[Kyuubi] Kyuubi MeetUP

Created by Li Xieming[李燮鳴], last modified on Mar 15, 2023

✓ DPHADOOP-4296 - Kyuubi Meetup **RESOLVED**

Spark SQL and Spark Thrift Server (STS)

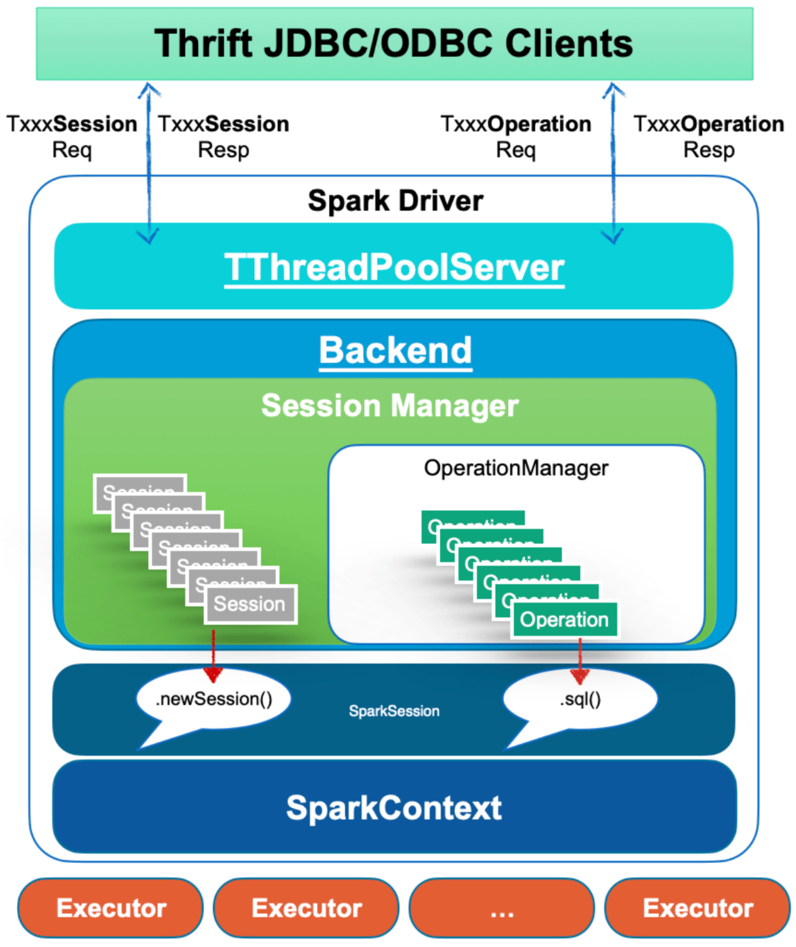
In the past, users have to understand the Spark RDD model and the various operators in order to use Spark effectively for data processing and analysis. However, because too many low-level implementation details are exposed to users, Spark has a high learning curve and is not very user-friendly for many novice users.

The SparkSQL has resolved the issue by offering a simple SQL interface, users with basic programming skills and knowledge of SQL can leverage Spark's powerful fast distributed computing capabilities to process and analyze their large-scale datasets.

Spark Thrift Server has further improved the usability of SparkSQL. By providing a standard JDBC interface, data engineers can quickly connect to Spark Thrift Server with ease.

As the name suggests, Spark ThriftServer is essentially an Apache Thrift server that includes a SparkContext and can execute Spark applications. Also, because it's a server, it has to be a long-running application, unlike the normal Spark Application which will finish after the processing is finished.

This is the Architecture of SparkSQL.



As

When a user executes an SQL statement through JDBC or beeline, TThreadPoolServer receives the SQL statement, 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(...)`***.

Limitation of Spark Thrift Server

The bottleneck of Spark Thrift Server comes from its architecture design – a single Spark Context for all sessions from all users.

The main problems are:

- **Resource Isolation / Noisy Neighbours**. All the sessions have to share one Spark Driver, hence the resource attached to the driver.

This will cause many issues such as,

  - All the sessions will share a single client connection to Hive Metastore, which will very likely be a bottleneck in a larger environment.
  - One job could suffer starvation when its neighbor is requesting a lot of resources.
  - Some malicious/faulty UDF could stop the whole Spark context unexpectedly. (e.g having exit(0) in the code)

- **Multi-tenancy Limitation.** Different users are able to connect to the Spark Thrift Server. However, from the resource management framework (K8s, YARN, MESOS)'s perspective,
  all the sessions are running under the user who started SparkContext. This limitation basically means there will be no native resource management based on different users
  - Can't set resource quota based on user.
  - Can't have resource accounting based on users.
  - Can't configure different queues for different applications

