Building Batch Data Analytics Solutions on AWS:

Lab 3 - Orchestrate data processing in Spark using AWS Step Functions

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

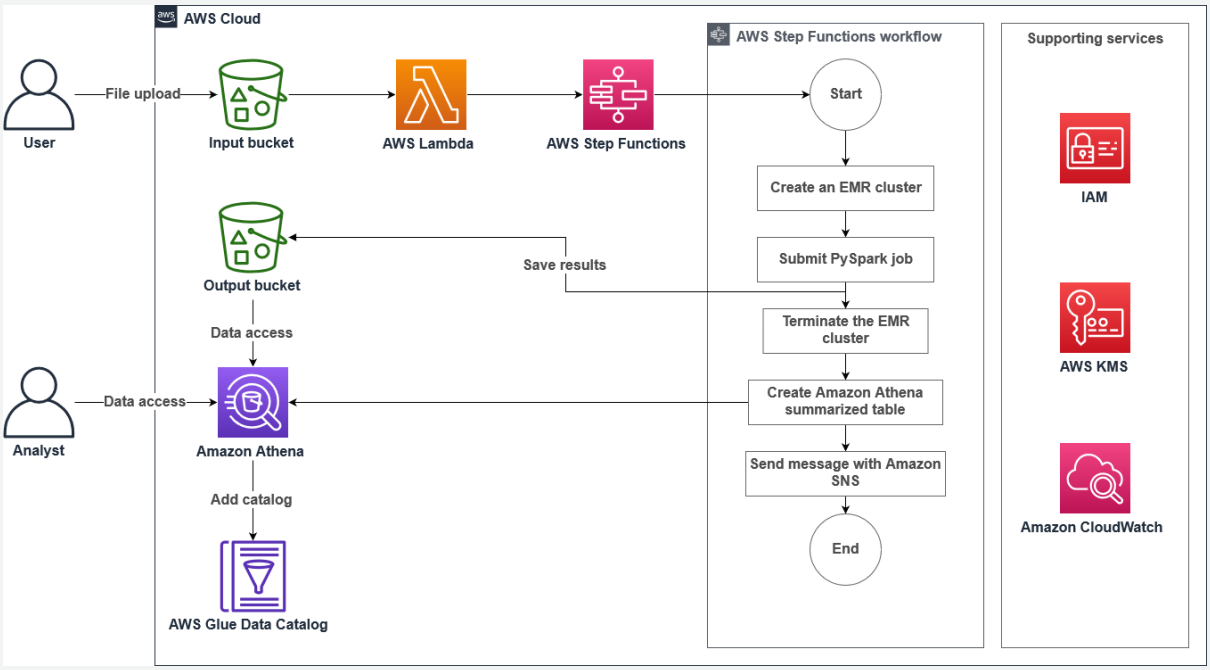
Corrections, feedback, or other questions? Contact us at *AWS Training and Certification*.

Lab overview

You have completed your research on how to use *Amazon EMR* with Apache Spark by building your prototype stock application in an *Amazon EMR* notebook (based on Jupyter Notebook). Now, you have been asked to automate the batch processing job for 2020 stock data and work on reducing costs.

Your task is to run batch processing on 2020 stock data and append it to the 2019 stock data that has already been processed. You will use *Step Functions* to implement a solution that creates an *Amazon EMR* cluster and then terminates the cluster when it is no longer needed to conserve resources and reduce cost. You will use AWS Lambda, Amazon Simple Notification Service (Amazon SNS), and state machines to achieve your goal.

You plan to use the following architecture to achieve your batch processing goal:



OBJECTIVES

By the end of this lab, you will be able to:

- Use S3 Event Notifications and AWS Lambda to automate the batch processing of data
- Use the *Step Functions* state machine language to:
    - Create an on-demand *Amazon EMR* cluster
    - Add an Apache Spark step job in *Amazon EMR* and create an *Amazon Athena* table to query the processed job
    - Add an *Amazon SNS* topic to send a notification
- Validate a Step Functions state machine run
- Review an *AWS Glue* table and validate the processed data using Athena

AWS SERVICES NOT USED IN THIS LAB

AWS service capabilities used in this lab are limited to what the lab requires. Expect errors when accessing other services or performing actions beyond those provided in this lab guide.

PREREQUISITES

This lab requires:

- Access to a text editor other than Microsoft Word such as Notepad or Notepad++

ICON KEY

Various icons are used throughout this lab to call attention to certain aspects of the guide. The following list explains the purpose for each one:

- Specifies the command you must run.
- Verify the output of a command or edited file.
- Specifies important hints, tips, guidance, or advice.
- Calls attention to information of special interest or importance. Failure to read the note does not result in physical harm to the equipment or data, but it could result in the need to repeat certain steps.
- **Note:** A hint, tip, or important guidance.
- Specifies where to find more information.

Start lab

1. To launch the lab, at the top of the page, choose Start lab.

Caution: You must wait for the provisioned AWS services to be ready before you can continue.

2. To open the lab, choose Open Console.

You are automatically signed in to the AWS Management Console in a new web browser tab.

WARNING: Do not change the Region unless instructed.

COMMON SIGN-IN ERRORS

Error: You must first sign out

If you see the message, **You must first log out before logging into a different AWS account:**

- Choose the **click here** link.
- Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
- Choose Open Console again.

