Processing Logs by Using Amazon EMR

Lab overview and objectives

Apache Hadoop is an open-source framework that supports massive data processing across a cluster of instances. Hadoop can run on a single Amazon Elastic Compute Cloud (Amazon EC2) instance or thousands of instances. Hadoop uses the Hadoop Distributed File System (HDFS) to store data across multiple instances, which Hadoop calls *nodes*. Hadoop can also read data from and write data to Amazon Simple Storage Service (Amazon S3). The Amazon EMR service bundles core Hadoop and other Hadoop-related projects into each release. You will see a listing of these services when you create an EMR cluster in this lab.

In this lab, you will use Amazon EMR to process logs that are stored in an S3 bucket. To process the data in Amazon EMR, you will use Apache Hive. Hive is an open-source data warehouse infrastructure framework that is built on top of Hadoop and is often used to process large datasets. With Hive, you can query large datasets with SQL-like queries. When you run a Hive query, the request is translated into a Yet Another Resource Negotiator (YARN) job. YARN is the part of the Hadoop software that coordinates job scheduling.

After completing this lab, you should be able to do the following:

- Launch an EMR cluster through the AWS Management Console.
- Run Hive interactively.
- Use Hive commands to create tables from log data that is stored in Amazon S3.
- Use Hive commands to join tables and store the joined table in Amazon S3.
- Use Hive to query tables that are stored in Amazon S3.

Duration

This lab will require approximately **90 minutes** to complete.

AWS service restrictions

In this lab environment, access to AWS services and service actions might be restricted to the ones that are needed to complete the lab instructions. You might encounter errors if you attempt to access other services or perform actions beyond the ones that are described in this lab.

Scenario

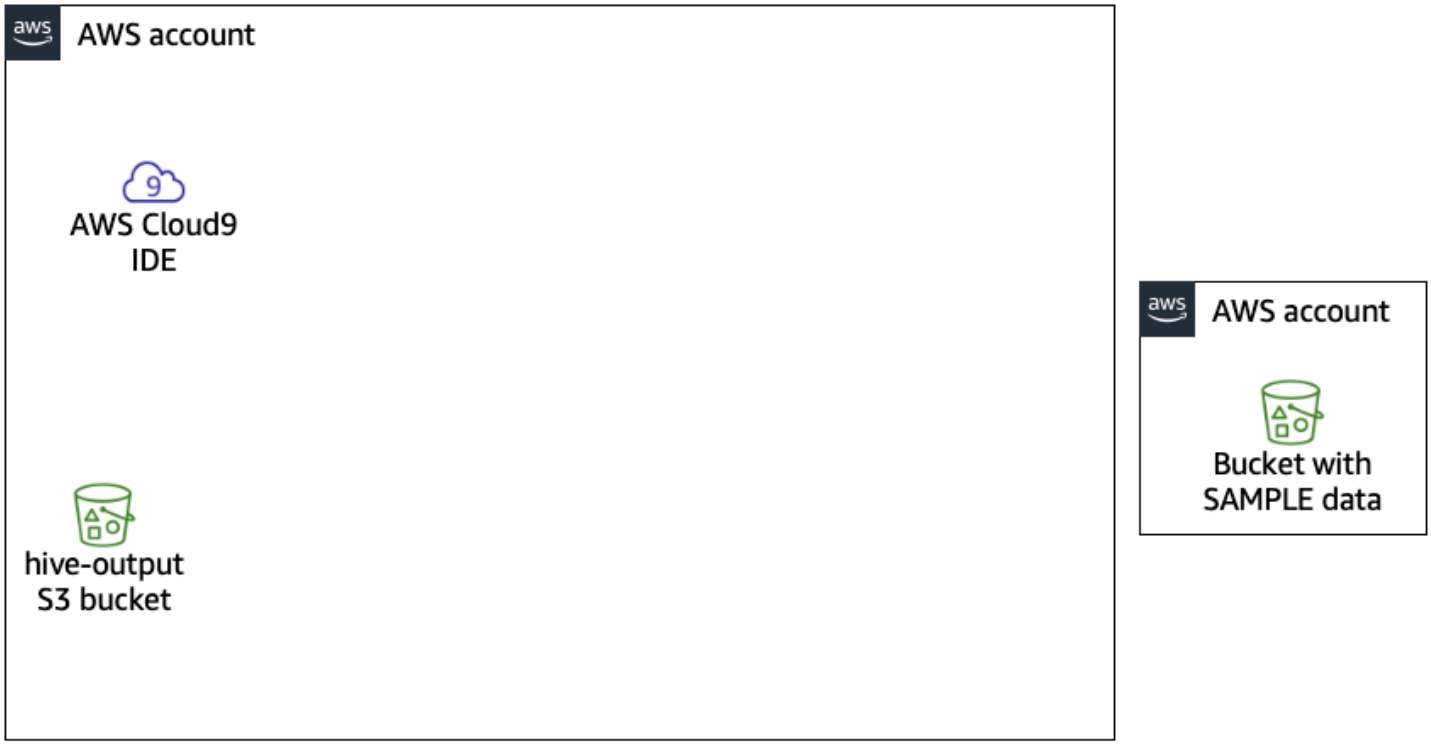
Another university department has asked your group to look for trends in the online advertising data that is contained in a set of weblog files. The logs exist in multiple files because of the log rotation process that is used to collect and store them. Two types of logs have been provided:

- **Impression logs:** Each entry indicates an advertisement being displayed to a user.
- **Click logs:** Each entry indicates someone clicking on an advertisement.

