using LDAP authentication
  $ beeline -u jdbc:hive2://hs2.local:10013/default <ldap-username> <ldap-password>
```

6.1.5 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`.

```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.

```xml
<!-- 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.

```java
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 {
```

(continues on next page)

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

    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();
    }
}
```

6.1.6 Getting Started With Kyuubi and 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.

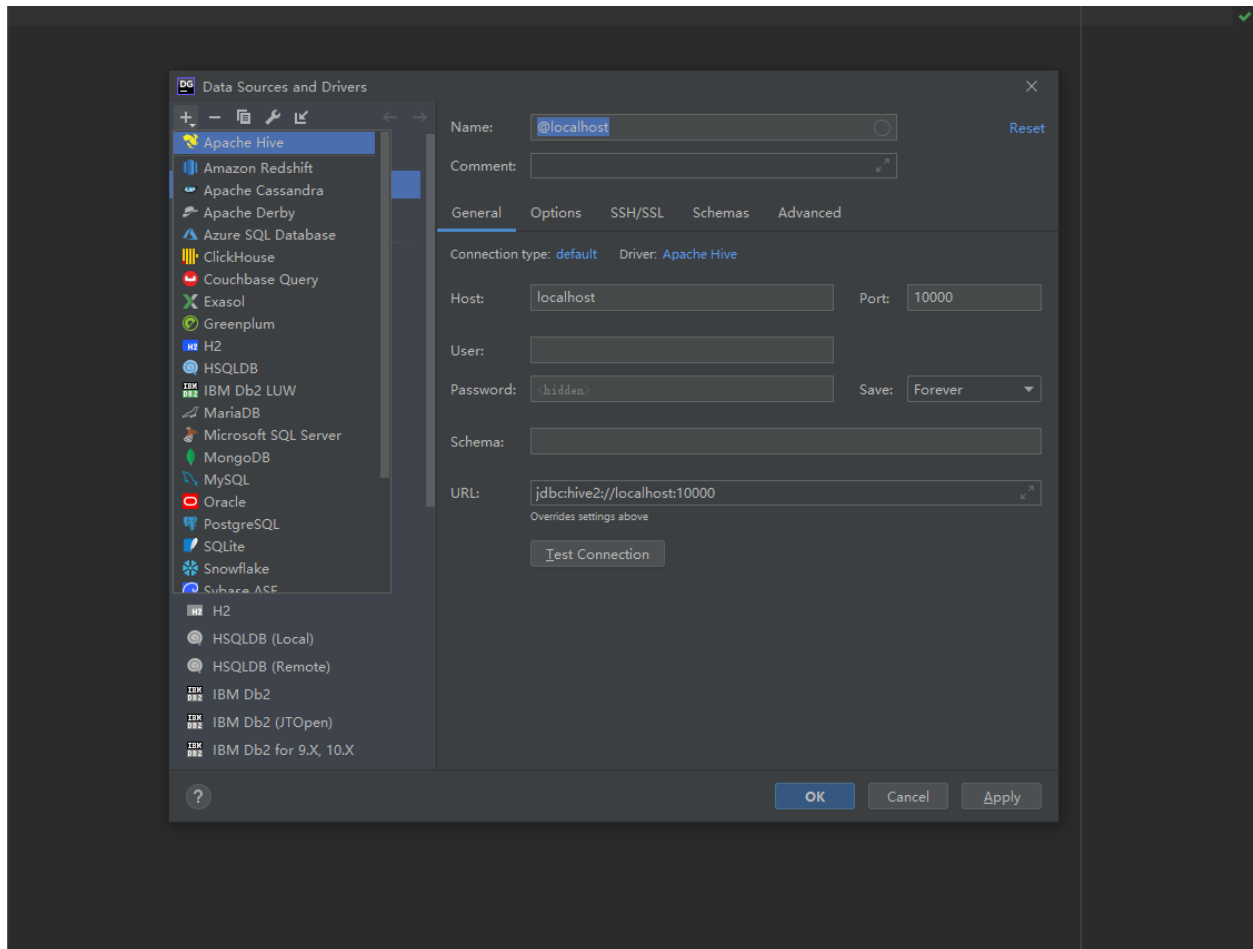For debugging purpose, you can use `tail -f` or `tailf` to track the server log.

Configurations

Start DataGrip

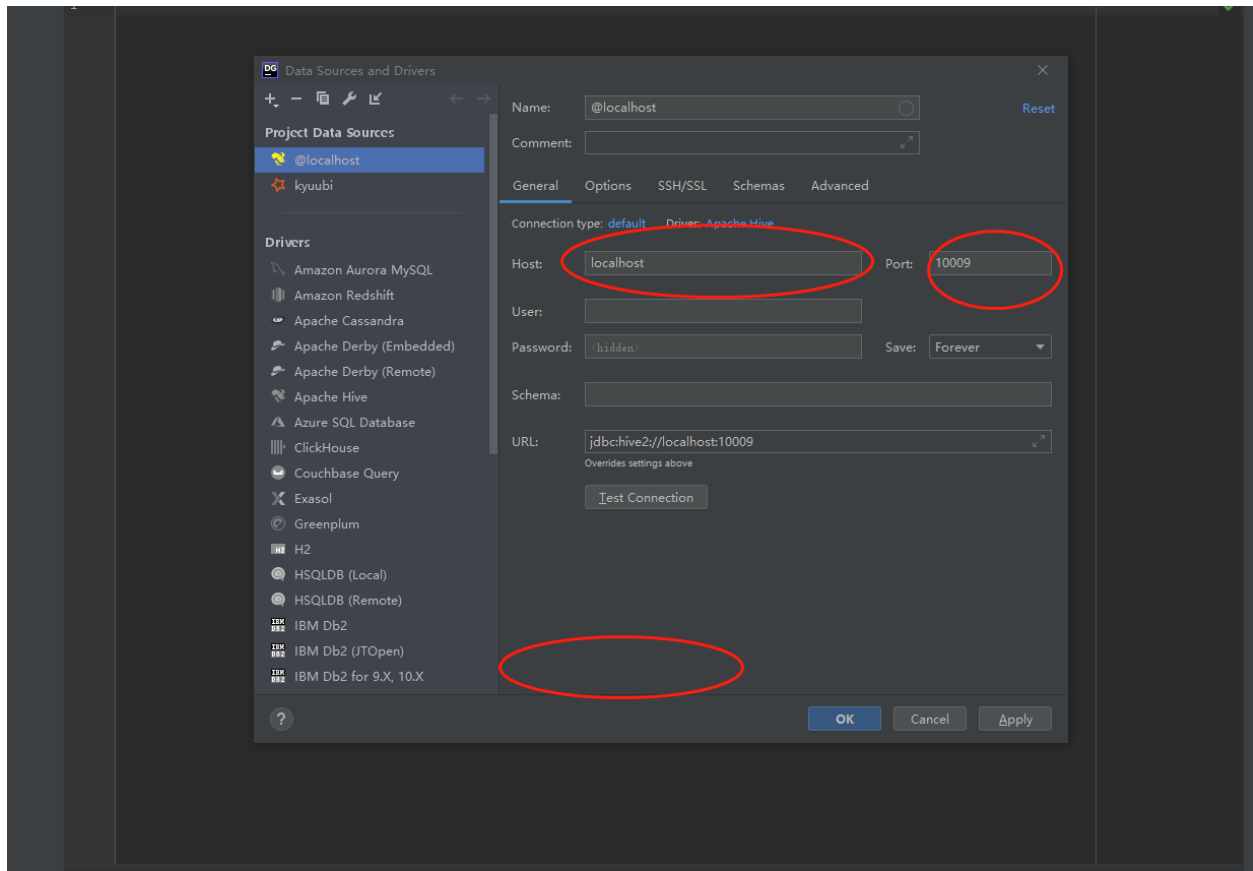After you install DataGrip, just launch it.

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.
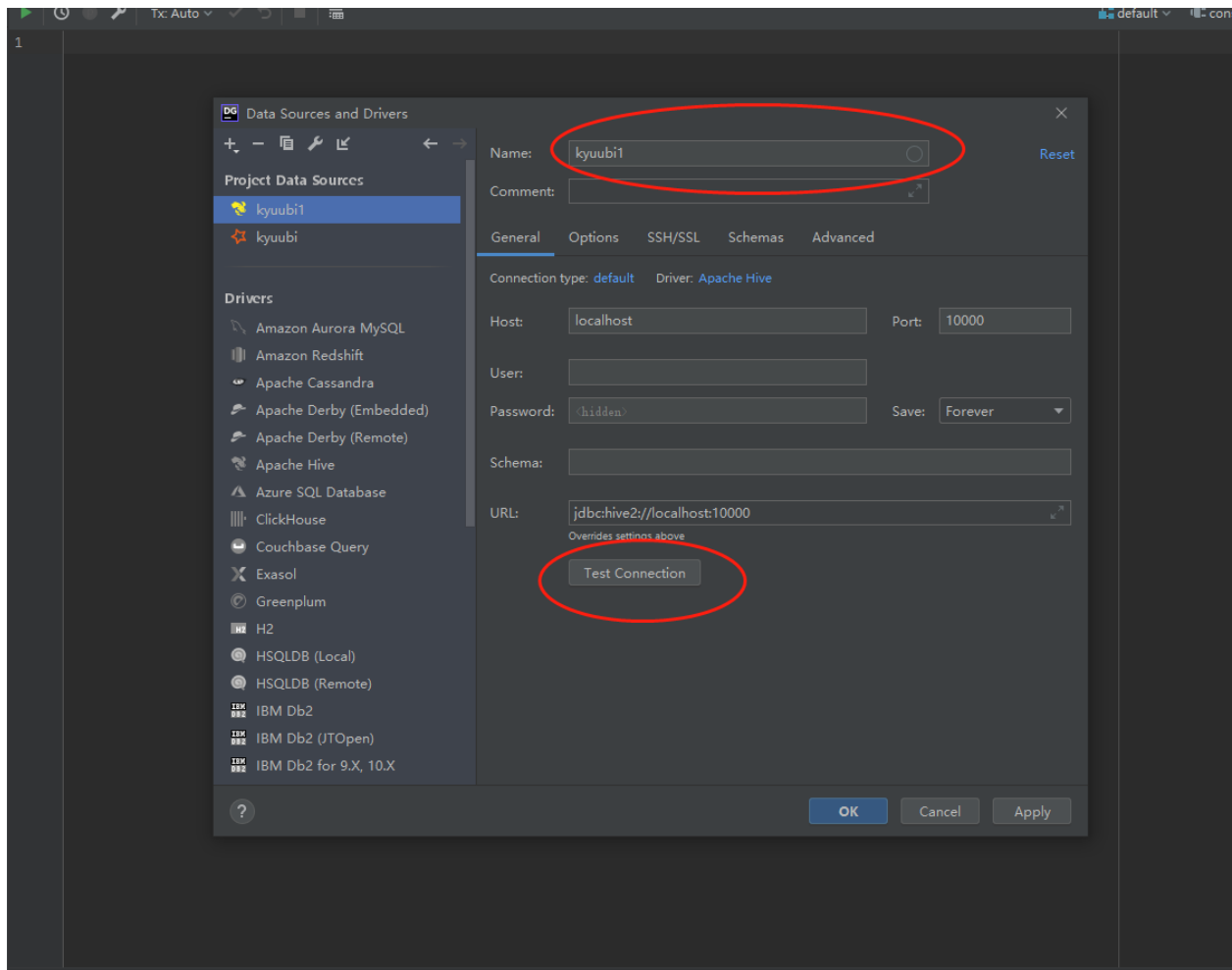
Generic JDBC Connection Settings