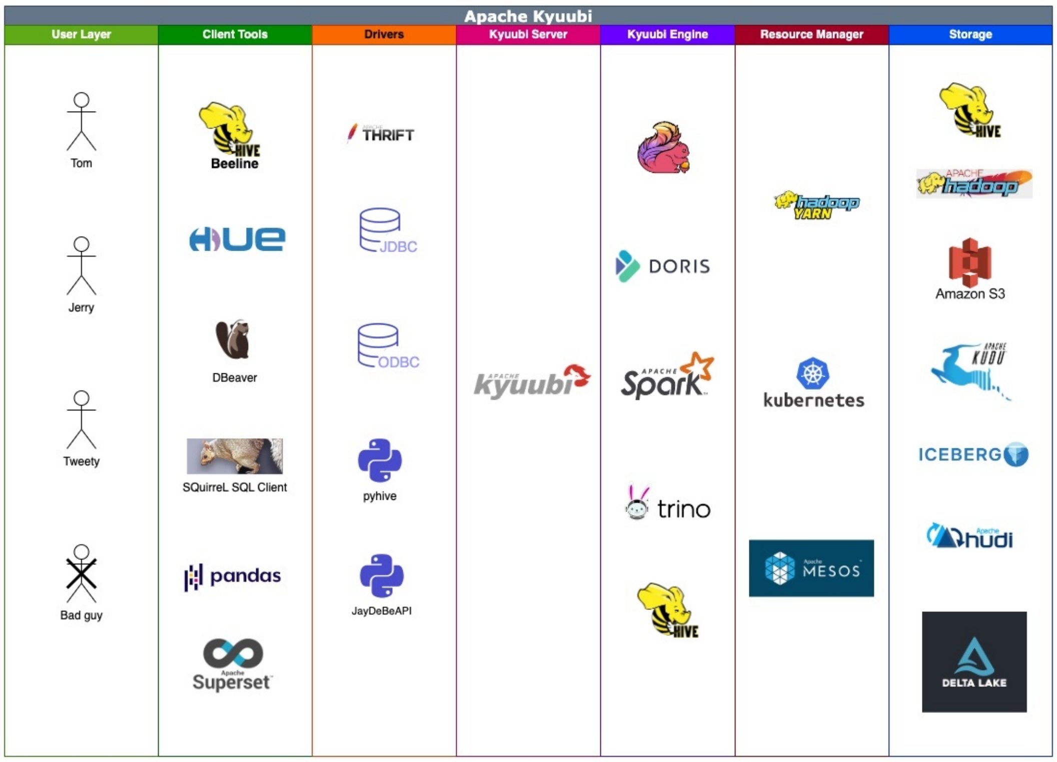
Due to these issues and limitations, Spark Thrift Server is not a solid choice for enterprises with large clusters.

What is Kyuubi

Apache Kyuubi is an open-source project initiated by NetEase.

It was accepted as an Apache Incubator project in Jun 2021 and promoted to a top-level Apache project on 2023 Jan 31.

This is a picture copied from Kyuubi's Official Site.



As you can see, Kyuubi is a Gateway that supports multiple client/driver and backend engines.

In LINE, beeline(thrift), JDBC, pyhive are used as clients to connect to the backend engines such as Trino, Spark, and Hive.
These accesses can all be unified with Kyuubi.

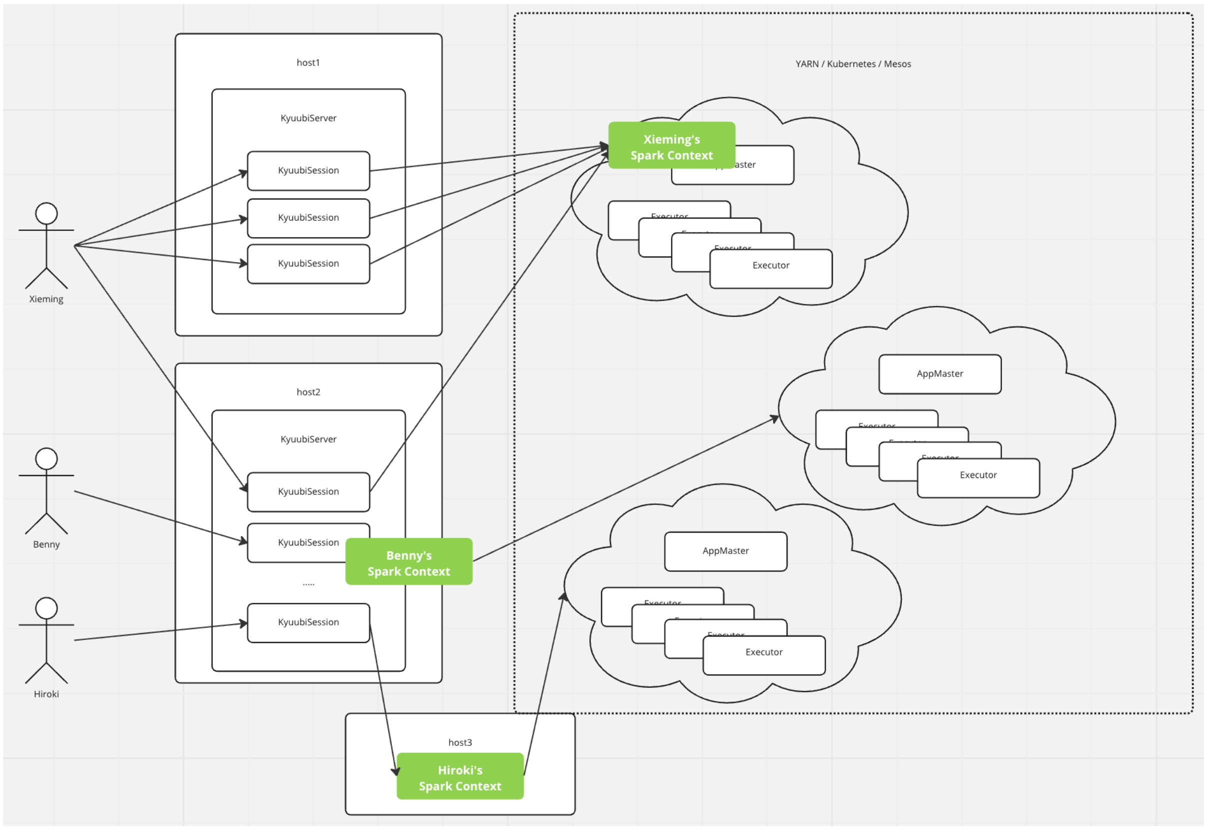
As of now, we are investigating Kyuubi because some of the HiveServer workloads will be migrated to Kyuubi + SparkSQL

In this session, we will mainly discuss Kyuubi + Spark + YARN (or K8S)

Kyuubi vs Spark Thrift Server (STS)

Kyuubi has resolved the main paint points of the Spark Thrift Server.