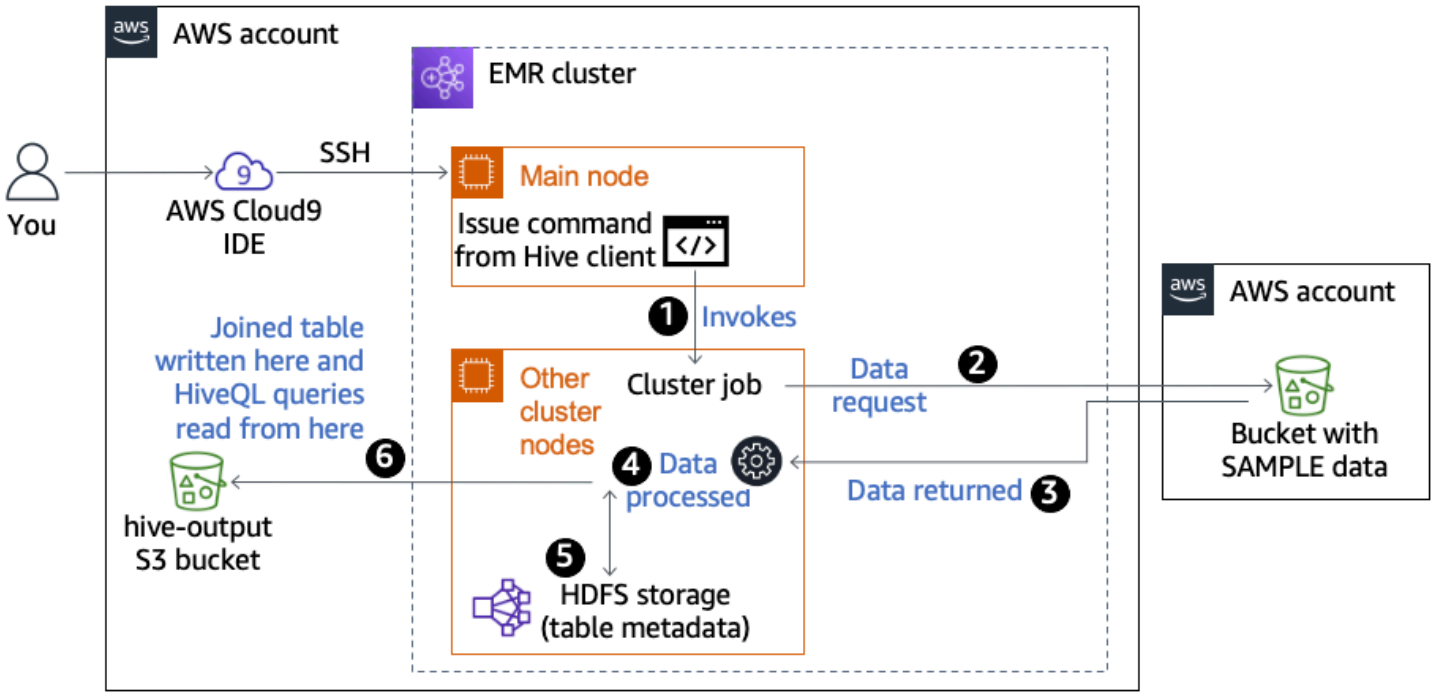
Your challenge is to develop a proof of concept (POC) to find trends in which impressions led to ad clicks.

After discussing the challenge with your teammates, you decided that Amazon EMR would be a good choice to use for this data challenge. You know that Amazon EMR can handle large datasets and that it can read from and write to Amazon S3. Additionally, you learned that you could use Hive (a tool that is bundled with Amazon EMR) to construct a data warehouse. Then, you can run SQL-like queries on the data.

When you *start* the lab, the environment will contain the resources shown in the following diagram. For this lab environment, the original data source is an S3 bucket that exists in another AWS account.



By the *end* of the lab, you will have created the additional architecture shown in the following diagram. The table after the diagram provides a detailed explanation of the architecture.



Numbered Step	Detail
1	After you create the EMR cluster, you will connect to an AWS Cloud9 integrated dev terminal to establish an SSH connection to the main node and launch the Hive comm commands, and many of those commands will invoke jobs to run on the cluster.
2	Some of the cluster jobs will read data from the SAMPLE (or source) data.
3	The requested data will be read into the cluster.
4	The jobs will process the data.
5	Hive table metadata will be stored in HDFS on the cluster. Some of the commands t HDFS.
6	Toward the end of the lab, you will join two tables and store the resulting joined data like SELECT queries in Hive, the queries will read data from this *hive-output* bucket

Accessing the AWS Management Console

1. At the top of these instructions, choose **Start Lab**.
   - The lab session starts.
   - A timer displays at the top of the page and shows the time remaining in the session.

     Tip: To refresh the session length at any time, choose **Start Lab** again before the timer reaches 0:00.
   - Before you continue, wait until the circle icon to the right of the AWS link in the upper-left corner turns green.

2. To connect to the AWS Management Console, choose the **AWS** link in the upper-left corner, above the terminal window.
    - A new browser tab opens and connects you to the console.

      Tip: If a new browser tab does not open, a banner or icon is usually at the top of your browser with the message that your browser is preventing the site from opening pop-up windows. Choose the banner or icon, and then choose **Allow pop-ups**.

      Also, if you see a message `The credentials in your login link were invalid... To logout, click here`, choose the **here** link, then double-click the AWS link above these instructions again.

Task 1: Launching an Amazon EMR cluster

In this task, you will launch an EMR cluster with Hive installed.

3. To access the EMR console, in the search box to the right of **Services**, search for and choose **EMR**.

4. Launch the process to create an EMR cluster.
    - Choose **Create cluster**.

5. Configure the options for Step 1: Software and Steps.
    - In the **Name and applications** section set Name to `Hive EMR cluster`.
    - For **Amazon EMR release**, choose **emr-5.29.0**.
    - Ensure these applications are selected:
        - Hadoop 2.8.5
        - Hive 2.3.6
    - Clear (*deselect*) all other selected applications (for example, Hue and Pig).

Analysis: The Amazon EMR release that you choose determines the version of Hadoop that will be installed on the cluster. You can also install many other Hadoop-related projects, such as Hadoop User Experience (Hue), Pig, and Spark. However, in this lab, you will only need Hadoop and Hive.

Hadoop will install and configure the cluster's internal HDFS as well as the YARN resource scheduler and coordinator to process jobs on the cluster. Hive is an open-source framework that you can use to store and query large datasets. Hive is an abstraction layer that translates SQL queries into YARN jobs that run and extract results from the data stored in HDFS. Hue and Pig are additional components of the Hadoop framework that you won't need to use in this lab.

6. Configure the options for Step 2: Hardware.

- In the **Cluster configuration** section, set the instance type and number of nodes to use:

  Note: In the console, you might see the *main node* referred to as the *primary node*. These instructions will use the term *main node*.