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.
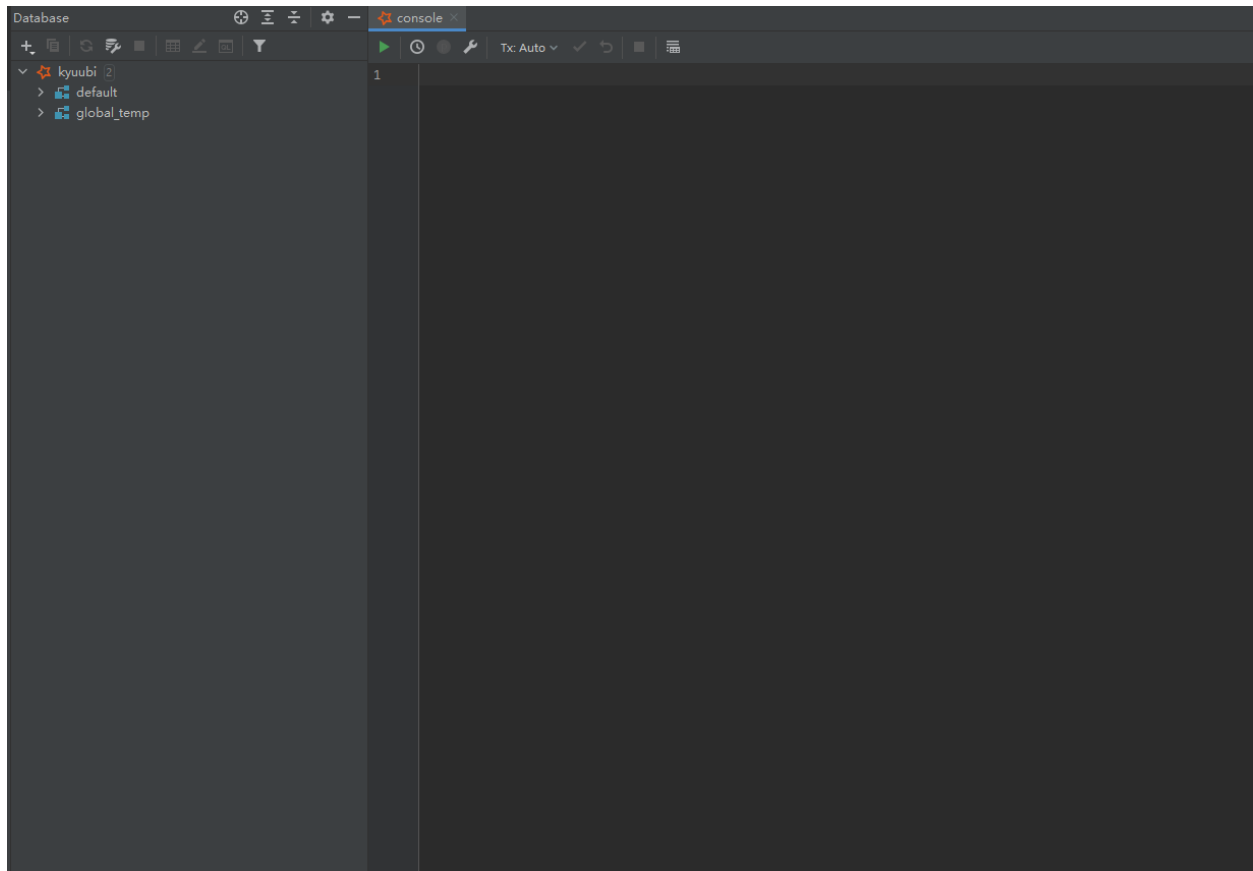
The End

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.

6.1.7 Getting Started with Kyuubi and 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\ \ \ \ \
       \ \_\ \_\/`____ \ \____/\ \____/ \ \_,__/\ \_\
        \/_/\/_/`/___/> \/___/  \/___/   \/___/   \/_/
                   /\___/
                   \/__/
```
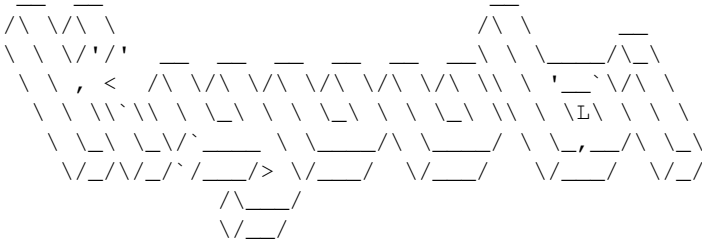
Run Hue in Docker

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`

```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!
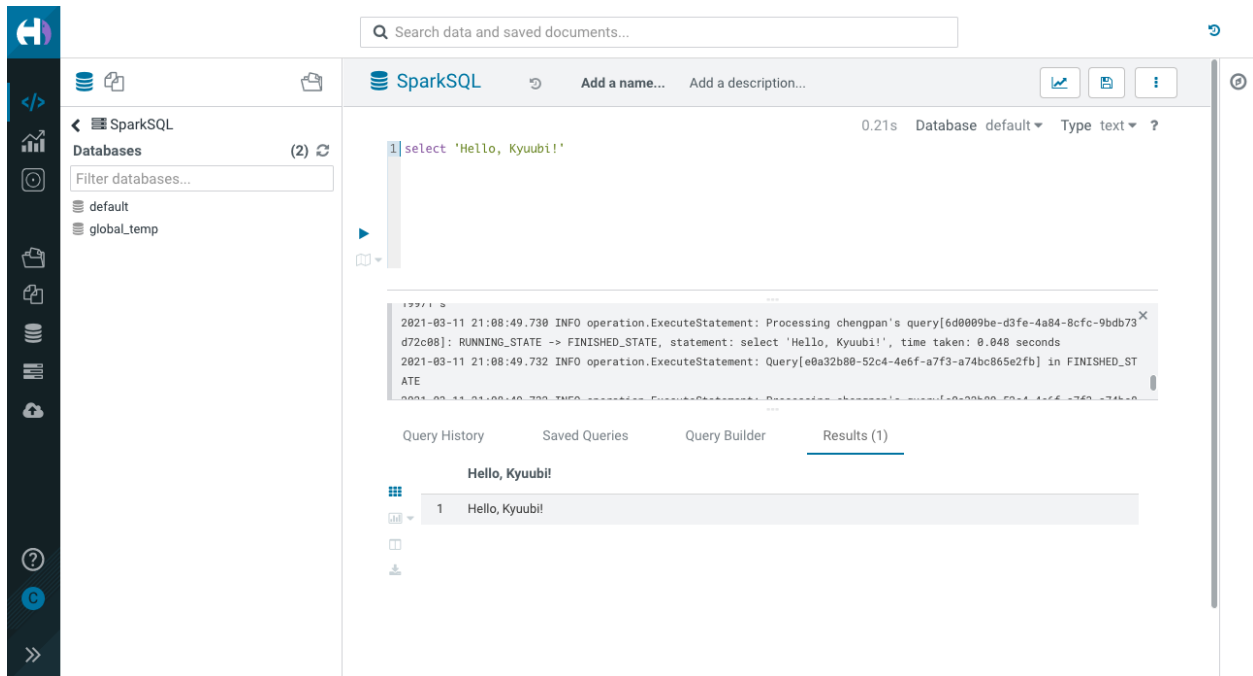
For CDH 6.x Users

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!

6.2 Deploying Kyuubi

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

6.2.1 Basics

Deploy Kyuubi engines 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 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.

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: *
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.

Load Balance Mode

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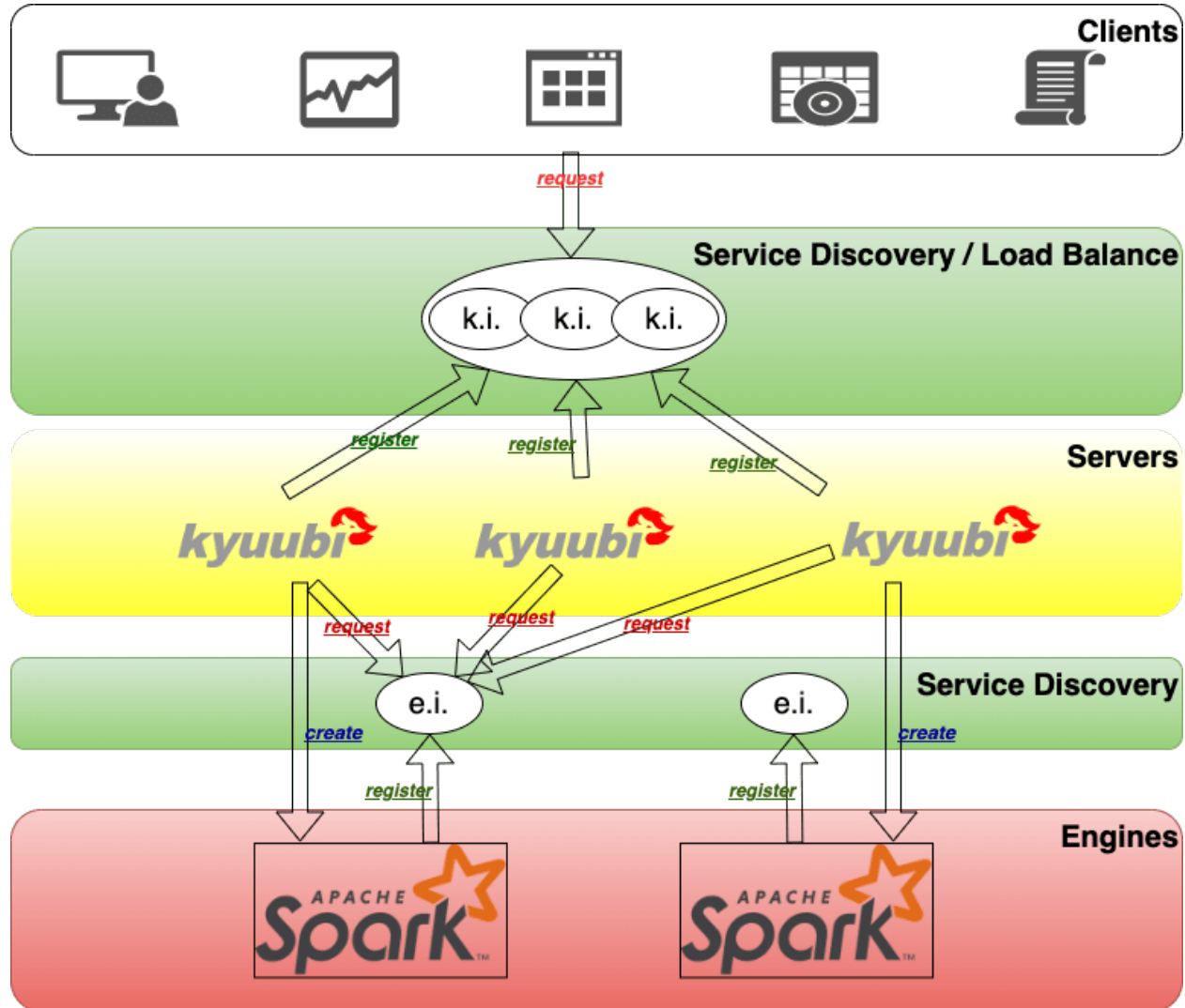
With Hive JDBC Driver, a client can specify service discovery mode in JDBC connection string, i.e. `serviceDiscoveryMode=zooKeeper;` and set `zooKeeperNamespace=kyuubi;`, then it can randomly pick one of the Kyuubi service uris from the specified ZooKeeper address in the `/kyuubi` path.

