Solutions (https://cloud.google.com/solutions/) Solutions

Using Apache Hive on Cloud Dataproc

This tutorial shows how to use Apache Hive (https://hive.apache.org/) on Dataproc (https://cloud.google.com/dataproc/) in an efficient and flexible way by storing Hive data in Cloud Storage (https://cloud.google.com/storage/) and hosting the Hive metastore in a MySQL (https://www.mysql.com/) database on Cloud SQL (https://cloud.google.com/sql/). This separation between compute and storage resources offers some advantages:

- Flexibility and agility: You can tailor cluster configurations (https://cloud.google.com/dataproc/docs/concepts#configuring-clusters) for specific Hive workloads and scale each cluster independently up and down as needed.
- Cost savings: You can spin up an ephemeral cluster (https://cloud.google.com/solutions/migration/hadoop/hadoop-gcp-migration-overview#moving_to_an_ephemeral_model) when you need to run a Hive job and then delete it when the job completes. The resources that your jobs require are active only when they're being used, so you pay only for what you use. You can also use preemptible VMs (https://cloud.google.com/dataproc/docs/concepts/compute/preemptible-vms) for noncritical data processing or to create very large clusters at a lower total cost.

Hive is a popular open source data warehouse system built on Apache Hadoop (https://hadoop.apache.org/). Hive offers a SQL-like query language called HiveQL (https://wikipedia.org/wiki/Apache_Hive#HiveQL), which is used to analyze large, structured datasets. The Hive metastore holds metadata about Hive tables, such as their schema and location. Where MySQL is commonly used as a backend for the Hive metastore, Cloud SQL makes it easy to set up, maintain, manage, and administer your relational databases on Google Cloud.

Dataproc is a fast, easy-to-use, fully managed service on Google Cloud for running Apache Spark (https://spark.apache.org/) and Apache Hadoop (https://hadoop.apache.org/) workloads in a

simple, cost-efficient way. Even though Dataproc instances can remain stateless, we recommend persisting the Hive data in Cloud Storage and the Hive metastore in MySQL on Cloud SQL.

Objectives

- Create a MySQL instance on Cloud SQL for the Hive metastore.
- Deploy Hive servers on Dataproc.
- Install the Cloud SQL Proxy (https://cloud.google.com/sql/docs/mysql/sql-proxy) on the Dataproc cluster instances.
- Upload Hive data to Cloud Storage.
- Run Hive queries on multiple Dataproc clusters.

Costs

This tutorial uses the following billable components of Google Cloud:

- Dataproc
- Cloud Storage
- Cloud SQL

You can use the pricing calculator (https://cloud.google.com/products/calculator) to generate a cost estimate based on your projected usage.

New GCP users might be eligible for a free trial (https://cloud.google.com/free-trial).

Before you begin

Create a new project

1. In the Cloud Console, go to the project selector page.

   GO TO THE PROJECT SELECTOR PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/PROJECTSELECT

2. Select or create a Cloud project.

Enable billing

- Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (https://cloud.google.com/billing/docs/how-to/modify-project).

Initialize the environment

1. Start a Cloud Shell instance:

   GO TO CLOUD SHELL (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/HOME/DASHBOARD?CLOUDSHELL

2. In Cloud Shell, set the default Compute Engine zone to the zone where you are going to create your Dataproc clusters. This tutorial uses the `us-central1-a` zone in the `us-central1` region.

```
export REGION=us-central1
export ZONE=us-central1-a
gcloud config set compute/zone $ZONE
```

3. Enable the Dataproc and Cloud SQL Admin APIs by running this command in Cloud Shell:

```
gcloud services enable dataproc.googleapis.com sqladmin.googleapis.com
```

Reference architecture

For simplicity, in this tutorial you deploy all compute and storage services in the same Google Cloud region (https://cloud.google.com/compute/docs/regions-zones/) to minimize network latency and network transport costs. Figure 1 presents the architecture for this tutorial.