  - For the main node choose **m4.large** from the list.
  - Repeat the same process for the Core node type.

    Important: Due to lab security settings, if you don't change the instance type as instructed, the EMR cluster creation will fail.
  - Verify that under **Cluster scaling and provisioning** the core **Instance size** is set to `2`.
  - Verify that the instance counts are shown as follows in the Summary pane - Core size: **2 instances**.

    Analysis: The main node coordinates jobs that will run on the cluster. The main node runs the HDFS NameNode as well as the YARN ResourceManager. The core nodes act as HDFS DataNodes and are where HDFS data is replicated and stored on disk. These nodes also run MapReduce tasks as directed by YARN. Task nodes are an option that can support parallelization and Spot Instance types, but you won't need any for this lab. For more information about HDFS, YARN, MapReduce, and other Hadoop topics, see the Apache Hadoop website.
- In the **Networking** section:
  - For **Network**, choose **Lab VPC**.
  - For **EC2 Subnet**, choose **Lab subnet**.

7. Configure the options for Step 3: Cluster termination/Cluster logs - optional.

- For **Cluster termination** clear (*deselect*) the **Use termination protection** option.
- For **Amazon S3 location**, choose **Browse S3**, and select the S3 bucket for Hive output that was created for you. The bucket name has the format *s3://hive-output-xxxxxx/* (where *xxxxxx* is a unique string).

8. Configure the options for Step 4: Security configuration and EC2 key pair - optional.
   - For **Amazon EC2 key pair for SSH to the cluster**, choose the **vockey** key pair, which already exists in this account.

     Note: You will download this key pair as labsuser.pem later in this lab. Vockey and labsuser.pem are the same key pair.
   - Leave **Choose an existing service role** selected.
   - For **Service role**, confirm that **EMR_DefaultRole** is chosen.
   - Under **EC2 instance profile for Amazon EMR** and **Instance profile**, confirm that **EMR_EC2_DefaultRole** is chosen.

9. To finish creating the cluster, choose **Create cluster**.

   Your cluster will now be provisioned. Don't wait for the provisioning process to complete before continuing to the next step.

   Tip: You can ignore the warning that says *Auto-termination is not available for this account when using this release of EMR*.

10. Configure the security group for the main node to allow SSH connections from the AWS Cloud9 instance.
    - While you are on the cluster's **Summary** tab, go to the **Network and security** section below.
    - Expand **EC2 security groups (firewall)** to see the security group for the main node, choose the link for the security group. If you don't see the link yet, refresh the page. It might take a minute or two to appear.

      A new tab opens to the Security Groups page in the Amazon EC2 console.
    - Select the security group for the main node.

      Tip: You might need to expand the **Security group name** column to see the full name.
    - In the bottom pane, choose the **Inbound rules** tab, and then choose **Edit inbound rules**.
    - At the bottom of the page, choose **Add rule**, and then configure SSH access for the AWS Cloud9 instance that has been created for you:
      - **Type:** Choose **SSH**.
      - **Source:** Choose **Anywhere-IPv4**.

        Note: In a production environment, for *Source* you would typically expose SSH access to a more limited range of IP addresses. For example, you could specify a particular security group. However, in this lab, to receive full credit for your work when you submit it, keep the Anywhere-IPv4 setting.
    - Choose **Save rules**.

      Note: In the next task, you will use a bash terminal that is available in an AWS Cloud9 instance to connect to the EMR cluster through SSH. That is why you set the allowed source for inbound SSH connections to EC2 instances that use the AWS Cloud9 security group.

11. Confirm that the cluster is now available.
    - Return to the Amazon EMR console, which should still be open in another browser tab.
    - Refresh the page.
    - In the **Instances (Hardware)** section, verify that the status for the main and core node types is *Running*.

      Tip: It might take up to 10 minutes since you created the cluster for it to finish provisioning. Refresh the page to update the status.

      Important: Don't proceed to the next task until the status of the cluster shows *Waiting* and the status of the nodes show *Running*.

In this task, you provisioned an EMR cluster with Hive running on it. You also configured the security group for the main node in the cluster so that it will accept SSH connections.

Task 2: Connecting to the Hadoop main node by using SSH

In this task, you will use an SSH client and an Amazon EC2 key pair private key to connect to the Amazon EMR main node.

12. On the cluster **Summary** tab, in the **Cluster management** section, find the public DNS address for the main node. Copy the value to your clipboard.

13. Connect to the AWS Cloud9 IDE.
    - In the search box to the right of **Services**, search for and choose **Cloud9** to open the AWS Cloud9 console.
    - For the **Cloud9 Instance** environment, choose **Open IDE**.
    - In the IDE, choose **File** > **New File**.
    - Paste the public DNS value from your clipboard into the file.

      You will use this value soon.

14. Copy the SSH key to the AWS Cloud9 instance and configure it.
    - Above these lab instructions, choose **AWS Details**.
    - In the panel that opens, choose **Download PEM** and save the file to the directory of your choice.
    - Return to the AWS Cloud9 IDE.
    - Choose **File** > **Upload Local Files...**.
    - Upload the labsuser.pem file into the AWS Cloud9 IDE.

      Note: After you upload the file, it appears under the **Cloud9 Instance** folder in the file list of the **Environment** window, on the left side of the IDE.
    - Close the Upload Files dialog box.
    - Next, to modify the permissions on the key pair file so that the SSH software will allow you to use it, run the following command in the IDE terminal tab:

      ```
      chmod 400 labsuser.pem
      ```

15. Establish an SSH connection to the cluster's main node.
    - Run the following command. Replace with the public DNS address of the main node that you copied earlier.

      ```
      ssh -i labsuser.pem hadoop@<PUBLIC-DNS>
      ```

      Note the use of the *hadoop* user to log in. By logging in as this user, you will be able to access the Hive client.
    - When prompted whether to continue connecting, enter `yes`

      A message similar to the following indicates a successful connection.

Note: Because you provided the correct key pair for authentication, you are *not* prompted for a password. Recall that when you created the cluster, you specified *vockey* as the Amazon EC2 key pair to use.

You are now connected to the Amazon EMR main node, which is running on an EC2 instance, by using SSH from the AWS Cloud9 instance.