When we specify the ZooKeeper address via `kyuubi.ha.zookeeper.quorum`, load balance mode is activated by default. Please make sure that you specify the correct ZooKeeper address via `kyuubi.ha.zookeeper.quorum` and `kyuubi.ha.zookeeper.client.port`. For more details, see Introduction to the Kyuubi Configurations System

6.2.2 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.

```bash
#!/usr/bin/env bash
#
# 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.
#
#
# - JAVA_HOME               Java runtime to use. By default use "java" from PATH.
#
#
# - KYUUBI_CONF_DIR         Directory containing the Kyuubi configurations to use.
#                           (Default: $KYUUBI_HOME/conf)
# - KYUUBI_LOG_DIR          Directory for Kyuubi server-side logs.
#                           (Default: $KYUUBI_HOME/logs)
# - KYUUBI_PID_DIR          Directory stores the Kyuubi instance pid file.
#                           (Default: $KYUUBI_HOME/pid)
# - KYUUBI_MAX_LOG_FILES    Maximum number of Kyuubi server logs can rotate to.
#                           (Default: 5)
# - KYUUBI_JAVA_OPTS        JVM options for the Kyuubi server itself in the form "-
→Dx=y".
```

(continues on next page)

```
#                                    (Default: none).
# - KYUUBI_BEELINE_OPTS       JVM options for the Kyuubi BeeLine in the form "-Dx=Y".
#                                    (Default: none)
# - KYUUBI_NICENESS           The scheduling priority for Kyuubi server.
#                                    (Default: 0)
# - KYUUBI_WORK_DIR_ROOT      Root directory for launching sql engine applications.
#                                    (Default: $KYUUBI_HOME/work)
# - HADOOP_CONF_DIR           Directory containing the Hadoop / YARN configuration to␣
↪use.
#
# - SPARK_HOME                Spark distribution which you would like to use in Kyuubi.
# - SPARK_CONF_DIR            Optional directory where the Spark configuration lives.
#                                    (Default: $SPARK_HOME/conf)
#


## Examples ##

# export JAVA_HOME=/usr/jdk64/jdk1.8.0_152
# export SPARK_HOME=/opt/spark
# export HADOOP_CONF_DIR=/usr/ndp/current/mapreduce_client/conf
# export KYUUBI_JAVA_OPTS="-Xmx10g -XX:+UnlockDiagnosticVMOptions -
↪XX:ParGCCardsPerStrideChunk=4096 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -
↪XX:+CMSConcurrentMTEnabled -XX:CMSInitiatingOccupancyFraction=70 -
↪XX:+UseCMSInitiatingOccupancyOnly -XX:+CMSClassUnloadingEnabled -
↪XX:+CMSParallelRemarkEnabled -XX:+UseCondCardMark -XX:MaxDirectMemorySize=1024m  -
↪XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=./logs -verbose:gc -
↪XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -Xloggc:./
↪logs/kyuubi-server-gc-%t.log -XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -
↪XX:GCLogFileSize=5M -XX:NewRatio=3 -XX:MetaspaceSize=512m"
# export KYUUBI_BEELINE_OPTS="-Xmx2g -XX:+UnlockDiagnosticVMOptions -
↪XX:ParGCCardsPerStrideChunk=4096 -XX:+UseParNewGC -XX:+UseConcMarkSweepGC -
↪XX:+CMSConcurrentMTEnabled -XX:CMSInitiatingOccupancyFraction=70 -
↪XX:+UseCMSInitiatingOccupancyOnly -XX:+CMSClassUnloadingEnabled -
↪XX:+CMSParallelRemarkEnabled -XX:+UseCondCardMark"
```

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` in engine side is set to `/apache/confs/spark/conf`.

Kyuubi Configurations

You can configure the Kyuubi properties in `$KYUUBI_HOME/conf/kyuubi-defaults.conf`. For example:

```
#
# 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.
#

## Kyuubi Configurations

#
# kyuubi.authentication          NONE
# kyuubi.frontend.bind.host      localhost
# kyuubi.frontend.bind.port      10009
#

# Details in https://kyuubi.apache.org/docs/latest/deployment/settings.html
```

Authentication

Backend

Credentials

Delegation

Engine

Frontend

Ha

Kinit

Metrics

Operation

Session

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 affect 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

Logging

Kyuubi uses log4j for logging. You can configure it using `$KYUUBI_HOME/conf/log4j.properties`.

```
#
# 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.
#

# Set everything to be logged to the console
log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss.SSS} %p %c{2}:
↪%m%n

# Set the default kyuubi-ctl log level to WARN. When running the kyuubi-ctl, the
```

```
# log level for this class is used to overwrite the root logger's log level.
log4j.logger.org.apache.kyuubi.ctl.ServiceControlCli=ERROR

# Analysis MySQLFrontend protocol traffic
# log4j.logger.org.apache.kyuubi.server.mysql.codec=TRACE

# Kyuubi BeeLine
log4j.logger.org.apache.hive.beeline.KyuubiBeeLine=ERROR
```

Other Configurations

Hadoop Configurations

Specifying `HADOOP_CONF_DIR` to the directory contains 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 treating 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.

The 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.

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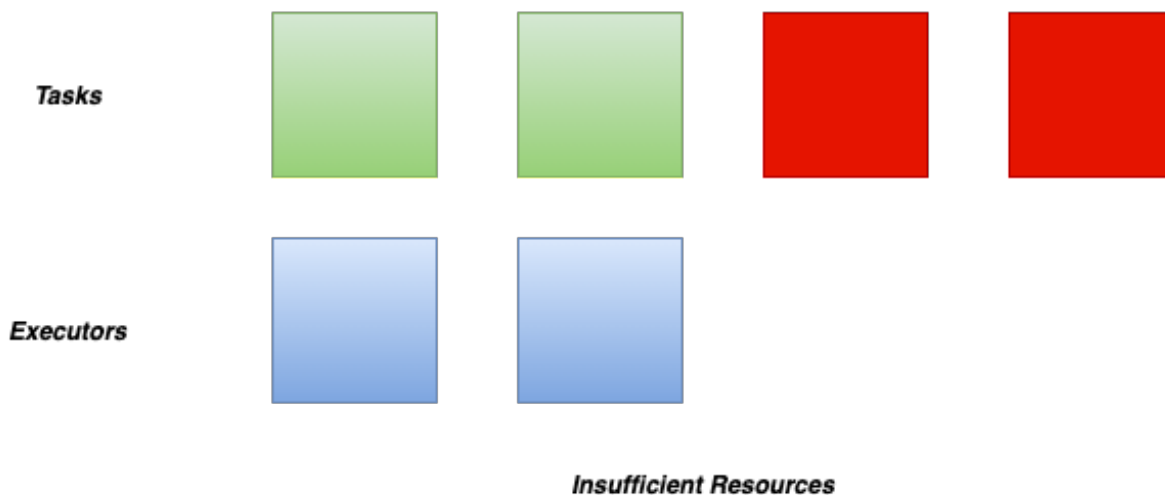
By default, the dynamic allocation will request enough executors to maximize the parallelism according to the number of tasks to process.

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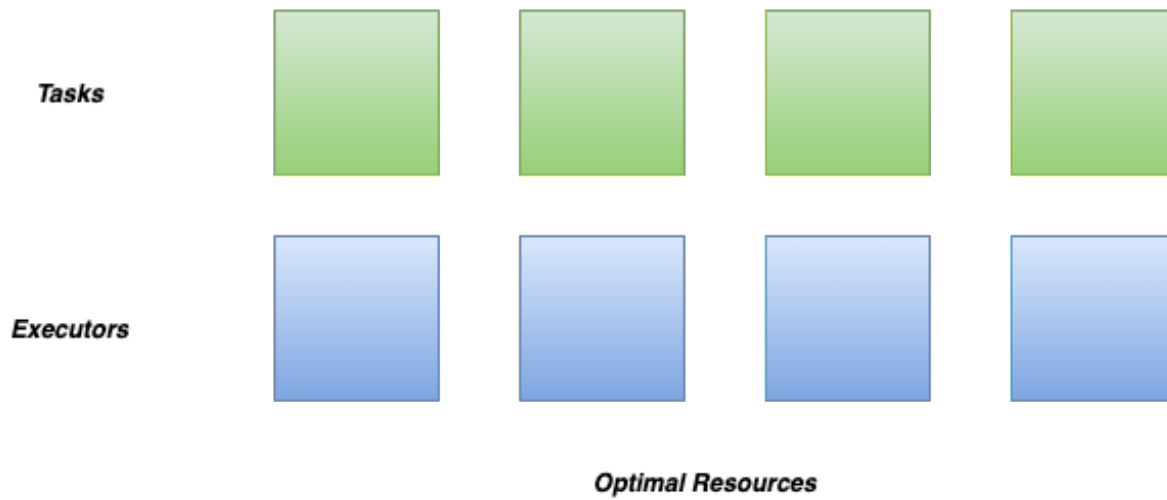
While this minimizes the latency of the job, but with small tasks, the default behavior can waste many resources due to executor allocation overhead, as some executors might not even do any work.

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.