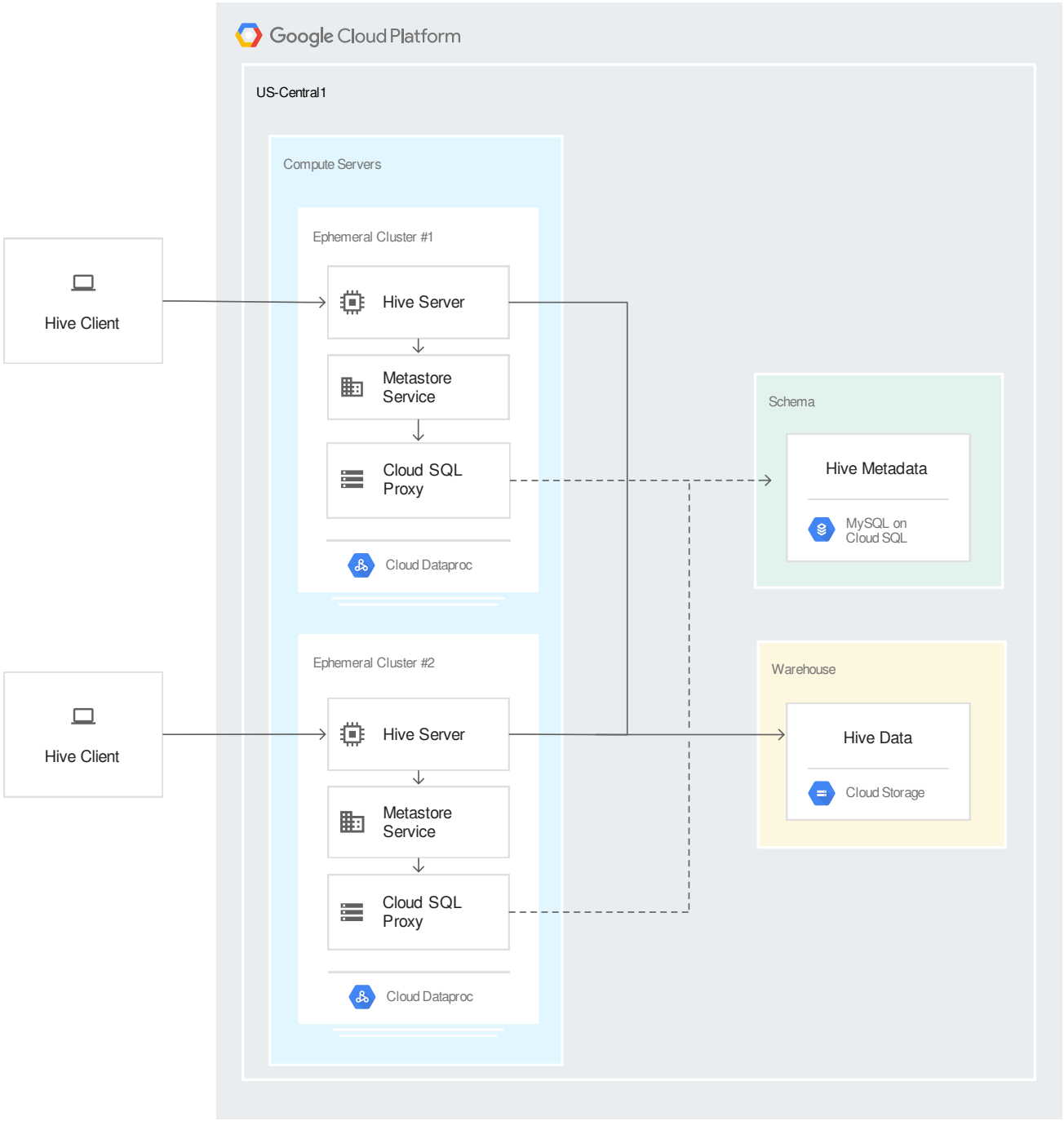

Figure 1. Example of a single-region Hive architecture

With this architecture, the lifecycle of a Hive query follows these steps:

1. The Hive client submits a query to a Hive server that runs in an ephemeral Dataproc cluster.
2. The server processes the query and requests metadata from the metastore service.