Congratulations! In this task, you successfully connected to the main node on the cluster by using the AWS Cloud9 terminal. You will use the terminal in the next task to run Hive commands.

Task 3: Running Hive interactively

In this task, you will configure a logging directory for Hive and then start an interactive Hive session on the EMR cluster's main node.

16. To create a logging directory for Hive, run the following commands:

```
sudo chown hadoop -R /var/log/hive
mkdir /var/log/hive/user/hadoop
```

Note: The Hive client expects this directory to exist when it is run. You will use the Hive client in a later step.

17. Configure Hive.
    - To retrieve the name of the S3 bucket where you want to store Hive output, run the following command:

```
aws s3 ls /
```

    - Run the following command. Replace with the full name for the *hive-output* bucket.

```
hive -d SAMPLE=s3://aws-tc-largeobjects/CUR-TF-200-ACDENG-1/emr-lab -d
DAY=2009-04-13 -d HOUR=08 -d NEXT_DAY=2009-04-13 -d NEXT_HOUR=09 -d
OUTPUT=s3://<HIVE-BUCKET-NAME>/output/
```

The command takes a few seconds to run and then displays the following output. This indicates that you are now connected to the interactive Hive client:

```
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-
log4j2.properties Async: true
hive>
```

Analysis: This command invoked the Hive program. Notice the `-d` parameters that you specified. Each of these defined a different key-value pair that applies to the session. For example, the *SAMPLE* key/value pair specified the location of the data in Amazon S3 that you want to query. This data is located in a bucket that exists outside your AWS account. The *OUTPUT* key-value pair specified the output location for query results. In this case, you specified an S3 bucket that exists in your AWS account as the place to store the results of any Hive queries that you run.

Excellent! In this task, you configured logging for Hive and established a connection to Hive.

Task 4: Creating tables out of source data by using Hive

In this task, you will create two Hive tables that will form your data warehouse. The tables will reference the data that is stored in Amazon S3. These tables will serve as the input data for queries that you will run later in the lab. In this task, you will use HiveQL, a SQL-like language, to load data and run queries.

18. To create an external table named *impressions*, run the following code in the AWS Cloud9 terminal:

```
CREATE EXTERNAL TABLE impressions (
requestBeginTime string,
adId string,
impressionId string,
referrer string,
userAgent string,
userCookie string,
ip string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe' with
serdeproperties
('paths'='requestBeginTime, adId, impressionId, referrer, userAgent,
userCookie, ip')
LOCATION '${SAMPLE}/tables/impressions';
```

The output is similar to the following:

```
OK
Time taken: 12.599 seconds
```

Analysis: Notice that this command created an *external* table. When you create an external table, the table data is *not* copied into HDFS. If you did not specify *external*, the table data would be copied into HDFS. The data for this table remains in Amazon S3. If this external table were ever to be dropped, the data would not be deleted from Amazon S3.

As you can see in the *LOCATION* part of the command, Hive is told that the data is located in the S3 bucket specified in the *SAMPLE* key-value pair that you used when you launched Hive in the previous task. Within that bucket is a *tables* folder with a subfolder named *impressions*. The *impressions* folder contains partitioned data.

If you are familiar with relational databases, where you first define a table structure and then insert data into the table, what you did here might seem backward and confusing. What you have done is point Hive to *existing* data. This existing data has structure, and in the *create table* command, you informed Hive what that structure is.

The following is a *single line* of the data that you pointed to in Amazon S3. The data contains thousands of lines similar to this. Here, the line is displayed with line breaks to make it easier for you to read, but in the source data all of this information would be on a single line. This source data is in JSON format and has 16 first-level name-value pairs. However, your table definition referenced only seven of these. Therefore, your table is only referencing *some* of the source data.


```
{
    "number": "348309",
    "referrer": "example.com",
    "processId": "1505",
    "adId": "2h8VvmXWGFrm4tAGoKPQ3UfWj3g4Jt",
    "browserCookie": "bxsbcgatmx",
    "userCookie": "CbHCJK7atU0RrnX4GAlh0w1kcvbuIJ",
    "requestEndTime": "1239581899000",
    "impressionId": "OtGijGGr0OjjUXnj181fXXsql7G3f6",
    "userAgent": "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; GTB6;
.NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; InfoPa",
    "timers": {
        "modelLookup": "0.2444",
        "requestTime": "0.7636"
    },
    "threadId": "98",
    "ip": "22.68.182.56",
    "modelId": "bxxiuxduab",
    "hostname": "ec2-64-28-73-15.amazon.com",
    "sessionId": "APrPRwrrXhwPUwsIKuOCCHpHSDTfxW",
    "requestBeginTime": "1239581898000"
}
```

If you review the statement that you used to create the table, notice how, when defining the row format, you referenced a JSON serializer/de-serializer (called *SerDe*) to parse the source data.

19. Update the Hive metadata for the *impressions* table to include all partitions.
    - To see how many partitions the *impressions* table currently has, run the following command:

      ```
      describe formatted impressions;
      ```

      The output describes the column names, location of the data, number of partitions, and other metadata about the table. The *numPartitions* line should show 0.
    - To inspect the input data and apply partitions to the metastore, run the following command:

      ```
      set hive.msck.path.validation=ignore;
      MSCK REPAIR TABLE impressions;
      ```

      Note: The *MSCK REPAIR TABLE* command scans Amazon S3 for Hive-compatible partitions that were added to the file system after the table was created. If partitions are found, they are added to the table metadata. This command is an extension found in the AWS version of Hive.

      After the command finishes running, the output is similar to the following:

      ```
      [truncated]...
      Repair: Added partition to metastore impressions:dt=2009-04-14-12-10
      Repair: Added partition to metastore impressions:dt=2009-04-14-12-15
      Repair: Added partition to metastore impressions:dt=2009-04-14-12-20
      Repair: Added partition to metastore impressions:dt=2009-04-14-13-00
      Time taken: ...[truncated]
      ```

    - Run the following command again:

      ```
      describe formatted impressions;
      ```

      The output should now indicate that 241 partitions exist in the table.

      Analysis: *Partitions* store data by organizing it logically to improve query performance. Typically, you segment data into partitions to provide faster access to specific subsets of the data. For example, queries against this log data will most likely be based on timestamps, so this data is partitioned based on time. You specified that when creating the table.