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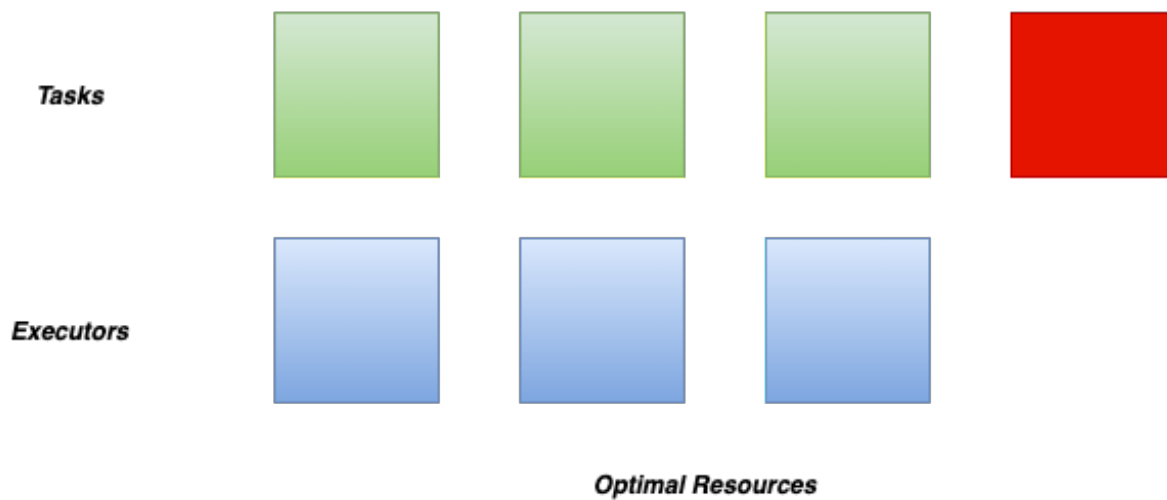
After finish one task, Spark Driver will schedule a new task for the executor with available cores. When pending tasks become fewer and fewer, some executors become idle for no new coming tasks.

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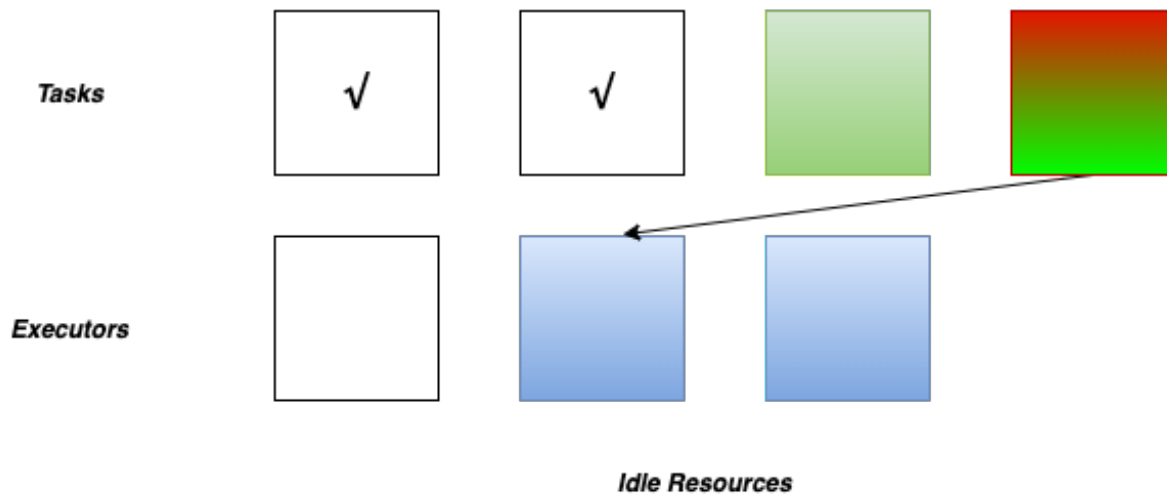
If one executor reached the maximum timeout, it will be removed.

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


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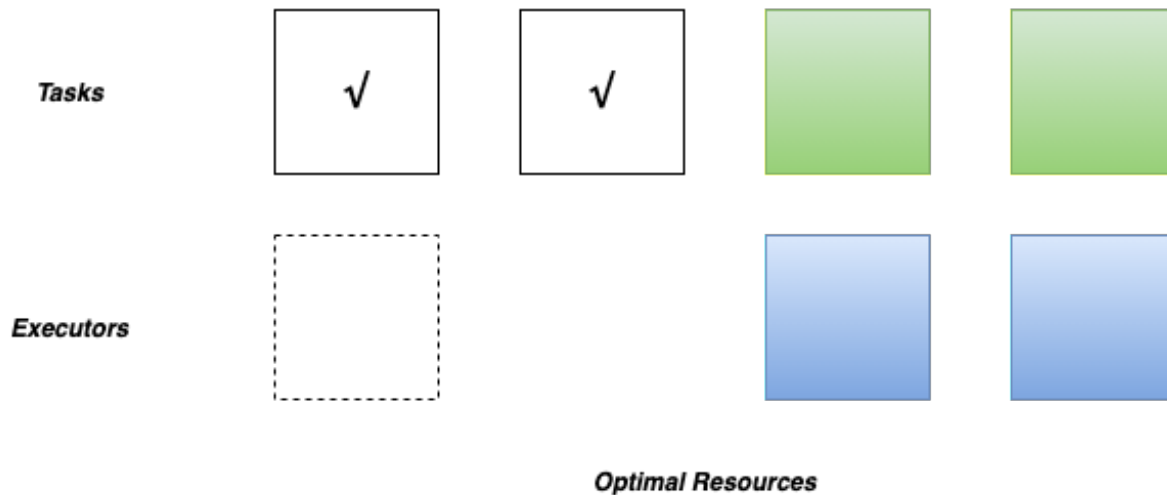
If the DRA finds there have been pending tasks backlogged for more than the timeouts, new executors will be requested, controlled by the following configs.

```
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 perfer 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 perfer 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` syntaxes before each SQL query. For example,

```sql
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`
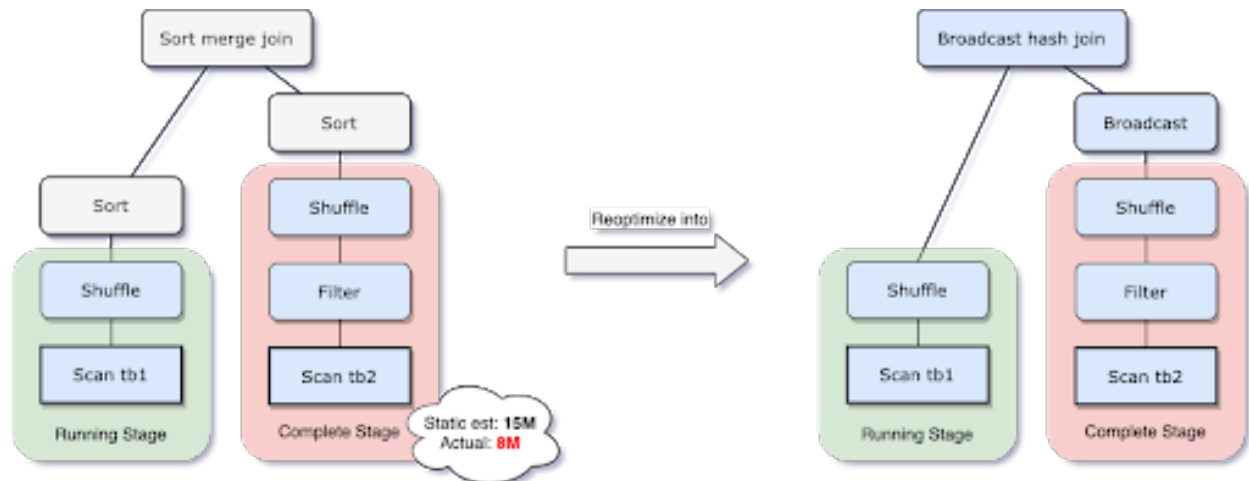
What's more, when `spark.sql.adaptive.localShuffleReader.enabled=true` and after converting `SortMerge Join` to `BroadcastHash Join`, Spark also does future optimize to reduce the network traffic by converting a regular shuffle to a localized shuffle.
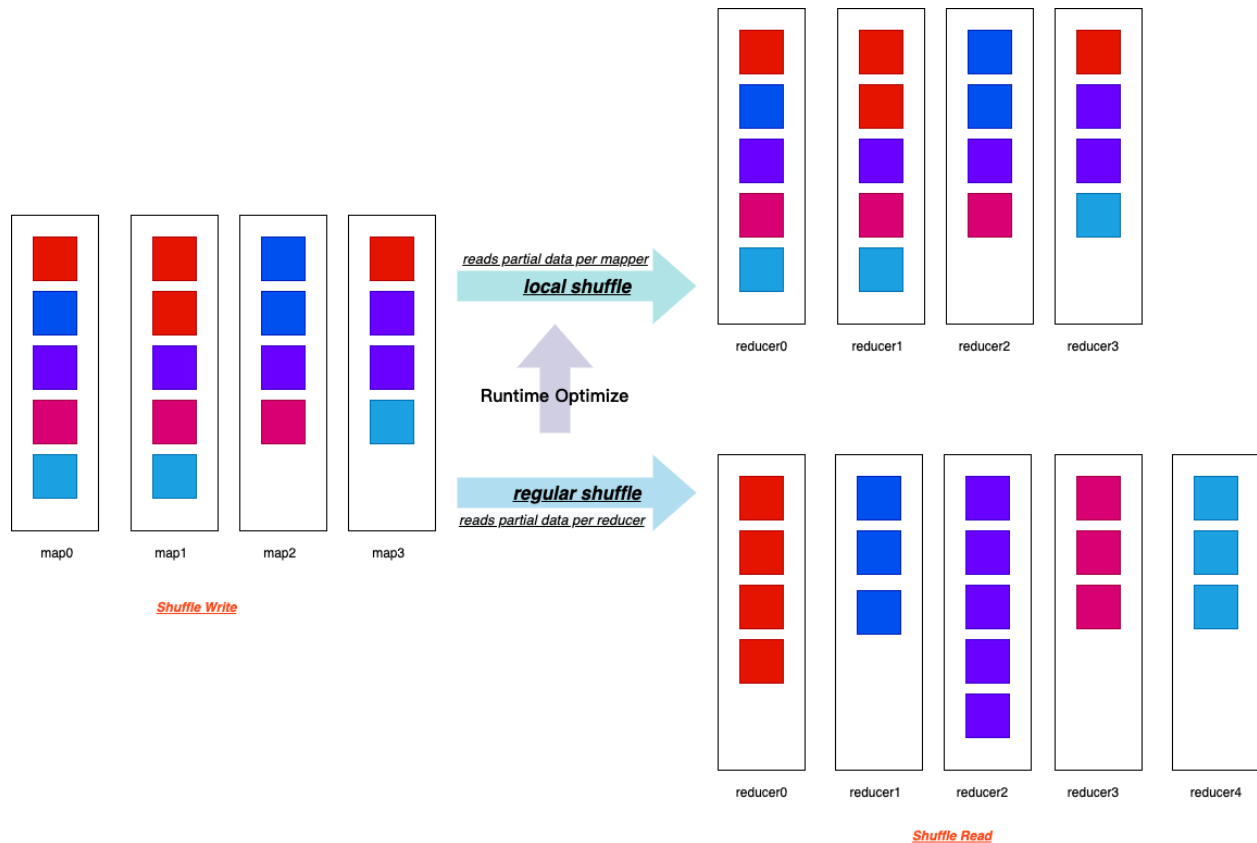
As shown in the above fig, the local shuffle reader can read all necessary shuffle files from its local storage, actually without performing the shuffle across the network.