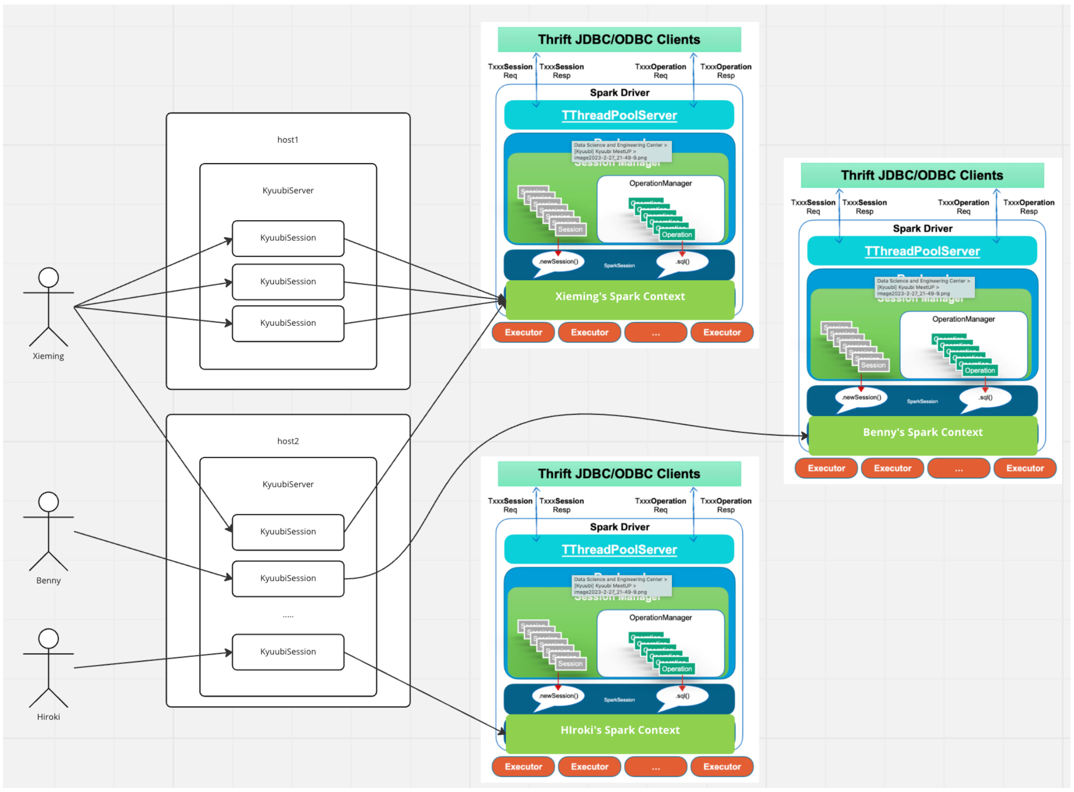
This is the main architecture of Kyuubi.



By the default configuration (engine share level: USER), Kyuubi will submit a Spark Context for each user on the user's behave.

- A user on the left side connects to Kyuubi Server with JDBC / Beeline or other supported clients.
- Upon the connection, Kyuubi Server will create a KyuubiSession, and try to find an existing Spark Context.
  - If an existing Spark Context is already initiated, KyuubiSession will connect to it. (Like Xieming's case in the graph)
  - If not, Kyuubi Server will submit a new SparkContext and then connect to it. (Like Benny's case in the graph)

Although **it's NOT entirely precise**, allowing each user to have their own Spark Context is **similar** to having Spark Thrift Server for each user. And this will tackle the challenges



- **Resource Isolation / Noisy Neighbours**.
  - Different users are using a different Spark Context. Therefore, there will be no contention in Spark Driver from the different users at all.
- **Multi-tenancy Limitation.**
  - Since the user's Spark Context is submitted under the name of the user himself, therefore, it's easy to set resource quota, set queue permission, and have resource accounting based on the user.

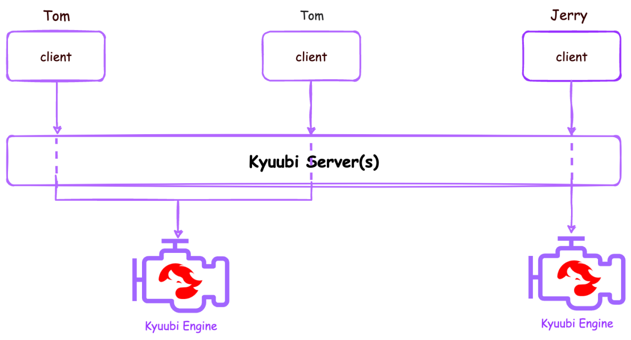
Kyuubi Engine Share Level

REF: https://kyuubi.readthedocs.io/en/master/deployment/engine_share_level.html

In the context of Kyuubi, the Spark Context that serves the real workload is called SparkSQLEngine.
You can define how you want to share the SparkContext with Kyuubi.