20. Create another external table and again discover its partitions. This table will be named **clicks**, and it references click-stream logs.
    - To create the table, run the following command:

```
CREATE EXTERNAL TABLE clicks (impressionId string, adId string)
PARTITIONED BY (dt string)
ROW FORMAT  serde 'org.apache.hive.hcatalog.data.JsonSerDe'
WITH SERDEPROPERTIES ('paths'='impressionId')
LOCATION '${SAMPLE}/tables/clicks';
```

The following example is a single line entry from the clicks source logs. As you can see, the logs have more data than what you are pulling in to the table; however, you did pull in the *impressionId* and *adId* values.

```
{
    "number": "8673",
    "processId": "1010",
    "adId": "S5KtvIerGRLpNxjnkn4MdRH2GqOq5A",
    "requestEndTime": "1239612672000",
    "impressionId": "PEWl8ecT2D77hdVce8oXdgtPe1m7kr",
    "timers": {
        "requestTime": "0.9578"
    },
    "threadId": "100",
    "hostname": "ec2-0-51-75-39.amazon.com",
    "requestBeginTime": "1239612671000"
}
```

- To return all partitions from the click-stream log data, run the following command:

```
MSCK REPAIR TABLE clicks;
```

It might take the command about 20 seconds to run and return you to the `hive>` prompt.

21. Verify that the two tables now exist.
    - Run the following command:

```
show tables;
```

The output returns the names of both external tables that are stored in Amazon S3: *clicks* and *impressions*.

- To describe the *clicks* table, run the following command:

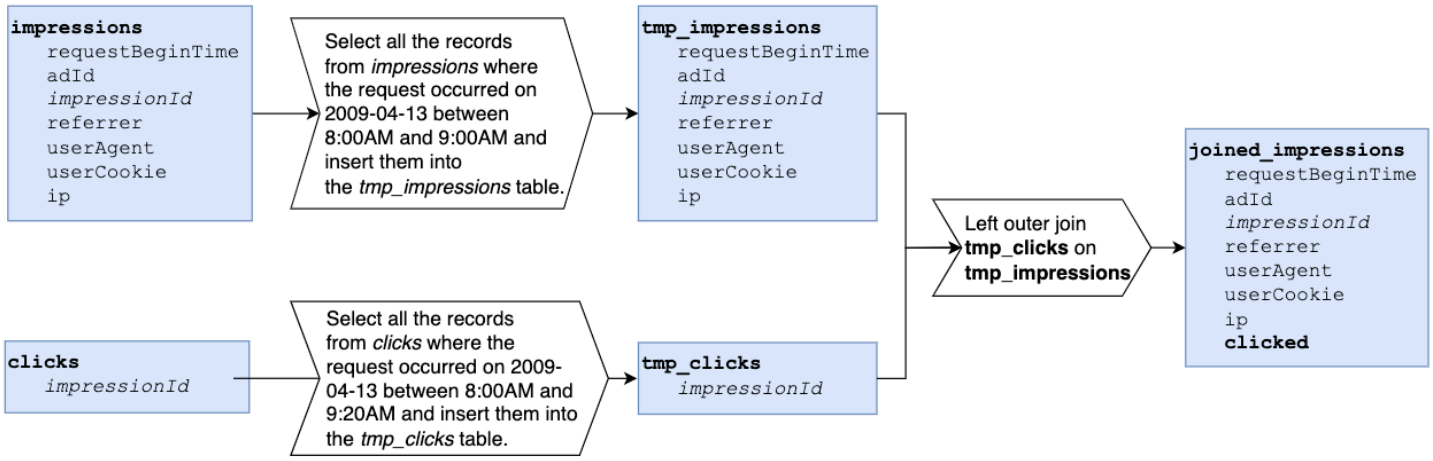
```
describe formatted clicks;
```

Perfect! In this task, you successfully created two external Hive tables that can now be used to query data.

Task 5: Joining tables by using Hive

In this section of the lab, you will join the *impressions* table with the *clicks* table. By combining information on the ads that were presented to a user (the *impressions* data) with information on which ads resulted in a user choosing the ad (the *clicks* data), you can gain insight into user behavior and target and monetize advertising services.

The following diagram illustrates what you will accomplish in this task.



Both the *clicks* and *impressions* tables are partitioned tables. When the two are joined, the *CREATE TABLE* command will tell the new table to be partitioned.

22. To create a new external table named *joined_impressions*, run the following command:

```
CREATE EXTERNAL TABLE joined_impressions (
requestBeginTime string,
adId string,
impressionId string,
referrer string,
userAgent string,
userCookie string,
ip string,
clicked Boolean)
PARTITIONED BY (day string, hour string)
STORED AS SEQUENCEFILE
LOCATION '${OUTPUT}/tables/joined_impressions';
```

Analysis: The *joined_impressions* table defines the same seven columns that exist in the *impressions* table; however, the new table also contains an eighth column named *clicked*.

Unlike the first two tables that you created (*impressions* and *clicks*), which already contain table data in the locations specified, for this table, you defined a new location that does not yet have data.

The *STORED AS SEQUENCEFILE* clause compresses the data before it is stored. Using this native Hadoop format for compression will provide better performance than keeping the data in JSON format. You will no longer need a SerDe to read the data as you did when it was stored in JSON.

So far, the table does not have any data. When data is added, it will be stored in the *hive-output* S3 bucket.