3. The metastore service fetches Hive metadata from Cloud SQL through the Cloud SQL Proxy.
4. The server loads data from the Hive warehouse located in a regional bucket (https://cloud.google.com/storage/docs/bucket-locations#location-r) in Cloud Storage.
5. The server returns the result to the client.

Considerations for multi-regional architectures

This tutorial focuses on a single-region architecture. However, you can consider a multi-regional architecture if you need to run Hive servers in different geographic regions. In that case, you should create separate Dataproc clusters that are dedicated to hosting the metastore service and that reside in the same region as the Cloud SQL instance. The metastore service can sometimes send high volumes of requests to the MySQL database, so it is critical to keep the metastore service geographically close to the MySQL database in order to minimize impact on performance. In comparison, the Hive server typically sends far fewer requests to the metastore service. Therefore, it can be more acceptable for the Hive server and the metastore service to reside in different regions despite the increased latency.

The metastore service can run only on Dataproc master nodes, not on worker nodes. Dataproc enforces a minimum of 2 worker nodes in standard clusters and in high-availability clusters (https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/high-availability). To avoid wasting resources on unused worker nodes, you can create a single-node (https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/single-node-clusters) cluster for the metastore service instead. To achieve high availability, you can create multiple single-node clusters.

The Cloud SQL proxy needs to be installed only on the metastore service clusters, because only the metastore service clusters need to directly connect to the Cloud SQL instance. The Hive servers then point to the metastore service clusters by setting the `hive.metastore.uris` property (https://cwiki.apache.org/confluence/display/Hive/AdminManual+Metastore+Administration#AdminManualMetastoreAdministration-AdditionalConfigurationParameters) to the comma-separated list of URIs. For example:

```
thrift://metastore1:9083,thrift://metastore2:9083
```

You can also consider using a multi-regional bucket (https://cloud.google.com/storage/docs/bucket-locations#location-mr) if the Hive data needs to be

accessed from Hive servers that are located in multiple locations. The choice between regional and multi-regional buckets depends on your use case. You must balance latency, availability, and bandwidth costs. Refer to the documentation on location considerations (https://cloud.google.com/storage/docs/bucket-locations#considerations) for more details.

Figure 2 presents an example of a multi-regional architecture.