USER Share Level

In the previous section, we can see that each of the users has one Spark Engine, and that's the **USER** share level.
This is also the default share level.



When connections are initiated by the same user (in this graph: Tom),
those sessions share the same engine with objects belonging to a same SparkContext instance,
including Classes/Classloaders, SparkConf, Driver/Executors, Hive Metastore Client, etc.

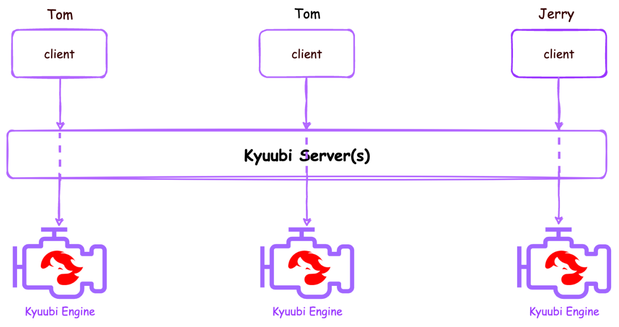
Initially, it takes some time to start the SparkSQLEngine. However, once the engine is started,
all the following connections and sessions will reuse the engine and it will respond faster.

By default, each session can still have its own SparkSession instance, which contains a separate session state,
including temporary views, SQL config, UDFs, etc.

Setting *kyuubi.engine.single.spark.session* to true will make SparkSession the instance a singleton and share across sessions.
To be specific, When *kyuubi.engine.single.spark.session=true*, a user can access the temporary views (similar to Hive Temporary Table) created by other beeline connections.

Finally, when all the sessions are closed, Kyuubi will wait for a certain period of time (defined by *kyuubi.session.engine.idle.timeout*), and if not further connections are made by the user,
the SparkSQLEngine will be shut down.

CONNECTION Share Level



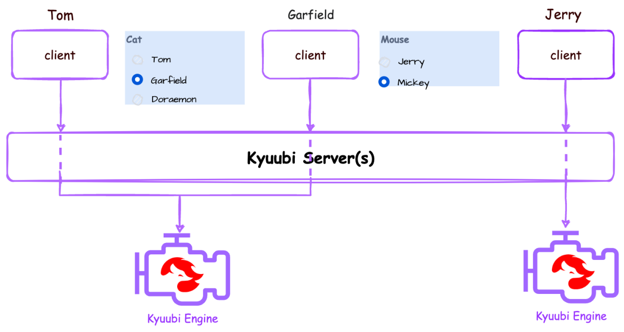
Each session with CONNECTION share level has a standalone engine for itself which is unreachable for anyone else.

On the good side, it has the best resource isolation because you are not sharing resources even with the other application submitted by yourself,
However, that means you need to wait for SparkSQLEngine to start every time you initiate a session.

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

This share level is closest to the implementation of HiveServer.

GROUP Share Level



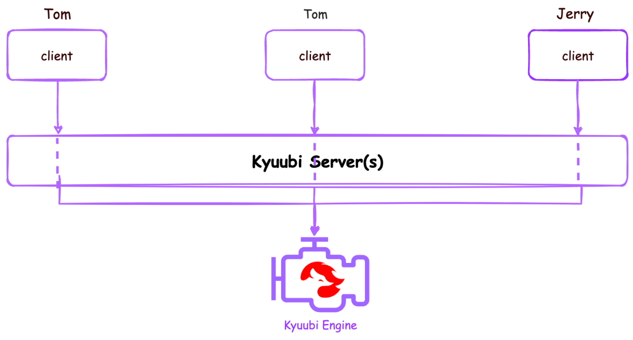
In Group Share Level, an engine will be shared by all sessions created by all users belonging to the same primary group name.

The engine will be launched by the group name as the effective username, so here the group name is considered as a special user who is able to visit the computing resources/data of a team.

It follows the Hadoop GroupsMapping to map a 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 the USER share level.

SERVER Share Level



In the SERVER share level, all the user sessions will use the exact same Spark Context.

Subdomain

or 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, when you connect to the server, you can specify Subdomain.

beeline -u 'jdbc:hive2://kyuubi-server.linecorp.com:10009/default?socketTimeout=60000;#**kyuubi.engine.share.level.subdomain=subdomain1**' -n jpz3032 -p

If you do so, the backend engine will be shared with the session that also set **kyuubi.engine.share.level.subdomain=subdomain1**.