23. To create a new temporary table in HDFS named *tmp_impressions* to store intermediate data, run the following command:

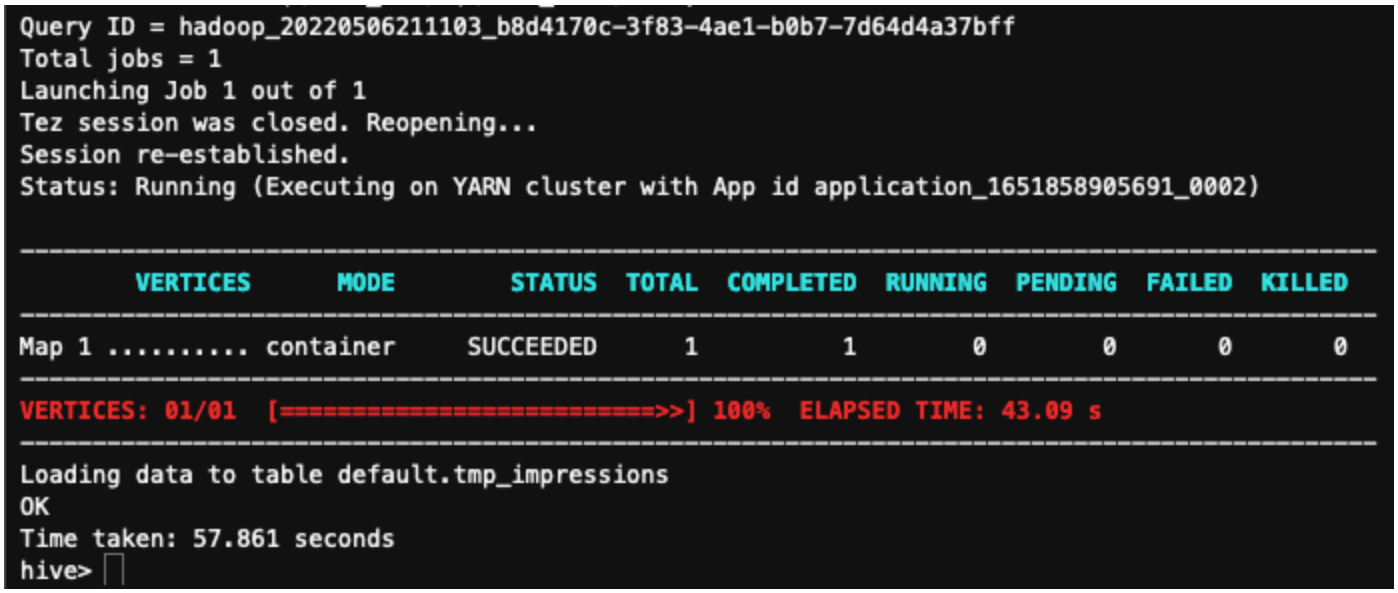
```
CREATE TABLE tmp_impressions (
requestBeginTime string,
adId string,
impressionId string,
referrer string,
userAgent string,
userCookie string,
ip string)
STORED AS SEQUENCEFILE;
```

Analysis: This table will store temporary data in the local HDFS partition. You know that it is stored in HDFS (not in S3) because the command did *not* specify *EXTERNAL* as the table type. The table has the same column names as the *impressions* table.

24. To insert impression log data for the time duration referenced, run the following command:

```
INSERT OVERWRITE TABLE tmp_impressions
SELECT
from_unixtime(cast((cast(i.requestBeginTime as bigint) / 1000) as int))
requestBeginTime,
i.adId,
i.impressionId,
i.referrer,
i.userAgent,
i.userCookie,
i.ip
   FROM impressions i
   WHERE i.dt >= '${DAY}-${HOUR}-00'
AND i.dt < '${NEXT_DAY}-${NEXT_HOUR}-00';
```

This command runs a MapReduce job to process the request. The output is similar to the following image:



Analysis: The *INSERT OVERWRITE TABLE* command will replace any existing data in the *tmp_impressions* table with new data.

The *SELECT* command selects data from the *impressions* table but only selects the records that correspond to a specific period of time. Because the *impressions* table is partitioned by requestBeginTime, only the partitions that are relevant to the specified time duration are read, which improves performance.

The naming syntax for the partitioned table is *[Partition column]=[Partition value]*. For example, *dt=2009-04-13-05*, where *dt* (date/time) is the partition column name and *2009-04-13-05* is the partition value.

The start of the time period is *DAY-HOUR*, and the end of the period is *NEXT_DAY-NEXT_HOUR*. *NEXT_DAY* is the day of the next time period. It differs from *${DAY}* only when you are processing the last hour of a day. In this case, the time period ends on the next day.

Hive converted the SELECT query into a distributed MapReduce application. The main node distributed the application to the core nodes. This is useful when you want to use the power of distributed processing to store and query your data without needing to write your own MapReduce application in a programming language such as Java.

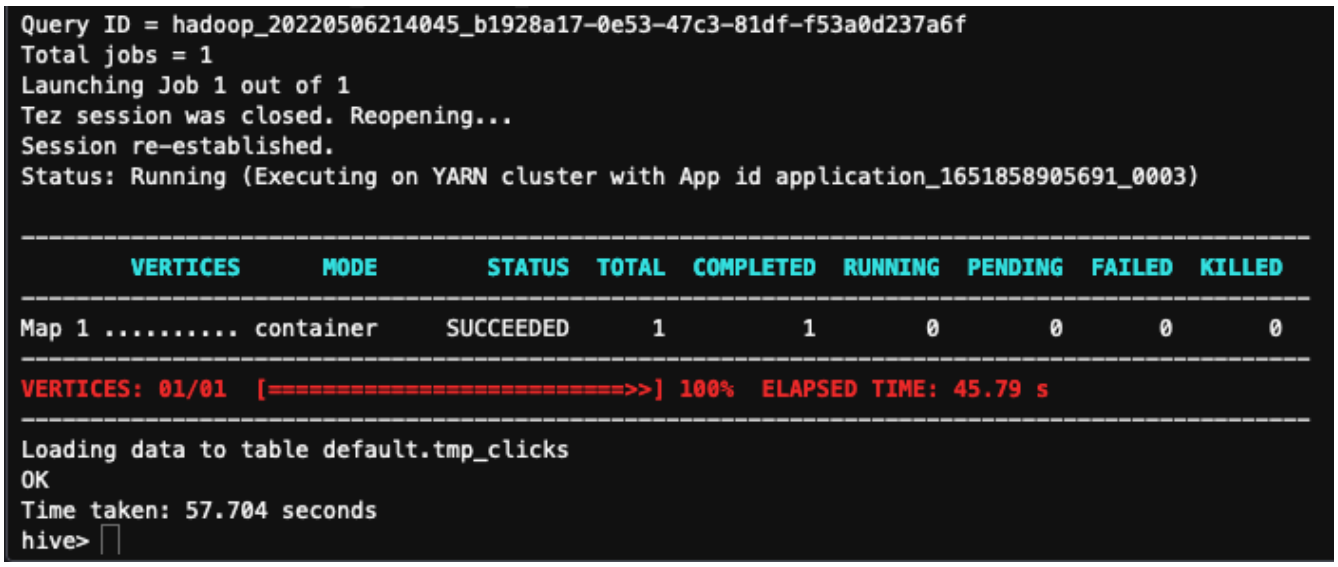
25. Create a temporary table named *tmp_clicks* and insert data into the table.
    - To create the table, run the following command:

```
CREATE TABLE tmp_clicks ( impressionId string, adId string )
STORED AS SEQUENCEFILE;
```