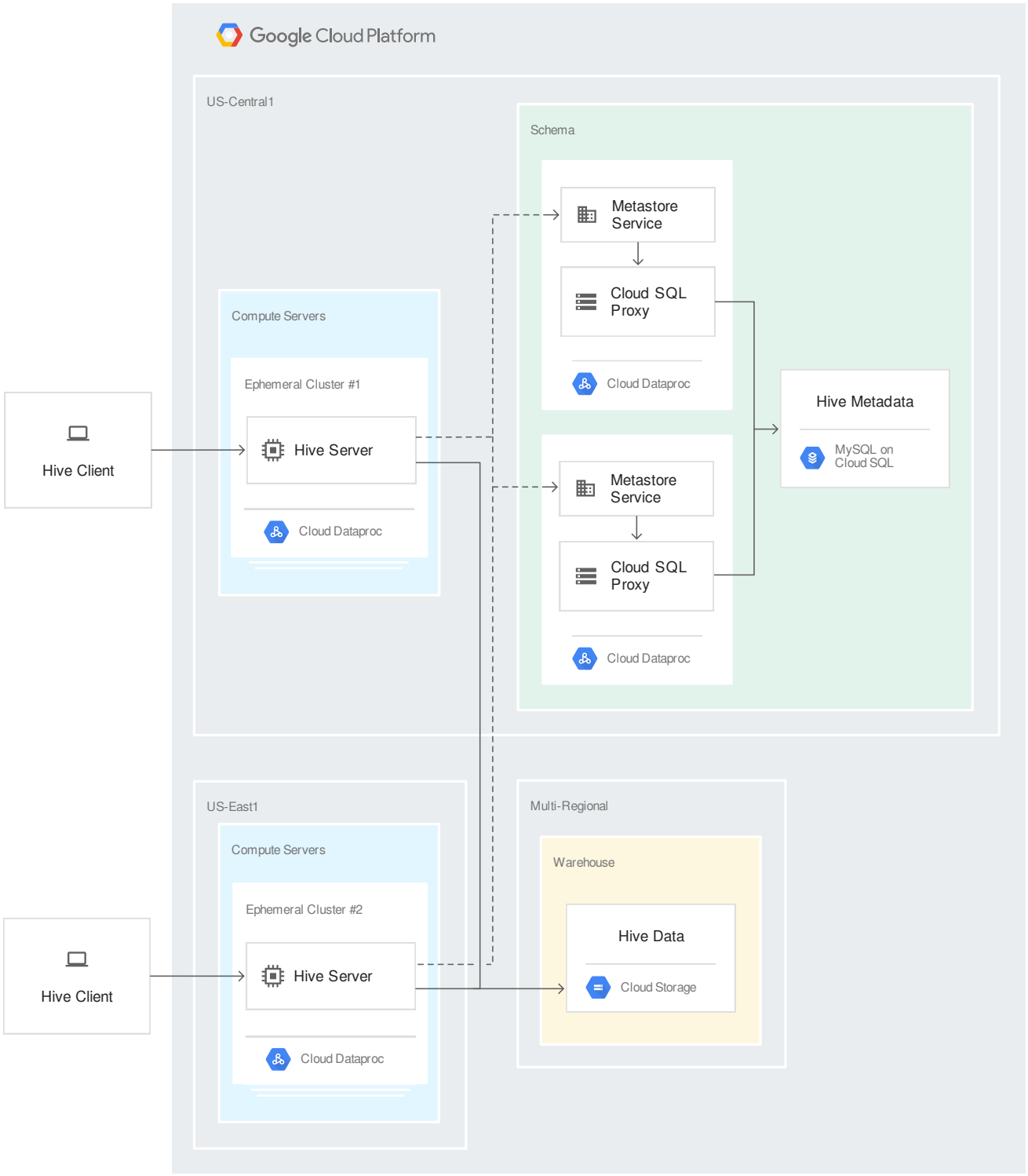

Figure 2. Example of a multi-regional Hive architecture

As you can see, the multi-regional scenario is slightly more complex. To stay concise, this tutorial uses a single-region architecture.

Creating the warehouse bucket

The first step is to create a warehouse bucket that will host the Hive data and be shared by all Hive servers.

To create the warehouse bucket, run the following commands in Cloud Shell:

```
export PROJECT=$(gcloud info --format='value(config.project)')
gsutil mb -l ${REGION} gs://${PROJECT}-warehouse
```

Creating the Cloud SQL instance

In this section, you create a new Cloud SQL instance that will later be used to host the Hive metastore.

In Cloud Shell, create a new Cloud SQL instance:

```
gcloud sql instances create hive-metastore \
    --database-version="MYSQL_5_7" \
    --activation-policy=ALWAYS \
    --gce-zone $ZONE
```

This command might take a few minutes to complete.

Creating a Dataproc cluster

Create the first Dataproc cluster:

```
gcloud dataproc clusters create hive-cluster \
    --scopes sql-admin \
    --image-version 1.3 \
    --initialization-actions gs://goog-dataproc-initialization-actions-${REGION}/clo
    --properties hive:hive.metastore.warehouse.dir=gs://${PROJECT}-warehouse/dataset
    --metadata "hive-metastore-instance=${PROJECT}:${REGION}:hive-metastore"
```

Notes:

- You provide the `sql-admin` access scope (https://cloud.google.com/sdk/gcloud/reference/dataproc/clusters/create#--scopes) to allow cluster instances to access the Cloud SQL Admin API.
- You specify the cluster image version 1.3 (https://cloud.google.com/dataproc/docs/concepts/versioning/dataproc-versions), which is the latest version available at the time of writing this tutorial.
- You provide the URI to the Hive warehouse bucket in the `hive:hive.metastore.warehouse.dir` property. This configures the Hive servers to read from and write to the correct location.
- You provide the Cloud SQL Proxy initialization action (https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/init-actions) that Dataproc automatically runs on all cluster instances. The action does the following:
  - Installs the Cloud SQL Proxy.
  - Establishes a secure connection to the Cloud SQL instance specified in the `hive-metastore-instance` metadata parameter.
  - Creates the `hive` user and the Hive metastore's database.

  You can see the full code (https://github.com/GoogleCloudDataproc/initialization-actions/tree/master/cloud-sql-proxy) for the Cloud SQL Proxy initialization action on GitHub.
- For simplicity, this tutorial uses only one master instance. To increase resilience in production workloads, you should consider creating a cluster with three master instances by using Dataproc's high availability mode (https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/high-availability).

Creating a Hive table

In this section, you upload a sample dataset to your warehouse bucket, create a new Hive table, and run some HiveQL queries on that dataset.

1. Copy the sample dataset to your warehouse bucket:

```
gsutil cp gs://hive-solution/part-00000.parquet \
gs://${PROJECT}-warehouse/datasets/transactions/part-00000.parquet
```

The sample dataset is compressed in the Parquet (https://parquet.apache.org/) format and contains thousands of fictitious bank transaction records with three columns: date, amount, and transaction type.

2. Create an external Hive table for the dataset:

```
gcloud dataproc jobs submit hive \
    --cluster hive-cluster \
    --execute "
      CREATE EXTERNAL TABLE transactions
      (SubmissionDate DATE, TransactionAmount DOUBLE, TransactionType STRING)
      STORED AS PARQUET
      LOCATION 'gs://${PROJECT}-warehouse/datasets/transactions';"
```

Running Hive queries

You can use different tools inside Dataproc to run Hive queries. In this section, you learn how to perform queries using the following tools:

- Dataproc's Hive jobs API (https://cloud.google.com/sdk/gcloud/reference/dataproc/jobs/submit/hive).
- Beeline (https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients#HiveServer2Clients-Beeline%E2%80%93CommandLineShell), a popular command line client that is based on SQLLine (http://sqlline.sourceforge.net/).
- SparkSQL (https://spark.apache.org/sql/), Apache Spark's API for querying structured data.

In each section, you run a sample query.

Querying Hive with the Dataproc Jobs API

Run the following simple HiveQL query to verify that the parquet file is correctly linked to the Hive table:

```
gcloud dataproc jobs submit hive \
    --cluster hive-cluster \
    --execute "
      SELECT *
```

```
    FROM transactions
    LIMIT 10;"
```

The output includes the following:

```
+-----------------+--------------------+------------------+
| submissiondate  | transactionamount  | transactiontype  |
+-----------------+--------------------+------------------+
| 2017-12-03      | 1167.39            | debit            |
| 2017-09-23      | 2567.87            | debit            |
| 2017-12-22      | 1074.73            | credit           |
| 2018-01-21      | 5718.58            | debit            |
| 2017-10-21      | 333.26             | debit            |
| 2017-09-12      | 2439.62            | debit            |
| 2017-08-06      | 5885.08            | debit            |
| 2017-12-05      | 7353.92            | authorization    |
| 2017-09-12      | 4710.29            | authorization    |
| 2018-01-05      | 9115.27            | debit            |
+-----------------+--------------------+------------------+
```

Querying Hive with Beeline

1. Open an SSH session with the Dataproc's master instance:

   ```
   gcloud compute ssh hive-cluster-m
   ```

2. In the master instance's command prompt, open a Beeline session:

   ```
   beeline -u "jdbc:hive2://localhost:10000"
   ```

   Notes:

   - You can also reference the master instance's name as the host instead of **`localhost`**:

     ```
     beeline -u "jdbc:hive2://hive-cluster-m:10000"
     ```

   - If you were using the high-availability mode with 3 masters, you would have to use the following command instead:

     ```
     beeline -u "jdbc:hive2://[CLUSTER_NAME]-m-0:2181,[CLUSTER_NAME]-m-1:2181,[
     ```

3. When the Beeline prompt appears, run the following HiveQL query:

```
SELECT TransactionType, AVG(TransactionAmount) AS AverageAmount
FROM transactions
WHERE SubmissionDate = '2017-12-22'
GROUP BY TransactionType;
```

The output includes the following:

```
+------------------+--------------------+
| transactiontype  |   averageamount    |
+------------------+--------------------+
| authorization    | 4890.092525252529  |
| credit           | 4863.769269565219  |
| debit            | 4982.781458176331  |
+------------------+--------------------+
```

4. Close the Beeline session:

```
!quit
```

5. Close the SSH connection:

```
exit
```

Querying Hive with SparkSQL

1. Open an SSH session with the Dataproc's master instance:

```
gcloud compute ssh hive-cluster-m
```

2. In the master instance's command prompt, open a new PySpark (http://spark.apache.org/docs/latest/api/python/pyspark.html) shell session:

```
pyspark
```

3. When the PySpark shell prompt appears, type the following Python code:

```
from pyspark.sql import HiveContext
hc = HiveContext(sc)
```

```
hc.sql("""
SELECT SubmissionDate, AVG(TransactionAmount) as AvgDebit
FROM transactions
WHERE TransactionType = 'debit'
GROUP BY SubmissionDate
HAVING SubmissionDate >= '2017-10-01' AND SubmissionDate < '2017-10-06'
ORDER BY SubmissionDate
""").show()
```

The output includes the following:

```
+----------------+-------------------+
| submissiondate |      avgdebit     |
+----------------+-------------------+
| 2017-10-01     | 4963.114920399849 |
| 2017-10-02     | 5021.493300510582 |
| 2017-10-03     | 4982.382279569891 |
| 2017-10-04     | 4873.302702503676 |
| 2017-10-05     | 4967.696333583777 |
+----------------+-------------------+
```

4. Close the PySpark session:

```
exit()
```

5. Close the SSH connection:

```
exit
```

Inspecting the Hive metastore

You now verify that the Hive metastore in Cloud SQL contains information about the `transactions` table.

1. In Cloud Shell, start a new MySQL session on the Cloud SQL instance:

```
gcloud sql connect hive-metastore --user=root
```

When you're prompted for the `root` user password, do not type anything and just press the `RETURN` key. For the sake of simplicity in this tutorial, you did not set any password for the

`root` user. For information about setting a password to further protect the metastore database, refer to the Cloud SQL documentation (https://cloud.google.com/sql/docs/mysql/create-manage-users#changing_a_user_password). The Cloud SQL Proxy initialization action also provides a mechanism for protecting passwords through encryption—for more information, see the action's code repository (https://github.com/GoogleCloudDataproc/initialization-actions/tree/master/cloud-sql-proxy#protecting-passwords-with-kms)
.

2. In the MySQL command prompt, make `hive_metastore` the default database for the rest of the session:

```
USE hive_metastore;
```

3. Verify that the warehouse bucket's location is recorded in the metastore:

```
SELECT DB_LOCATION_URI FROM DBS;
```

The output looks like this:

```
+-----------------------------------+
| DB_LOCATION_URI                   |
+-----------------------------------+
| gs://[PROJECT]-warehouse/datasets   |
+-----------------------------------+
```

4. Verify that the table is correctly referenced in the metastore:

```
SELECT TBL_NAME, TBL_TYPE FROM TBLS;
```

The output looks like this:

```
+--------------+----------------+
| TBL_NAME     | TBL_TYPE       |
+--------------+----------------+
| transactions | EXTERNAL_TABLE |
+--------------+----------------+
```