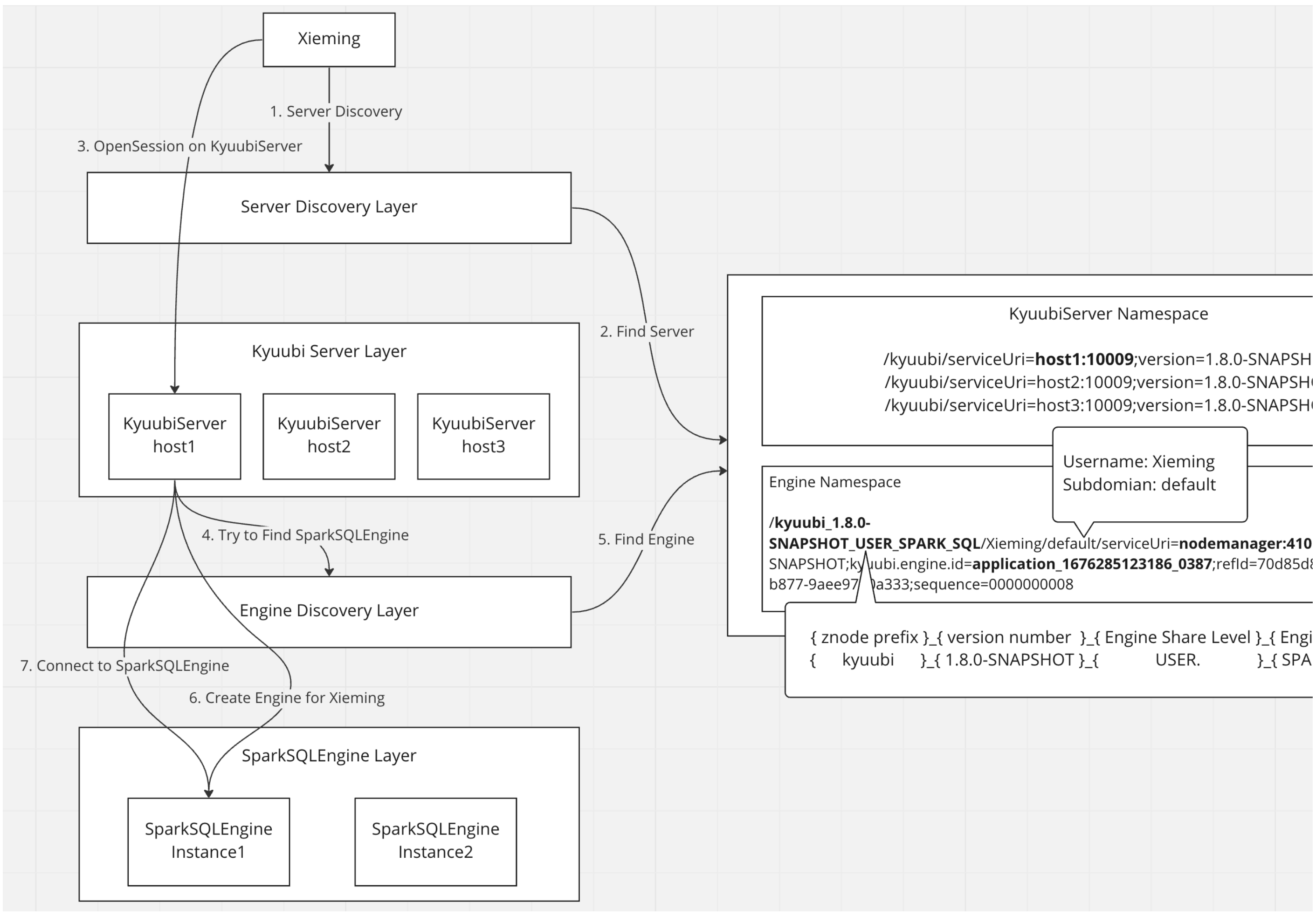
Share Level Recommendation

Share Level	Syntax	Scenario	Isolation Degree	Shareability
CONNECTION	One engine per session	Large-scale ETL Ad hoc	High	Low
USER	One engine per user	Ad hoc Small-scale ETL	Medium	Medium
GROUP	One engine per primary group	Ad hoc Small-scale ETL	Low	High
SERVER	One engine per cluster	Admin	Highest If Secured Lowest If Unsecured	Admin ONLY If Secured

- Better isolation degree of engines gives us better stability of an engine and the query executions running on it.

- Better shareability of engines means we are more likely to reuse an engine which is already in full speed.

General Kyuubi Exection Flow


Xieming
1. Server Discovery
3. OpenSession on KyuubiServer
Server Discovery Layer
2. Find Server
Kyuubi Server Layer
KyuubiServer host1
KyuubiServer host2
KyuubiServer host3
4. Try to Find SparkSQLEngine
5. Find Engine
Engine Discovery Layer
7. Connect to SparkSQLEngine
6. Create Engine for Xieming
SparkSQLEngine Layer
SparkSQLEngine Instance1
SparkSQLEngine Instance2
KyuubiServer Namespace
/kyuubi/serviceUri=host1:10009;version=1.8.0-SNAPSH
/kyuubi/serviceUri=host2:10009;version=1.8.0-SNAPSH
/kyuubi/serviceUri=host3:10009;version=1.8.0-SNAPSH
Engine Namespace
/kyuubi_1.8.0-SNAPSHOT_USER_SPARK_SQL/Xieming/default/serviceUri=nodemanager:410
SNAPSHOT;kyuubi.engine.id=application_1676285123186_0387;refId=70d85d
b877-9aee97 a333;sequence=0000000008
Username: Xieming
Subdomian: default
{ znode prefix }_{ version number }_{ Engine Share Level }_{ Engi
{ kyuubi }_{ 1.8.0-SNAPSHOT }_{ USER. }_{ SPA

Kyuubi Spark Engine Extensions

The relationship between **Kyuubi Spark Engine Extensions** and **Kyuubi Server** is not actually strong,
in the sense that the Kyuubi Spark Engine Extensions can be applied to Spark Engine even **without** using Kyuubi Server.