    - To insert data into the *tmp_clicks* table, run the following command:

```
INSERT OVERWRITE TABLE tmp_clicks
SELECT impressionId, adId
FROM clicks c
WHERE c.dt >= '${DAY}-${HOUR}-00'
AND c.dt < '${NEXT_DAY}-${NEXT_HOUR}-20';
```

The command launches another MapReduce job. The output is similar to what is shown in the image below. It will take approximately 1 minute to complete.



Analysis: For the *tmp_clicks* table, the time period analyzed is *20 minutes longer* than the time period captured in the *tmp_impressions* table. This ensures that any ad clicks that occurred up to 20 minutes after the ad was displayed will still appear in the result set.

26. To create a left outer join of *tmp_clicks* and *tmp_impressions* that writes the resulting data set to the *joined_impressions* table in your S3 output bucket, run the following command:

```
INSERT OVERWRITE TABLE joined_impressions
PARTITION (day='${DAY}', hour='${HOUR}')
SELECT
i.requestBeginTime,
i.adId,
i.impressionId,
i.referrer,
i.userAgent,
i.userCookie,
i.ip,
(c.impressionId is not null) clicked
FROM tmp_impressions i
LEFT OUTER JOIN tmp_clicks c ON i.impressionId=c.impressionId;
```

Another MapReduce job runs. It will take approximately 30 seconds. The output should look similar to the following:

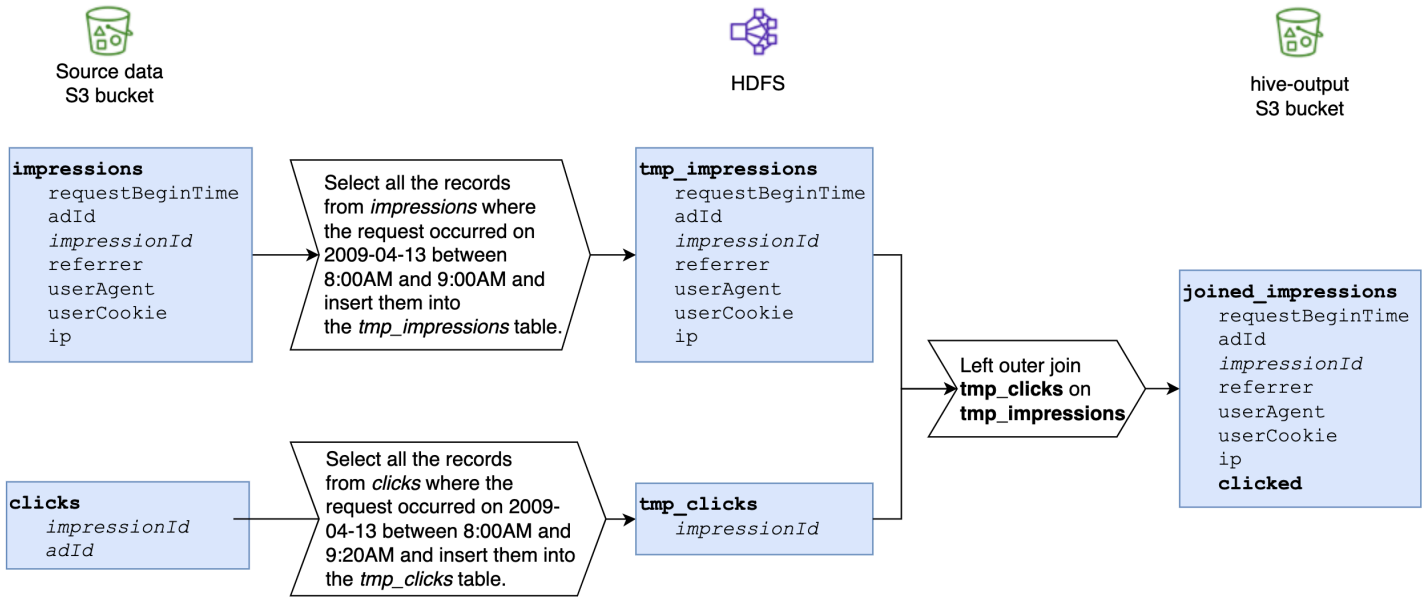
```
Query ID = hadoop_20220510204632_c9bb9b58-a70d-47e2-9b19-b28280f3ee24
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1652214366184_0002)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      1          1        0        0       0       0
Map 2 .......... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 14.38 s
----------------------------------------------------------------------------------------------
Loading data to table default.joined_impressions partition (day=2009-04-13, hour=08)
OK
Time taken: 18.842 seconds
hive>
```

Analysis: The left outer join returns all records from *tmp_impressions*, even if the join condition does not find and return matching records from the *tmp_clicks* table. If matching records are not found in *tmp_clicks*, null values are returned for each column.

The left outer join *excludes* any data from *tmp_clicks* that did not originate from an impression in the selected time period.

Because the *joined_impressions* table is stored in Amazon S3, the output can be made available to other EMR clusters.

Congratulations! In this task, you successfully combined the data from the two source tables into a single queryable table. The following diagram summarizes your accomplishment and also shows where the data of each table resides.



Because you maintain the nontemporary data in Amazon S3, you can access it at anytime. For example, you might not want to keep this cluster running for long because running servers is relatively expensive compared to only storing data. By storing the data in Amazon S3, you could delete this cluster at any time and then later, when you want to run more queries, create a new cluster to connect to the same data.

Task 6: Querying the resulting dataset

In this task, you will use Hive to run SQL-like queries on the data that now exists in the *joined_impressions* table. This will allow you to analyze which impressions led to ad clicks.