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.

No AQE Partition Coalescing
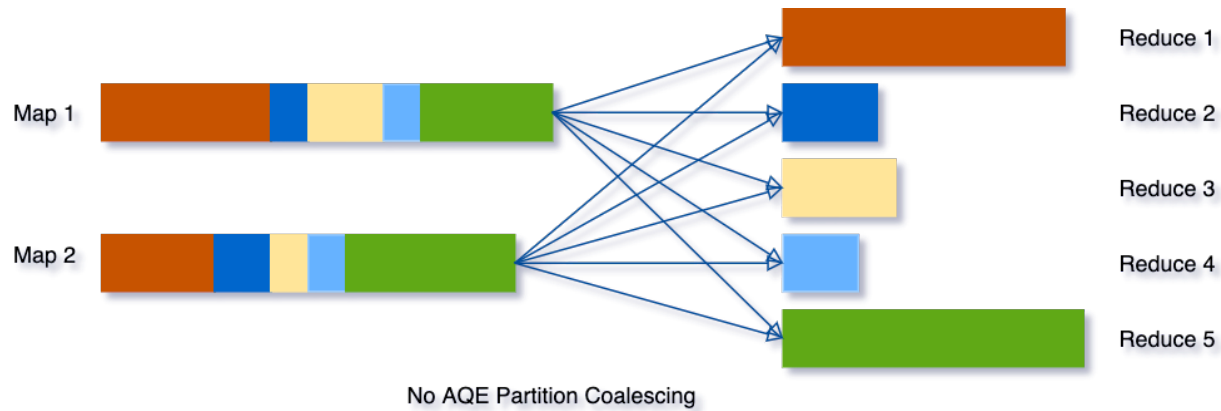
Combining small partitions saves resources and improves cluster throughput. Spark provides several ways to handle small file issues, for example, adding an extra shuffle operation on the partition columns with the `distribute by` clause or using `HINT`[5]. In most scenarios, you need to have a good grasp 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.

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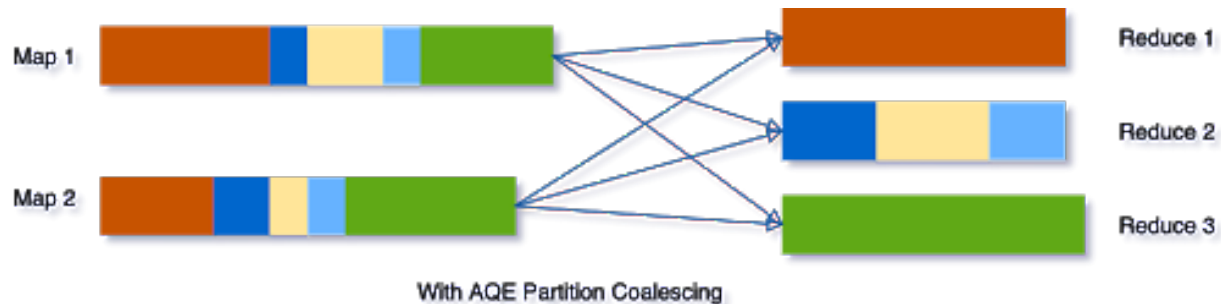
It can simplify the tuning of shuffle partition numbers when running Spark SQL queries. You do not need to set a proper shuffle partition number to fit your dataset.

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.
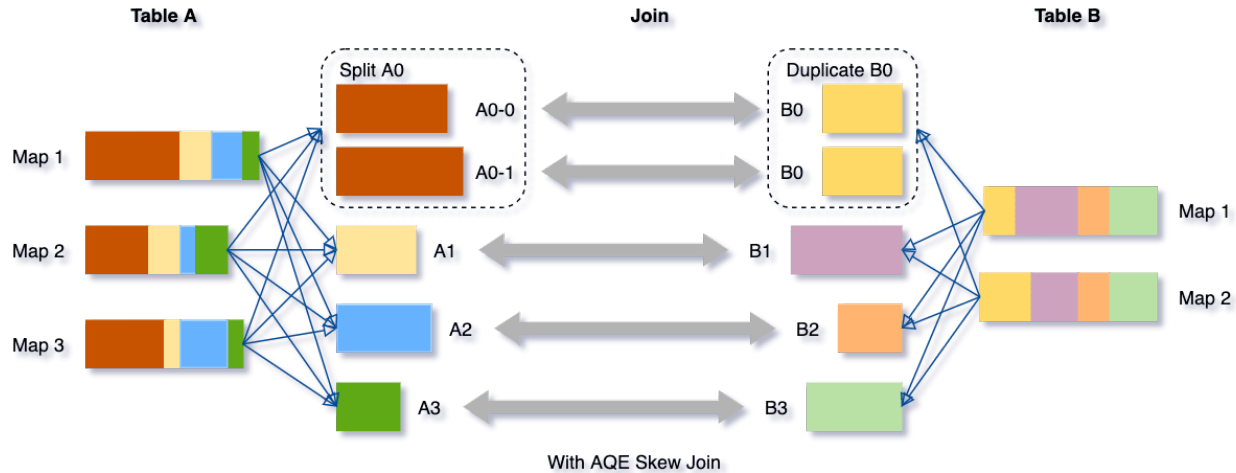
With AQE Skew Join

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

6.2.3 Engines

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 sharability 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.

6.3 Kyuubi Security Overview

6.3.1 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 that, this authentication only authenticate whether a user can connect with Kyuubi server or not. 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.

In `$KYUUBI_HOME/conf/kyuubi-defaults.conf`, specify `kyuubi.authentication` to one of the authentication types listing below.

Using KERBEROS

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

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.

Set following for KERBEROS mode:

For example,

- Configure with Kyuubi service principal

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

- Start Kyuubi

```
$ ./bin/kyuubi start
```

- Kinit with user principal and connect using beeline

```
$ kinit -kt user.keytab user.principal

$ beeline -u "jdbc:hive2://localhost:10009/;principal=spark/kyuubi.apache.org@KYUUBI.
↪APACHE.ORG"
```

6.3.2 Kinit Auxiliary Service

In order to work with a kerberos-enabled cluster, Kyuubi provides this kinit auxiliary service. It will periodically re-kinit with to keep the Ticket Cache fresh.

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 `hadoop.security.authentication` is set to `KERBEROS`, in `$HADOOP_CONF_DIR/core-site` or `$KYUUBI_HOME/conf/kyuubi-defaults.conf`, it indicates that we are targeting a secured cluster, then we need to specify `kyuubi.kinit.principal` and `kyuubi.kinit.keytab` for authentication.

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

6.3.3 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

6.3.4 ACL Management Guide

- *Authorization Modes*
    - *Storage-Based Authorization*
    - *SQL-Standard Based Authorization*
    - *Ranger Security Support*

Authorization Modes

Three primary modes for Kyuubi authorization are available by Submarine Spark Security:

Storage-Based Authorization

Enabling Storage Based Authorization in the `Hive Metastore Server` uses the HDFS permissions to act as the main source for verification and allows for consistent data and metadata authorization policy. This allows control over metadata access by verifying if the user has permission to access corresponding directories on the HDFS. Similar with `HiveServer2`, files and directories will be translated into hive metadata objects, such as dbs, tables, partitions, and be protected from end user's queries through Kyuubi.

Storage-Based Authorization offers users with Database, Table and Partition-level coarse-gained access control.

SQL-Standard Based Authorization

Enabling SQL-Standard Based Authorization gives users more fine-gained control over access comparing with Storage Based Authorization. Besides of the ability of Storage Based Authorization, SQL-Standard Based Authorization can improve it to Views and Column-level. Unfortunately, Spark SQL does not support grant/revoke statements which controls access, this might be done only through the HiveServer2. But it's gratifying that Submarine Spark Security makes Spark SQL be able to understand this fine-grain access control granted or revoked by Hive.

With Kyuubi, the SQL-Standard Based Authorization is guaranteed for the security configurations, metadata, and storage information is preserved from end users.

Please refer to the Submarine Spark Security in the online documentation for an overview on how to configure SQL-Standard Based Authorization for Spark SQL.

Ranger Security Support (Recommended)

Apache Ranger is a framework to enable, monitor and manage comprehensive data security across the Hadoop platform but end before Spark or Spark SQL. The Submarine Spark Security enables Kyuubi with control access ability reusing Ranger Plugin for Hive MetaStore . Apache Ranger makes the scope of existing SQL-Standard Based Authorization expanded but without supporting Spark SQL. Submarine Spark Security sticks them together.

Please refer to the Submarine Spark Security in the online documentation for an overview on how to configure Ranger for Spark SQL.

6.4 Client Documentation

6.4.1 Access Kyuubi with Hive JDBC and ODBC Drivers

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>;<sessionConfs>?<sparkConfs>#<[spark|hive]Vars>
```

Example

```
jdbc:hive2://localhost:10009/default;spark.sql.adaptive.enabled=true?spark.ui.
↪enabled=false#var_x=y
```

Unsupported Hive Features

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

6.5 Integrations

6.5.1 Kyuubi On Apache 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
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
<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)
        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/incubator-kyuubi https://spark.apache.org/docs/latest/sql-data-sources.html

6.6 Monitoring

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

6.6.1 Monitoring Kyuubi - Logging System

Kyuubi uses Apache Log4j for logging.

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 `log4j.properties` file in the `$KYUUBI_HOME/conf` directory. One way to start is to make a copy of the existing `log4j.properties.template` located there.

For example,

```
# cd $KYUUBI_HOME
cp conf/log4j.properties.template conf/log4j.properties
```

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
```

(continues on next page)

```
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:
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 Log4j Settings

Taking control of `$KYUUBI_HOME/conf/log4j.properties` 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,

```
log4j.rootCategory=INFO, FA
log4j.appender.FA=org.apache.log4j.FileAppender
log4j.appender.FA.append=false
log4j.appender.FA.file=log/dummy.log
log4j.appender.FA.layout=org.apache.log4j.PatternLayout
log4j.appender.FA.layout.ConversionPattern=%d{HH:mm:ss.SSS} %t %p %c{2}: %m%n
log4j.appender.FA.Threshold=DEBUG
```

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.

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[*]
```

(continues on next page)

```
              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
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%
```

```
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)
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

6.6.2 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:

6.6.3 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)
        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$2.doSubmit(SparkSubmit.
→scala:1007)
        at org.apache.spark.deploy.SparkSubmit$.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$1.
→run(ExecuteStatement.scala:80)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
```

(continues on next page)

```
        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 URLClass-Loader.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.

6.7 SQL References

This part describes the use of the SQL References in Kyuubi, including lists of the available extension, data types and functions for use in SQL commands.

6.7.1 Auxiliary SQL extension for Spark SQL

Kyuubi provides SQL extension out of box. Due to the version compatibility with Apache Spark, currently we only support Apache Spark branch-3.1 (i.e 3.1.1 and 3.1.2). 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.

What feature does Kyuubi SQL extension provide

- 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.

How to use Kyuubi SQL extension

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 `dev/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, and also Kyuubi provides some configs to make these feature easy to use.

6.7.2 Auxiliary SQL Functions for Spark SQL

Kyuubi provides several auxiliary SQL functions as supplement to Spark's Built-in Functions

6.7.3 Z-order Benchmark

Z-order is a technique that allows you to map multidimensional data to a single dimension. We did a performance test.

For this test ,we used aliyun Databricks Delta test case https://help.aliyun.com/document_detail/168137.html?spm=a2c4g.11186623.6.56

Prepare data for the three scenarios:

1. 10 billion data and 2 hundred files (parquet files): for big file(1G)

2. 10 billion data and 1 thousand files (parquet files): for medium file(200m)

3. 1 billion data and 10 thousand files (parquet files): for smaller file(200k)

Test env: spark-3.1.2 hadoop-2.7.2 kyuubi-1.4.0

Test step:

Step1: create hive tables.