However, some extensions only make sense when used in conjunction with Kyuubi Server.

According to the latest reference at this moment, Kyuubi currently supports the following extensions.

- Z-Ordering Support
- **Auxiliary Optimization Rules**
- **Kyuubi Spark AuthZ Plugin**
- Auxiliary SQL Functions
- Connectors for Spark SQL Query Engine
- SQL Lineage Support
- Hive Dialect Support

Here, we take two examples to show why "some extensions only make sense when used in conjunction with Kyuubi Server."

Fine-Grained Control with Kyuubi Authz Extension

This part is excerpted from [iucdpdev] Kyuubi On-Premise

[iucdpdev] Kyuubi On-Premise

Part 2: Set Up Spark with Row Level Filter

In this test, we will use Kyuubi Authz Plugin to Implement Row Level Fitlering.

1. Download Authz jar to Spark's jar directory

```
sudo wget --directory-prefix=/opt/spark/jars/ https://repo1.maven.org/maven2/org/apache/kyuubi/kyuubi-spark-authz_2.
```

2. Add Additional Jars to spark-defaults.conf and Update the Spark configuration to enable the authz filter

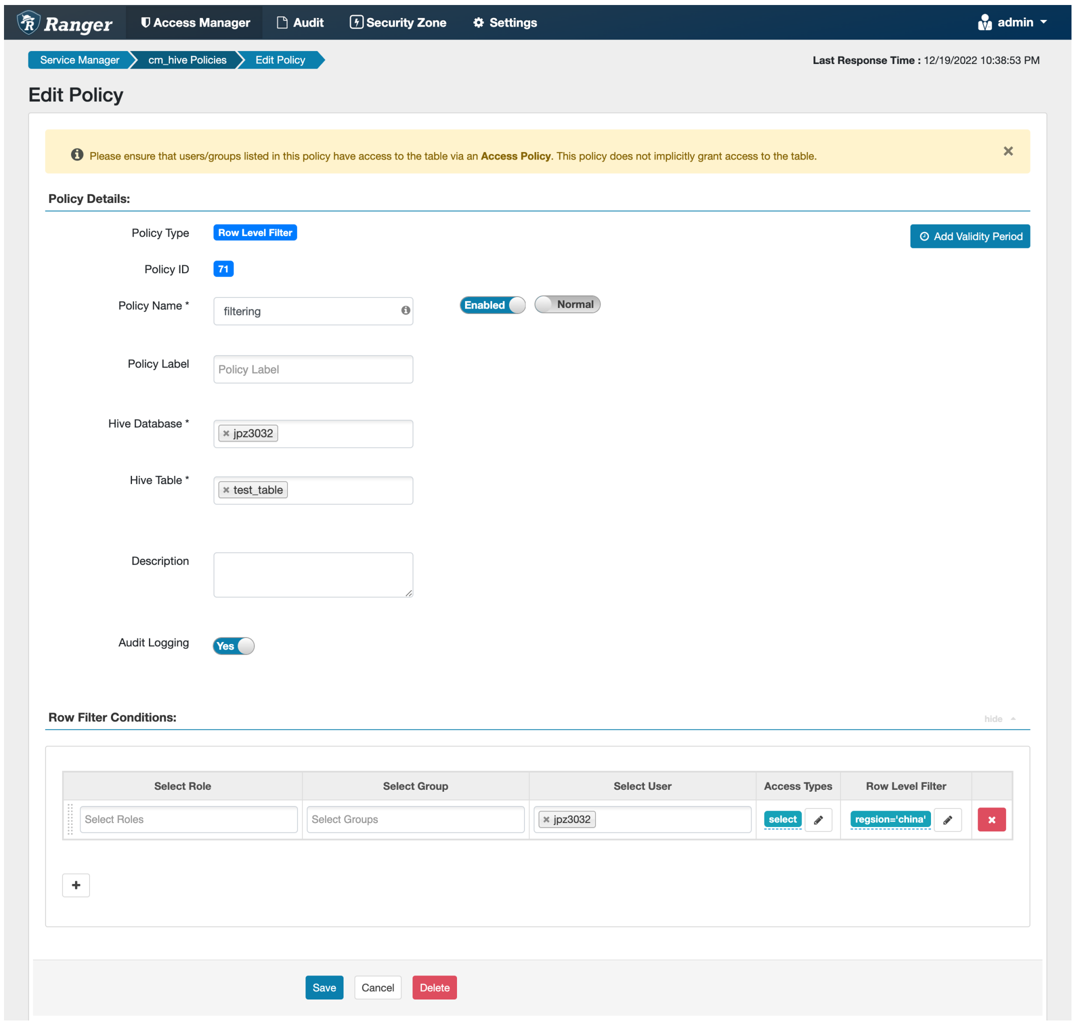
```
# Additional Dependencies Required for Row Level Filter

ADD_JARS=$(cat << _EOF_ | xargs -I{} sh -c 'ls {}*' | tr '\n' ':' | sed 's/:$//'
/opt/cloudera/parcels/CDH/lib/ranger-hive-plugin/lib/ranger-hive-plugin-impl/ranger-plugins-common-
/opt/cloudera/parcels/CDH/lib/ranger-hive-plugin/lib/ranger-hive-plugin-impl/ranger-plugins-audit-
/opt/cloudera/parcels/CDH/lib/ranger-atlas-plugin/lib/ranger-atlas-plugin-impl/jersey-bundle-
/opt/cloudera/parcels/CDH/lib/ranger-admin/ews/lib/gethostname4j-
/opt/cloudera/parcels/CDH/lib/ranger-admin/ews/lib/jna-5
/opt/cloudera/parcels/CDH/lib/ranger-admin/ews/lib/ranger-plugins-cred-
/opt/cloudera/parcels/CDH/lib/ranger-atlas-plugin/lib/ranger-atlas-plugin-impl/jackson-jaxrs-
_EOF_
)

# Check the format just in case
echo $ADD_JARS

# Add Jars to Configuraton Path
cat << _EOF_  | sudo tee -a /opt/spark/conf/spark-defaults.conf
spark.sql.extensions=org.apache.kyuubi.plugin.spark.authz.ranger.RangerSparkExtension
spark.driver.extraClassPath=${ADD_JARS:?}
spark.executor.extraClassPath=${ADD_JARS:?}
_EOF_
```

```
cat << '_EOF_' | sudo tee /opt/spark/conf/ranger-spark-security.xml
<configuration>
    <property>
        <name>ranger.plugin.spark.policy.rest.url</name>
        <value>http://iucdpdev-master001-iu-dev-jp2v-prod.lineinfra.com:6080</value>
    </property>

    <property>
        <name>ranger.plugin.spark.service.name</name>
        <value>cm_hive</value>
    </property>

    <property>
        <name>ranger.plugin.spark.policy.cache.dir</name>
        <value>/tmp/ranger_spark_policy</value>
    </property>

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

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

3. Test the Row-Level Filtering

Configuring Row-Level Filtering in Ranger.

In this test, we configure jpz3032 to access the record with **regsion='china'**.



Last Response Time : 12/19/2022 10:38:53 PM

Edit Policy

Please ensure that users/groups listed in this policy have access to the table via an **Access Policy**. This policy does not implicitly grant access to the table.

Policy Details:

Policy Type	Row Level Filter
Policy ID	71
Policy Name *	filtering (Enabled, Normal)
Policy Label	Policy Label
Hive Database *	jpz3032
Hive Table *	test_table
Description	
Audit Logging	Yes

Row Filter Conditions:

Select Role	Select Group	Select User	Access Types	Row Level Filter
Select Roles	Select Groups	jpz3032	select	regsion='china'

After this setting, you can confirm that "jpz3032" can only access the data labeled "regsion='china'".