5. Verify that the table's columns are also correctly referenced:

```
SELECT COLUMN_NAME, TYPE_NAME
FROM COLUMNS_V2 c, TBLS t
WHERE c.CD_ID = t.SD_ID AND t.TBL_NAME = 'transactions';
```

The output looks like this:

```
+-------------------+-----------+
| COLUMN_NAME       | TYPE_NAME |
+-------------------+-----------+
| submissiondate    | date      |
| transactionamount | double    |
| transactiontype   | string    |
+-------------------+-----------+
```

6. Verify that the input format and location are also correctly referenced:

```
SELECT INPUT_FORMAT, LOCATION
FROM SDS s, TBLS t
WHERE s.SD_ID = t.SD_ID AND t.TBL_NAME = 'transactions';
```

The output looks like this:

```
+-------------------------------------------------------------+-----------
| INPUT_FORMAT                                                | LOCATION
+-------------------------------------------------------------+-----------
| org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat | gs://[PROJECT
+-------------------------------------------------------------+-----------
```

7. Close the MySQL session:

```
exit
```

Creating another Dataproc cluster

In this section, you create another Dataproc cluster to verify that the Hive data and Hive metastore can be shared across multiple clusters.

1. Create a new Dataproc cluster:

```
gcloud dataproc clusters create other-hive-cluster \
    --scopes cloud-platform \
    --image-version 1.3 \
    --initialization-actions gs://goog-dataproc-initialization-actions-${REGION
    --metadata  "hive-metastore-instance=${PROJECT}:${REGION}:hive-metastore"
```

You do not provide a reference to the Hive warehouse bucket the way you did earlier, when you created the first cluster with the `hive:hive.metastore.warehouse.dir` property. The bucket's location is already recorded in the Hive metastore, as you verified in the previous section.

2. Verify that the new cluster can access the data:

```
gcloud dataproc jobs submit hive \
    --cluster other-hive-cluster \
    --execute "
      SELECT TransactionType, COUNT(TransactionType) as Count
      FROM transactions
      WHERE SubmissionDate = '2017-08-22'
      GROUP BY TransactionType;"
```

The output includes the following:

```
+------------------+--------+
| transactiontype  | count  |
+------------------+--------+
| authorization    | 696    |
| credit           | 1722   |
| debit            | 2599   |
+------------------+--------+
```

Congratulations, you've completed the tutorial!

Cleaning up

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

To avoid incurring charges to your Google Cloud account for the resources used in this tutorial:

- Clean up any resources you created so you won't be billed for them in the future. The easiest way to eliminate billing is to delete the project you created for the tutorial.
- Alternatively, you can delete individual resources.

Deleting the project

> **Caution**: Deleting a project has the following effects:
>
> - **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
> - **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

   GO TO THE MANAGE RESOURCES PAGE (HTTPS://CONSOLE.CLOUD.GOOGLE.COM/IAM-ADMIN/PRC

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

Deleting individual resources

Run the following commands in Cloud Shell to delete individual resources instead of deleting the whole project:

```
gcloud dataproc clusters delete hive-cluster --quiet
gcloud dataproc clusters delete other-hive-cluster --quiet
gcloud sql instances delete hive-metastore --quiet
gsutil rm -r gs://${PROJECT}-warehouse
```

What's next

- Try BigQuery (https://cloud.google.com/bigquery/), Google's serverless, highly scalable, low-cost enterprise data warehouse.

- Check out this guide (https://cloud.google.com/solutions/migration/hadoop/hadoop-gcp-migration-overview) on migrating Hadoop workloads to Google Cloud.
- Check out this initialization action (https://github.com/GoogleCloudDataproc/initialization-actions/tree/master/hive-hcatalog) for more details on how to use Hive HCatalog (https://cwiki.apache.org/confluence/display/Hive/HCatalog) on Dataproc.
- Learn how to configure Cloud SQL for high availability (https://cloud.google.com/sql/docs/mysql/configure-ha) to increase service reliability.
- Try out other Google Cloud features for yourself. Have a look at our tutorials (https://cloud.google.com/docs/tutorials).

Last updated January 16, 2020.