27. Use SQL-like queries to query the resulting dataset.
    - To instruct the HiveSQL client to print the names of the columns as headers in result sets, run the following command:

      ```
      set hive.cli.print.header=true;
      ```

    - To return the first 10 rows of data in the *joined_impressions* table, run the following SELECT command:

      ```
      SELECT * FROM joined_impressions LIMIT 10;
      ```

      The results should return quickly. The following screen capture shows the first few columns for the 10 rows that are returned in the results:

```
hive> SELECT * FROM joined_impressions LIMIT 10;
OK
2009-04-13 08:03:27     MQhMjITLoRSUaNnJ1gMLWPKH79102B  FLnMb6IqwlstiXdBPuDJCH43f5CDbQ
2009-04-13 08:10:10     wM8Tfrdq1HoQX4lTMH0wSlwJqLBNdw  F5LolrpIngS8gKP5TTpHt3QAF4EeGb
2009-04-13 08:02:03     pu6cmp9VA4T6Fu48JfrtuVL3b9Hwqe  lgWQgftxXiaedGCMhsVG6BpG3fFGhp
2009-04-13 08:04:16     J1fppK8UvpkJjT1eH5AXOhGAVQ9I0K  BNGiPA6DvUD2A2TIue2B3DWkMU8Egg
2009-04-13 08:05:25     AgiUsIkD6F2XvT0eBRA6PG5VvLQP7A  hCc0NweiIHw8RDKsL49qbjl38TE03u
2009-04-13 08:06:35     VwTcf8G30AfOOqf0gBBi73K6BUKGf5  uTXlFkBsImckMcnacu3XPgLKjWR3IC
2009-04-13 08:01:11     OlUBqMx3XiJ3mRmQiKBksSsUjo7Toe  SQ1KsOaJxs5l4iMhEkm7KGmdwp4UR6
2009-04-13 08:03:26     4fC04kGg4G1fQK88GrBmcqRAq5qVgG  p25ObGFBuxTQHxnquHaVNIoHCV2lmC
2009-04-13 08:07:48     wraqhTKLXsvNvq0k43GmBqsn434FHD  GurRll3BUETwtWvoMXcmiHPpAKnrI0
2009-04-13 08:02:43     FUQXdOaav8FrEhFpWVw6qdAiHi6PP7  RJ5RfN4rQ9kWuCXE4EIf9chQB0tnha
Time taken: 0.296 seconds, Fetched: 10 row(s)
hive>
```

This proves that the table has data.

- Now run a more interesting query.

```
SELECT adid, count(*) AS hits FROM joined_impressions WHERE clicked = true
GROUP BY adid ORDER BY hits DESC LIMIT 10;
```

This query returns the *adId* values for the 20 most clicked ads during the hour of weblogs that you have been analyzing. It seems that the ad with adId 70n4fCvgAV0wfojgw5xw3QAEiaULMg was the most clicked during that time. That's interesting. But it might be more interesting to figure out which referrers to the website provided the users who clicked the highest number of ads. Referrers are the websites that sent visitors to the website that you are analyzing.

You will do this next, but you will also learn how to run a Hive query from outside the Hive shell.

- To exit the Hive CLI, run the following command:

```
exit;
```

You should see that the prompt no longer displays `hive>`. Instead, the prompt should be in the format `[hadoop@ip-xx-xx-xx-xx ~]$`, which indicates that you are still connected from the AWS Cloud9 instance to one of the EMR cluster instances.

28. Continue running SQL queries against the data set.
    - Next, run the following command:

```
hive -e "SELECT referrer, count(*) as hits FROM joined_impressions WHERE
clicked = true GROUP BY referrer ORDER BY hits DESC LIMIT 10;" >
/home/hadoop/result.txt
```

The query will return a list of the top ten most effective website referrers, meaning that the referrals resulted in the most ad clicks. The result is written to a file.

- When the previous command completes, run the following command to read the results:

```
cat /home/hadoop/result.txt
```

As you can see, now that you have loaded the table data into Amazon EMR and learned how to use Hive to make SQL-like queries, you have the tools at your disposal to conduct trend analysis.

- To exit the SSH session, run the following command:

```
exit
```

The prompt should now display `voclabs:~/environment $`.

29. Analyze one of the log files that makes up the output from Amazon S3.
    - To rediscover the name of the bucket that contains the *joined_impressions* table data, run the following command:

```
aws s3 ls /
```

    - Next, to download the file that contains the data that you have been querying, run the following command (be sure to replace `<bucket-name>` with the actual hive-output bucket name):

```
aws s3 cp s3://<bucket-name>/output/tables/joined_impressions/day=2009-04-13/hour=08/000000_0 example-log.txt
```

      A new *example-log.txt* file appears in the file list of the **Environment** window, on the left side of the IDE.

    - Open the *example-log.txt* file, and review the contents.

      Analysis: Recall that when you created the *joined_impressions* table, you specified that the data should be stored in Sequence format. This isn't a normal text file, but Hadoop can interpret it. That is why some of the contents of this file doesn't display well in a text editor.

Congratulations! In this lab, you created an EMR cluster and then used Hive to create a data warehouse based on two sets of log files that you merged into a joined table. You were then able to run queries to uncover some trends in the data, such as which referrers to the website resulted in the most frequent occurrences of users clicking ads.

Your POC to demonstrate how to process this large dataset was successful. With more analysis, you could reveal more actionable information from this dataset. For example, you could build an analytics dashboard to show the data in a more user-friendly way. You will do this in a later lab.

Submitting your work

30. To record your progress, choose **Submit** at the top of these instructions.

31. When prompted, choose **Yes**.

After a couple of minutes, the grades panel appears and shows you how many points you earned for each task. If the results don't display after a couple of minutes, choose **Grades** at the top of these instructions.

Tip: You can submit your work multiple times. After you change your work, choose **Submit** again. Your last submission is recorded for this lab.

32. To find detailed feedback about your work, choose **Submission Report**.

Lab complete

Congratulations! You have completed the lab.

33. At the top of this page, choose **End Lab**, and then choose Yes to confirm that you want to end the lab.

    A message panel indicates that the lab is terminating.

34. To close the panel, choose **Close** in the upper-right corner.