```
[irteamsu@xieming-cdp-client001-iu-dev-jp2v-prod ~]$ klist
Default principal: jpz3032@IUZ.ITSC.COM

...

[irteamsu@xieming-cdp-client001-iu-dev-jp2v-prod ~]$ spark-sql
Setting default log level to "WARN".

...

Spark master: yarn, Application Id: application_1670833586410_0016
spark-sql> select * from jpz3032.test_table;
22/12/19 22:32:58 WARN session.SessionState: METASTORE_FILTER_HOOK will be ignored, since hive.security.authorizati
china   1
china   2
china   3
Time taken: 8.921 seconds, Fetched 3 row(s)
```

On the other hand, if you access the same table with iu_admin, then you can access all the users.

```

[irteamsu@xieming-cdp-client001-iu-dev-jp2v-prod ~]$ spark-sql
Setting default log level to "WARN".

...

Spark master: yarn, Application Id: application_1670833586410_0017
spark-sql> select * from jpz3032.test_table;
22/12/19 22:49:05 WARN session.SessionState: METASTORE_FILTER_HOOK will be ignored, since hive.security.authori
22/12/19 22:49:06 WARN ipc.Client: Exception encountered while connecting to the server : org.apache.hadoop.ipc.
china   1
china   2
china   3
japan   4
japan   5
japan   6
japan   7
Time taken: 8.861 seconds, Fetched 7 row(s)
```

As you can see from the above steps, to enable the Authz extension, all you need to do is

1. download the jar.
2. **upgrade the Spark config.**

However, in the normal Spark case, a user can change the "spark-default.conf" located on their local machine at */opt/spark/conf/spark-default.conf*. E.g. User can just delete the line of "spark.sql.extensions=org.apache.kyuubi.plugin.spark.authz.ranger.RangerSparkExtension", and disable the plugin. Therefore, there is no way to enforce the use of the Authz extension.

This situation can be improved by using Kyuubi Server:

1. set *spark.sql.extensions=org.apache.kyuubi.plugin.spark.authz.ranger.RangerSparkExtension* in the *spark-default.conf* on the server side.
2. add *spark.sql.extensions* to *kyuubi.session.conf.ignore.list* or *kyuubi.session.conf.restrict.list*.
    a. The format will ignore the *spark.sql.extensions* set on the client side, and the latter will reject a session if the *spark.sql.extensions* is set.

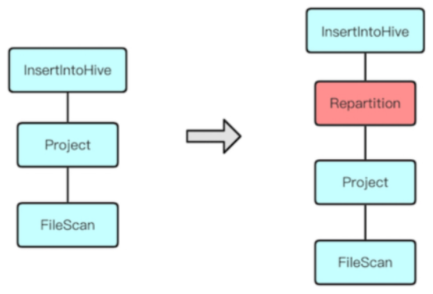
So this is an example of "extensions only make sense when used in conjunction with Kyuubi Server"

Auxiliary Optimization Rules

Auxiliary Optimization Rules is an example where a Spark Extension could be used without the kyuubi server.

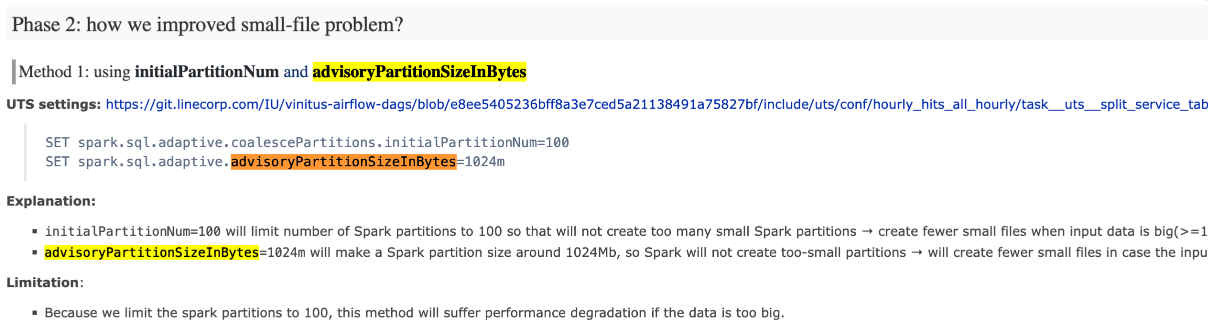
"Auxiliary Optimization Rules" is a plugin that can extend Spark in a non-intrusive way, and it includes three main features.

- Merging small files automatically
    - e.g: spark.sql.optimizer.insertRepartitionBeforeWrite.enabled
        - When this is set to enable, an additional shuffle will be introduced to repartition the output.
          → In the final stage where DataFrame is being inserted into a Hive Table, if the partition distribution doesn't match that of Hive partition, then each Spark tasks have to write multiple Hive partitions
          → This will generate tons of small files
        - Enabling RepartitionBeforeWrite will add the repartition operator to make sure the DataFrame partition matches the Hive Partition to avoid small file problem.



- Insert shuffle node before Join to make AQE OptimizeSkewedJoin
- Stage level config isolation in AQE