```
spark.sql(s"drop database if exists $dbName cascade")
spark.sql(s"create database if not exists $dbName")
spark.sql(s"use $dbName")
spark.sql(s"create table $connRandomParquet (src_ip string, src_port int, dst_ip
→string, dst_port int) stored as parquet")
spark.sql(s"create table $connOrderbyOnlyIp (src_ip string, src_port int, dst_ip
→string, dst_port int) stored as parquet")
spark.sql(s"create table $connOrderby (src_ip string, src_port int, dst_ip string,
→dst_port int) stored as parquet")
spark.sql(s"create table $connZorderOnlyIp (src_ip string, src_port int, dst_ip
→string, dst_port int) stored as parquet")
spark.sql(s"create table $connZorder (src_ip string, src_port int, dst_ip string, dst_
→port int) stored as parquet")
spark.sql(s"show tables").show(false)
```

Step2: prepare data for parquet table with three scenarios, we use the following code.

```
def randomIPv4(r: Random) = Seq.fill(4)(r.nextInt(256)).mkString(".")
def randomPort(r: Random) = r.nextInt(65536)

def randomConnRecord(r: Random) = ConnRecord(
  src_ip = randomIPv4(r), src_port = randomPort(r),
  dst_ip = randomIPv4(r), dst_port = randomPort(r))
```

Step3: do optimize with z-order only ip and do optimize with order by only ip, sort column: src_ip, dst_ip and shuffle partition just as file numbers.

```
INSERT overwrite table conn_order_only_ip select src_ip, src_port, dst_ip, dst_port
→from conn_random_parquet order by src_ip, dst_ip;
OPTIMIZE conn_zorder_only_ip ZORDER BY src_ip, dst_ip;
```

Step4: do optimize with z-order and do optimize with order by, sort column: src_ip, src_port, dst_ip, dst_port and shuffle partition just as file numbers.

```
INSERT overwrite table conn_order select src_ip, src_port, dst_ip, dst_port from conn_
→random_parquet order by src_ip, src_port, dst_ip, dst_port;
OPTIMIZE conn_zorder ZORDER BY src_ip, src_port, dst_ip, dst_port;
```

The complete code is as follows:

```
./spark-shell
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession

case class ConnRecord(src_ip: String, src_port: Int, dst_ip: String, dst_port: Int)

val  conf  = new SparkConf().setAppName("zorder_test")
val spark = SparkSession.builder().config(conf).enableHiveSupport().getOrCreate()
import spark.implicits._

val sc = spark.sparkContext
sc.setLogLevel("WARN")
//ten billion rows and two hundred files
val numRecords = 10*1000*1000*1000L
val numFiles = 200

val dbName = s"zorder_test_$numFiles"
val baseLocation = s"hdfs://localhost:9000/zorder_test/$dbName/"
val connRandomParquet = "conn_random_parquet"
val connZorderOnlyIp = "conn_zorder_only_ip"
val connZorder = "conn_zorder"
spark.conf.set("spark.sql.shuffle.partitions", numFiles)
spark.conf.get("spark.sql.shuffle.partitions")
spark.conf.set("spark.sql.hive.convertMetastoreParquet",false)
spark.sql(s"drop database if exists $dbName cascade")
spark.sql(s"create database if not exists $dbName")
spark.sql(s"use $dbName")
spark.sql(s"create table $connRandomParquet (src_ip string, src_port int, dst_ip␣
↪string, dst_port int) stored as parquet")
spark.sql(s"create table $connOrderbyOnlyIp (src_ip string, src_port int, dst_ip␣
↪string, dst_port int) stored as parquet")
spark.sql(s"create table $connOrderby (src_ip string, src_port int, dst_ip string,␣
↪dst_port int) stored as parquet")
spark.sql(s"create table $connZorderOnlyIp (src_ip string, src_port int, dst_ip␣
↪string, dst_port int) stored as parquet")
spark.sql(s"create table $connZorder (src_ip string, src_port int, dst_ip string, dst_
↪port int) stored as parquet")
spark.sql(s"show tables").show(false)

import scala.util.Random
// Function for preparing Zorder_Test data
def randomIPv4(r: Random) = Seq.fill(4)(r.nextInt(256)).mkString(".")
def randomPort(r: Random) = r.nextInt(65536)

def randomConnRecord(r: Random) = ConnRecord(
src_ip = randomIPv4(r), src_port = randomPort(r),
dst_ip = randomIPv4(r), dst_port = randomPort(r))

val df = spark.range(0, numFiles, 1, numFiles).mapPartitions { it =>
val partitionID = it.toStream.head
val r = new Random(seed = partitionID)
Iterator.fill((numRecords / numFiles).toInt)(randomConnRecord(r))
}

df.write
.mode("overwrite")
.format("parquet")
```

```
.insertInto(connRandomParquet)

spark.read.table(connRandomParquet)
.write
.mode("overwrite")
.format("parquet")
.insertInto(connZorderOnlyIp)

spark.read.table(connRandomParquet)
.write
.mode("overwrite")
.format("parquet")
.insertInto(connZorder)
spark.stop()
```

Z-order Optimize statement:

```
set spark.sql.hive.convertMetastoreParquet=false;

OPTIMIZE conn_zorder_only_ip ZORDER BY src_ip, dst_ip;

OPTIMIZE zorder_test.conn_zorder ZORDER BY src_ip, src_port, dst_ip, dst_port;
```

ORDER BY statement:

```
INSERT overwrite table conn_order_only_ip select src_ip, src_port, dst_ip, dst_port
→from conn_random_parquet order by src_ip, dst_ip;

INSERT overwrite table conn_order select src_ip, src_port, dst_ip, dst_port from conn_
→random_parquet order by src_ip, src_port, dst_ip, dst_port;
```

Query statement:

```
set spark.sql.hive.convertMetastoreParquet=true;

select count(*) from conn_random_parquet where src_ip like '157%' and dst_ip like
→'216.%';

select count(*) from conn_zorder_only_ip where src_ip like '157%' and dst_ip like
→'216.%';