    - Allowing setting different Final stage
    - When spark.sql.optimizer.finalStageConfigIsolation.enabled is set to true, when can modify `spark.sql.adaptive.advisoryPartitionSizeInBytes` for different stages:
        - `spark.sql.adaptive.advisoryPartitionSizeInBytes=64MB or smaller for better parallelism`
        - `spark.sql.finalStage.adaptive.advisoryPartitionSizeInBytes=256MB or bigger to avoid small file issue.`

- `This setting may resolve the issue summerized by Uchida-san in a previous meetup`



Phase 2: how we improved small-file problem?

Method 1: using **initialPartitionNum** and **advisoryPartitionSizeInBytes**

UTS settings: https://git.linecorp.com/IU/vinitus-airflow-dags/blob/e8ee5405236bff8a3e7ced5a21138491a75827bf/include/uts/conf/hourly_hits_all_hourly/task__uts__split_service_tab

```
SET spark.sql.adaptive.coalescePartitions.initialPartitionNum=100
SET spark.sql.adaptive.advisoryPartitionSizeInBytes=1024m
```

Explanation:

- `initialPartitionNum=100` will limit number of Spark partitions to 100 so that will not create too many small Spark partitions → create fewer small files when input data is big(>=1
- `advisoryPartitionSizeInBytes=1024m` will make a Spark partition size around 1024Mb, so Spark will not create too-small partitions → will create fewer small files in case the inpu

Limitation:

- Because we limit the spark partitions to 100, this method will suffer performance degradation if the data is too big.

- `This setting may resolve the issue summerized by Uchida-san in a previous meetup`

Just like how we enable Authz for Spark, we can also enable "Auxiliary Optimization Rules" by setting *spark.sql.extensions=org.apache.kyuubi.sql.KyuubiSparkSQLExtension* in the *spark-default.conf*.

In this case, because "Auxiliary Optimization Rules" is an optimization, it doesn't need to be enforced, at least from a security perspective.
Users can choose to enable this function for better performance, or they can disable it until they finish the test and are confident with the optimization.

Of course, if the consistency of "Auxiliary Optimization Rules" is widely recognized, we can set this option to Kyuubi Server, and allow all the Spark queries benefit from the plugin.

Issues Found During the Verification

- Authz doesn't work well with Spark Cluster Mode.
  - → RangerAdmin Server doesn't support DelegationToken + Kyuubi doesn't support Keytab authentication with Cluster Mode
  - → Can be workarounded by using simple authentication (need to change the Ranger Plugin code), https://github.com/apache/kyuubi/discussions/3620

See: [iucdpdev] Kyuubi on Kubernetes (Optimized),

- Failed SparkContext (Application Master) will not be retried.
  - In the case of HiveServer, when ApplicationMaster is killed, YARN will retry for three times by default.
  - KyuubiServer cannot do this, and nor can it be implemented easily.
    → because it doesn't hold the query itself in its data structure.
- Failed SparkContext will cause the beeline to hang.
  - This should be able to be implemented. → Trying to do this now

See: [kyuubi] Kyuubi Fault Tolerance Test

Deploying Kyuubi Server

- [iucdpdev] Kyuubi on Kubernetes (Optimized)
- [iucdpdev] Kyuubi On-Premise

Kyuubi Implementation Deep Dive

- https://miro.com/app/board/uXjVPkwxaxY=/
- [Kyuubi] Source Code Analysis

No labels