select count(*) from conn_zorder where src_ip like '157%' and dst_ip like '216.%';
```

Benchmark result

We have done two performance tests: one is to compare the efficiency of Z-order Optimize and Order by Sort, and the other is to query based on the optimized Z-order by data and Random data.

Efficiency of Z-order Optimize and Order-by Sort

10 billion data and 1000 files and Query resource: 200 core 600G memory

Z-order by or order by only ip:

Z-order by or order by all columns:

Z-order by benchmark result

By querying the tables before and after optimization, we find that:

10 billion data and 200 files and Query resource: 200 core 600G memory

10 billion data and 1000 files and Query resource: 200 core 600G memory

1 billion data and 10000 files and Query resource: 10 core 40G memory

6.8 Tools

6.8.1 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
      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

6.9 Overview

6.9.1 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.
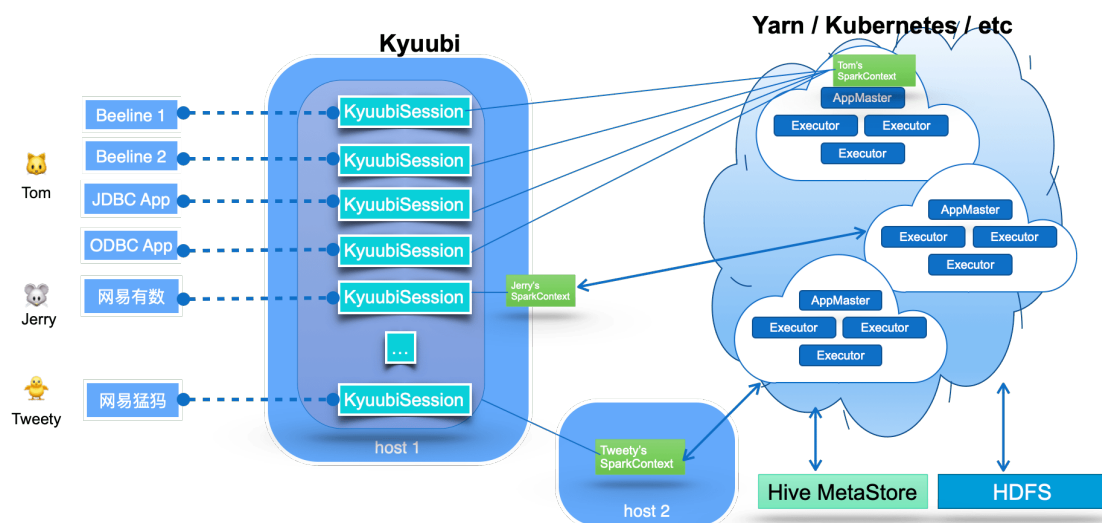
The middle part of the diagram shows the Kyuubi server's main component, which handles the clients' connection and execution requests shown in the left part of the image. Within Kyuubi, these connection requests are maintained as the Kyuubi session's, and execution requests are supported as the Kyuubi 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.
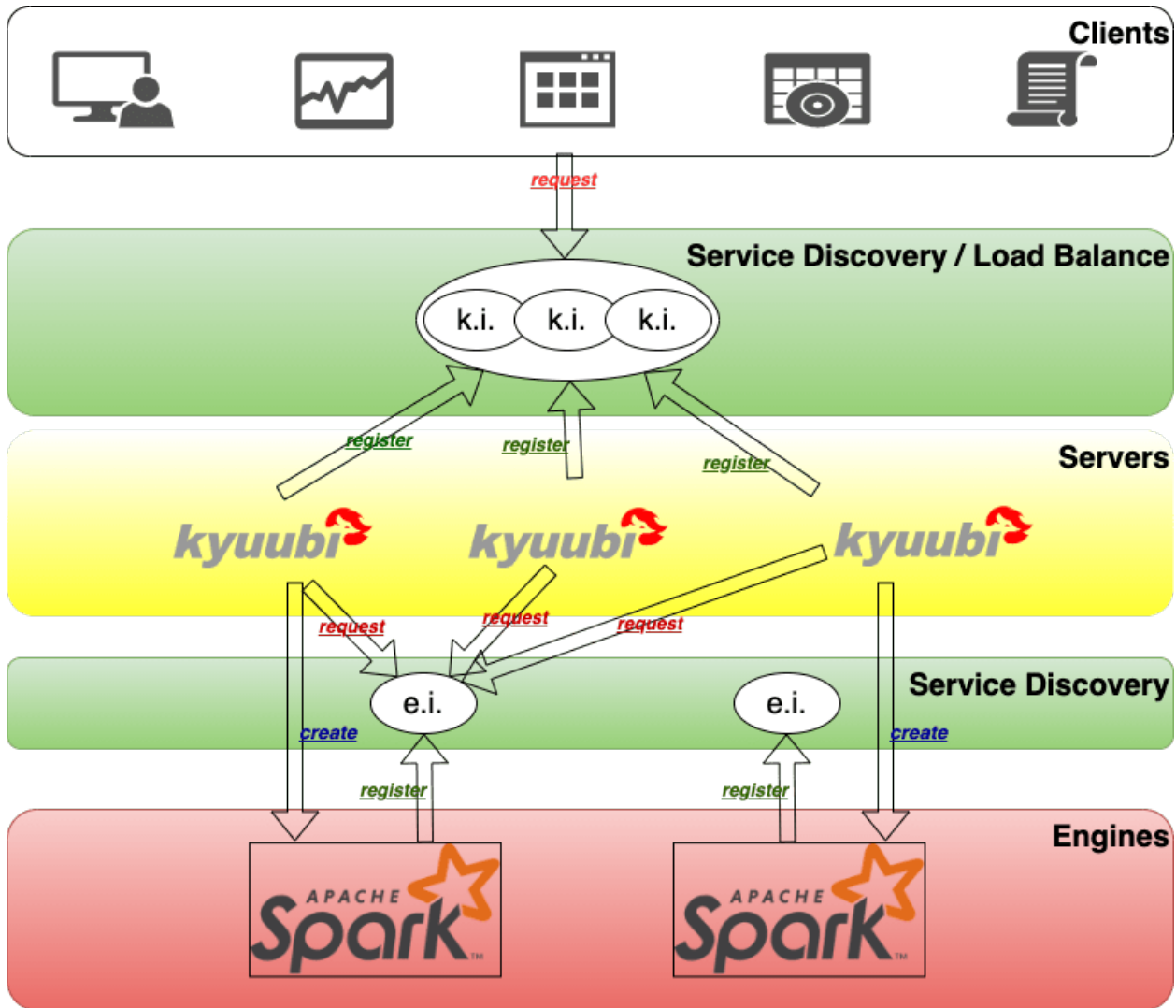
Let us try to break it down from top to bottom based on the above diagram.

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.

6.9.2 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

Performance

References

1. HiveServer2 Overview

6.9.3 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.
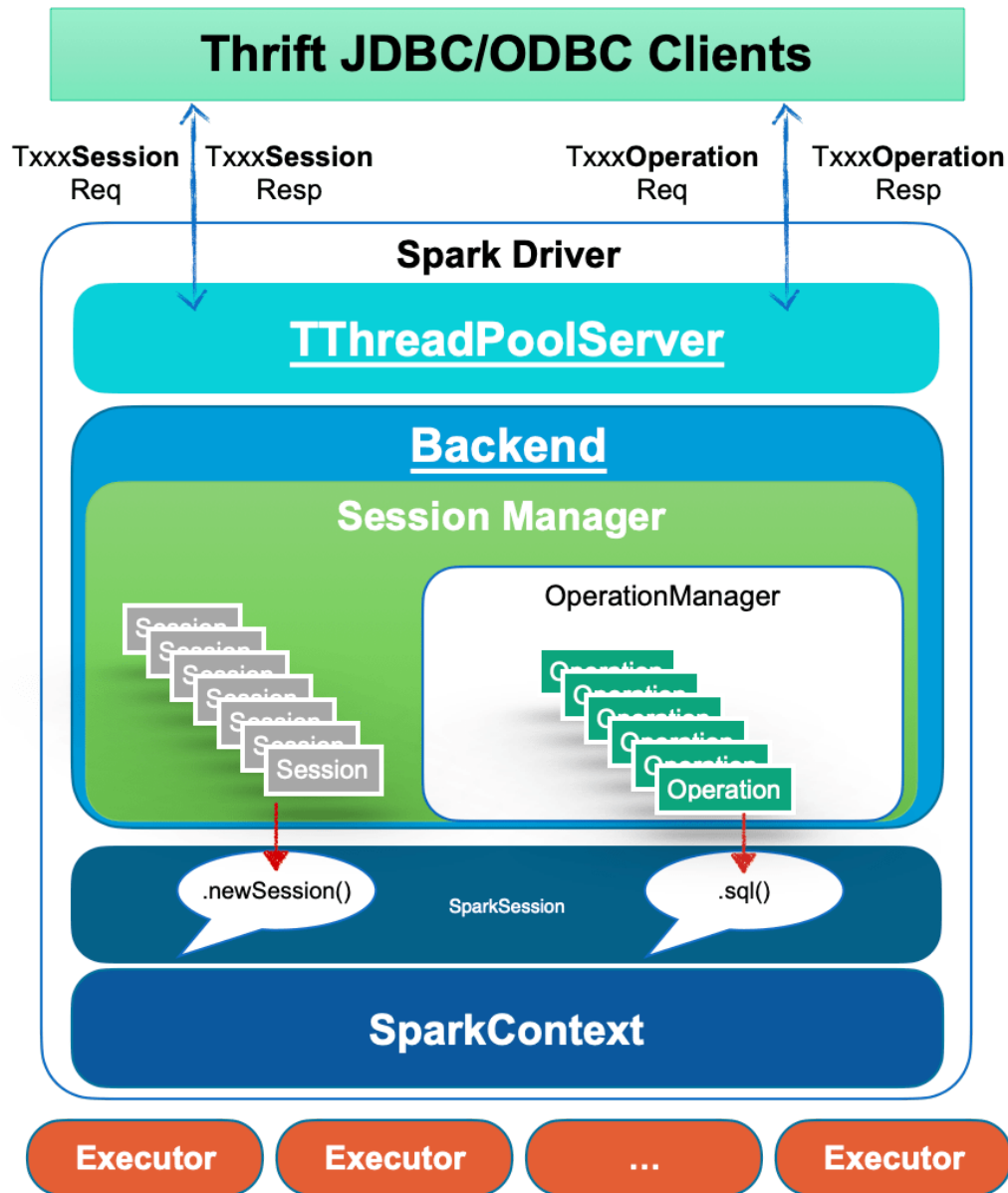
The JDBC connections and operations are handled by the frontend thread pool as various requests. And the corresponding methods of the backend are called to bind to the `SparkSession` related interface. For example, the `DriverManager.getConnection` at the client-side will invoke `SparkSession.newSession()` 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 `Unions`.

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.

6.10 Develop Tools

6.10.1 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,

Defining the Apache Mirror for Spark or Flink

By default, we use `https://archive.apache.org/dist/spark/` to download the built-in Spark or use `https://archive.apache.org/dist/flink/` to download the built-in Flink release package, but if you find it hard to reach, or the downloading speed is too slow, you can define the `spark.archive.mirror` or `flink.archive.mirror` property to a suitable Apache mirror site. For instance,

```
build/mvn clean package -Dspark.archive.mirror=https://mirrors.bfsu.edu.cn/apache/
↪spark/spark-3.0.1
```

Visit Apache Mirrors and choose a mirror based on your region.

Specifically for developers in China mainland, you can use the pre-defined profile named `mirror-cn` which use `mirrors.bfsu.edu.cn` to speed up Spark Binary downloading. For instance,

```
build/mvn clean package -Pmirror-cn
```


6.10.2 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 Server

Usage:
+-----------------------------------------------------------------------------
↪-+
| ./build/dist [--name <custom_name>] [--tgz] [--spark-provided] <maven build options>
↪ |
+-----------------------------------------------------------------------------
↪-+
name:            - custom binary name, using project version if undefined
tgz:             - whether to make a whole bundled package
spark-provided: - whether to make a package without Spark binary
```

For instance,

```
./build/dist --name custom-name --tgz
```

This results in a Kyuubi distribution named `kyuubi-{version}-bin-custom-name.tgz` for you.

If you are planing to deploy Kyuubi where `spark` is provided, in other word, it's not required to bundle spark binary, use

```
./build/dist --tgz --spark-provided
```

Then you will get a Kyuubi distribution without spark binary named `kyuubi-{version}-bin.tgz`.

6.10.3 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 $KTUUBI_HOME/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 html
```

If the build process succeed, the HTML pages are in `_build/html`.

View Locally

Open the `_build/html/index.html` file in your favorite web browser.

6.10.4 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 test
```

Running Tests for a Module

```
./build/mvn clean test -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 test -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
```


6.10.5 Debugging Kyuubi

You can use the Java Debug Wire Protocol to debug Kyuubi with your favorite IDE tool, e.g. IntelliJ IDEA.
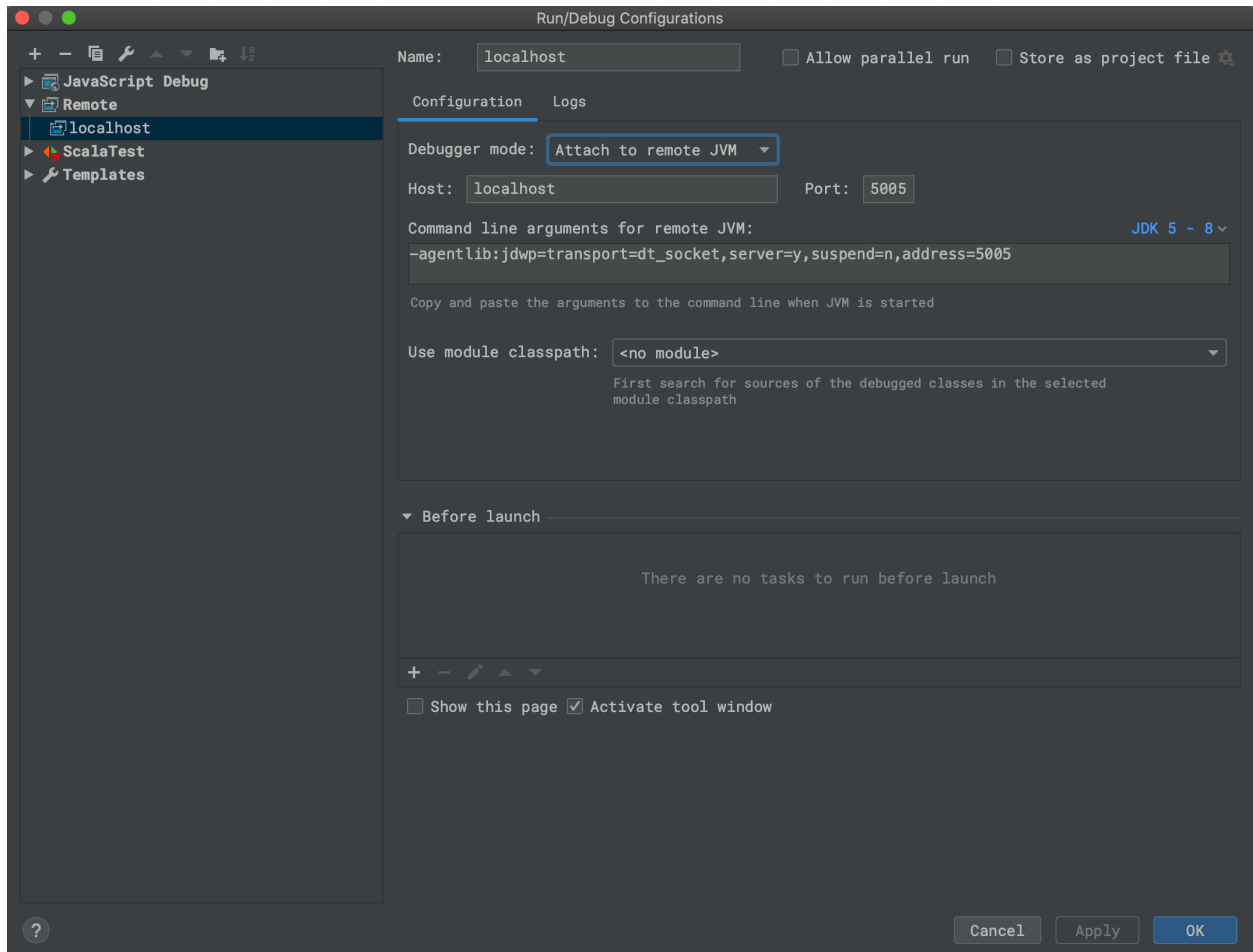
Debugging Server

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 Apps

- 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
```

6.10.6 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.

6.11 Community

6.11.1 Collaborators

PPMC Members and Committers

See full contributor list at contributors.

6.11.2 Contribution Guidelines

Kyuubi is an Apache License v2.0 open source software.

Contributing to Kyuubi including source code, documents, tests e.t.c. means that you agree to the Apache License v2.0.

- Better to search the issue history first before reporting an Issue
- Better to create an Issue to describe the feature or bug first before make a pull request.
- Better to use English for world widely understanding.
- Ask us anything

Before you start, please read the Code of Conduct carefully, familiarize yourself with it and refer to it whenever you need it.

Prepare github environment

If you are new to submit a Pull Request, the follow steps are helpful to you.

1. An available git, you can run `git version` if you are not sure if you have
2. Fork Kyuubi on github, now you have a cloned Kyuubi repo
3. Clone your Kyuubi repo with cmd `git clone https://github.com/${yourname}/kyuubi.git`
4. Create a new branch with cmd `git checkout -b test-branch`
5. Modify the code you want
6. Commit and push code to your Kyuubi repo with command `git commit -am "comment"; git push test-branch test-branch`
7. Back to Kyuubi, you can see a banner about `new pull request`
8. Now we can create a pull request to Kyuubi

Creating a Pull Request

When creating a Pull Request, you will automatically get the template below.

Fulfilling it thoroughly can improve the speed of the review process.

```
<!--
Thanks for sending a pull request!

Here are some tips for you:
  1. If this is your first time, please read our contributor guidelines:
     https://kyuubi.apache.org/docs/latest/community/contributions.html
```

```
-->

### _Which issue are you going to fix?_
<!--
Replace ${ID} below with the actual issue id from
https://github.com/apache/incubator-kyuubi/issues,
so that the issue will be linked and automatically closed after merging
-->

Fixes #${ID}

### _Why are the changes needed?_
<!--
Please clarify why the changes are needed. For instance,
  1. If you add a feature, you can talk about the user case of it.
  2. If you fix a bug, you can clarify why it is a bug.
-->


### _How was this patch tested?_
- [ ] Add some test cases that check the changes thoroughly including negative and␣
↪positive cases if possible

- [ ] Add screenshots for manual tests if appropriate

- [ ] [Run test](https://kyuubi.apache.org/docs/latest/develop_tools/testing.html
↪#running-tests) locally before make a pull request
```

6.11.3 Kyuubi Project Improvement Proposals (KPIP)

The purpose of a KPIP is to inform and involve the user community in significant improvements to the Kyuubi codebase throughout the development process to increase user needs.

KPIPs should be used for significant user-facing or cross-cutting changes, not minor incremental improvements. When in doubt, if a committer thinks a change needs a KPIP, it does.

What is a KPIP?

A KPIP is similar to a product requirement document commonly used in product management.

A KPIP:

- Is a ticket labeled "KPIP" proposing a major improvement or change to Kyuubi

- Follows the template defined below

- Includes discussions on the ticket about the proposal

Who?

Any **community member** can help by discussing whether a KPIP is likely to meet their needs and propose KPIPs.

Contributors can help by discussing whether a KPIP is likely to be technically feasible.

Committers can help by discussing whether a KPIP aligns with long-term project goals, and by shepherding KPIPs.

KPIP Author is any community member who authors a KPIP and is committed to pushing the change through the entire process. KPIP authorship can be transferred.

KPIP Shepherd is a PMC member who is committed to shepherding the proposed change throughout the entire process. Although the shepherd can delegate or work with other committers in the development process, the shepherd is ultimately responsible for the success or failure of the KPIP. Responsibilities of the shepherd include, but are not limited to:

- Be the advocate for the proposed change

- Help push forward on design and achieve consensus among key stakeholders

- Review code changes, making sure the change follows project standards

- Get feedback from users and iterate on the design & implementation

- Uphold the quality of the changes, including verifying whether the changes satisfy the goal of the KPIP and are absent of critical bugs before releasing them

KPIP Process

Proposing a KPIP

Anyone may propose a KPIP, using the document template below. Please only submit a KPIP if you are willing to help, at least with discussion.

If a KPIP is too small or incremental and should have been done through the normal JIRA process, a committer should remove the KPIP label.

KPIP Document Template

A KPIP document is a short document with a few questions, inspired by the Heilmeier Catechism:

- Q1. What are you trying to do? Articulate your objectives using absolutely no jargon.

- Q2. What problem is this proposal NOT designed to solve?

- Q3. How is it done today, and what are the limits of current practice?

- Q4. What is new in your approach, and why do you think it will be successful?

- Q5. Who cares? If you are successful, what difference will it make?

- Q6. What are the risks?

- Q7. How long will it take?

- Q8. What are the mid-term and final "exams" to check for success?

- Appendix A. Proposed API Changes. Optional section defining APIs changes, if any. Backward and forward compatibility must be taken into account.

- Appendix B. Optional Design Sketch: How are the goals going to be accomplished? Give sufficient technical detail to allow a contributor to judge whether it's likely to be feasible. Note that this is not a full design document.

- Appendix C. Optional Rejected Designs: What alternatives were considered? Why were they rejected? If no alternatives have been considered, the problem needs more thought.

Discussing a KPIP

All discussions of a KPIP should take place in a public forum, preferably the discussion attached to the ticket. Any discussion that happen offline should be made available online for the public via meeting notes summarizing the discussions.

Implementing a KPIP

Implementation should take place via the contribution guidelines. Changes that require KPIPs typically also require design documents to be written and reviewed.

6.11.4 Kyuubi Release Guide

Introduction

The Apache Kyuubi (Incubating) 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 **major** 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

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]
```

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/dev/incubator/kyuubi"␣
→work/svn-kyuubi
(gpg --list-sigs "${ASF_USERNAME}@apache.org" && gpg --export --armor "${ASF_USERNAME}
→@apache.org") >> KEYS
svn commit --username "${ASF_USERNAME}" --password "${ASF_PASSWORD}" --message
→"Update KEYS" work/svn-kyuubi
```

Cut branch iff for major release

Kyuubi use version pattern `{MAJOR_VERSION}.{MINOR_VERSION}.{PATCH_VERSION}[-{OPTIONAL_SUFFIX}]`, e.g. `1.3.0-incubating`. **Major Release** means `MAJOR_VERSION` or `MINOR_VERSION` changed, and **Patch Release** means `PATCH_VERSION` changed.

The main step towards preparing a major 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.

The release branch pattern is `branch-{MAJOR_VERSION}.{MINOR_VERSION}`, e.g. `branch-1.3`.

After cutting release branch, don't forget bump version in `master` branch.

Build a release candidate

1. Set environment variables.

```
export RELEASE_VERSION=<release version, e.g. 1.3.0-incubating>
export RELEASE_RC_NO=<RC number, e.g. 0>
```

1. Bump version.

```
build/mvn versions:set -DgenerateBackupPoms=false \
  -DnewVersion="${RELEASE_VERSION}" \
  -Pspark-3.1,spark-block-cleaner

git commit -am "[RELEASE] Bump ${RELEASE_VERSION}"
```

1. Create a git tag for the release candidate.

The tag pattern is `v${RELEASE_VERSION}-rc${RELEASE_RC_NO}`, e.g. `v1.3.0-incubating-rc0`

1. Package the release binaries & sources, and upload them to the Apache staging SVN repo. Publish jars to the Apache staging Maven repo.

```
build/release/release.sh publish
```

Vote on the release candidate

The release voting takes place on the Apache Kyuubi (Incubating) developers list (the (P)PMC is voting).

- 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

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] …**.

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 (P)PMC members can move to release directory)

Move the sub-directory 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/incubator/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 sync'd to Maven Central automatically.

Promote the release

Update Website

Fork and clone Apache Kyuubi website

1. Add a new markdown file in `src/news/`

2. Add a new markdown file in `src/release/`

3. Follow *Build Document* to build documents, then copy `apache/incubator-kyuubi`'s folder `docs/_build/html` to `apache/incubator-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. The mailing list includes: `announce@apache.org`, `dev@kyuubi.apache.org`, `user@spark.apache.org`.

Enjoy an adult beverage of your choice, and congratulations on making a Kyuubi release.

6.11.5 Community

Watch Star Fork Issue Download

6.12 Appendixes

6.12.1 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 highlevel 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).