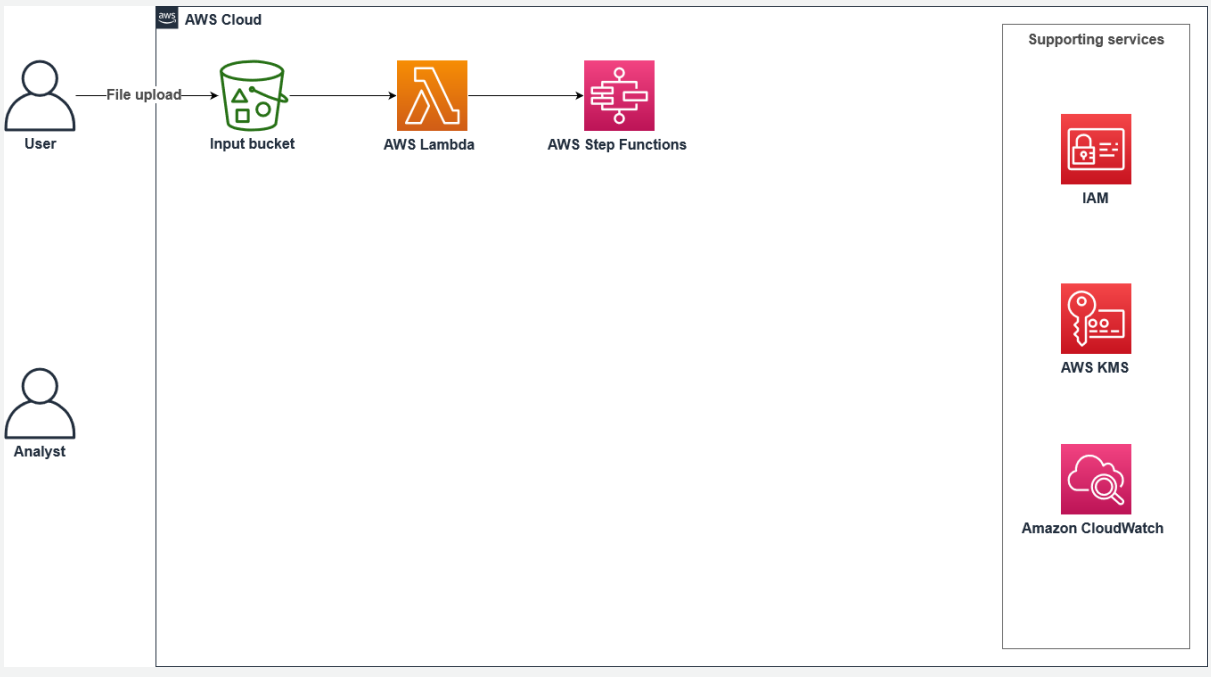
Error: Choosing Start Lab has no effect

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

- Add the lab domain name to your pop-up or script blocker's allow list or turn it off.
- Refresh the page and try again.

Task 1: Explore the lab environment

In this task, you review the account resources that were created when the lab was started. You review 2019 stock data, the Lambda function that starts the *Step Functions* step, and the *Step Functions* state machine.



REVIEW THE CONTENTS IN THE AMAZON SIMPLE STORAGE SERVICE (AMAZON S3) BUCKET

3. At the top of the page, in the unified search bar, search for and choose

S3

4. Select the bucket with **databucket** in its name.
5. The following folders were created for the task:

- **data/** - store the input stock price file in CSV format.
- **logs/** - store the *Amazon EMR* logs for troubleshooting any failures.
- **output/** - store the Spark-processed data in Parquet format.
- **results/** - store the *Amazon Athena* query results.
- **scripts/** - store the PySpark script to process data.

6. Select the name of the **data/** folder.

A file called **stock_prices_2019.csv** is saved to the folder. This object lists stock prices for a variety of large tech companies (AAPL, SQ, AMZN, GE, M, TSLA, and MSFT) for the year 2019. Data columns you can find include **Trade_Date**, **Ticker**, **High**, **Low**, **Open**, **Close**, **Volume**, and **Adj_Close**.

SAMPLE DATA

Trade_Date	Ticker	High	Low	Open	Close	Volume	Adj_Close
2019-01-02	aapl	39.712501525878906	38.557498931884766	38.722499847412109	39.479999542236328	148158800.0	38.439735412597656
2019-01-02	sq	57.830001831054690	53.560001373291016	54.099998474121094	57.200000762939450	13434000	57.200000762939450
2019-01-02	amzn	1553.3599853515625	1460.9300537109375	1465.1999511718750	1539.1300048828125	7983100	1539.1300048828125
2019-01-02	ge	7.8653850555419920	7.125	7.1730771064758300	7.7403850555419920	134528264.0	7.6648511886596680
2019-01-02	m	30.959999084472656	29.010000228881836	29.090000152587890	30.760000228881836	8168200.0	27.279628753662110
2019-01-02	tsla	63.026000976562500	59.759998321533200	61.220001220703125	62.0239982v01.750	98.940002441406250	99.550003051757810

REVIEW THE LAMBDA FUNCTION

7. At the top of the page, in the unified search bar, search for and choose

Lambda

8. In the left pane, select **Functions**.
9. On the **Functions** page, select the lambda function with **runFunction** in its name.
10. Scroll down to the **Code source** section and open the **index.py** file to review the code.

In this code, the Lambda function starts the *Step Functions* step if the event contains **databucket** as a source bucket and the event name contains **ObjectCreated:Put**.

REVIEW THE STEP FUNCTIONS STATE MACHINE DEFINITION

We have already created the *Step Functions* state machine definition as a part of the lab build. In this task, you review the *Step Functions* state machine language. The state machine completes the following tasks:

- Creates an *Amazon EMR* cluster
- Runs Spark step jobs
- Tears down the *Amazon EMR* cluster resources
- Creates an *Amazon Athena* table to query the processed data
- Adds an *Amazon SNS* topic to notify the stakeholder that the task is complete

11. At the top of the page, in the unified search bar, search for and choose

12. In the left pane, select **State machines**.
13. On the **State machines** page, select the state machine with **BatchProcessingStep** in its name.
14. Choose the **Definition** tab.

The *Step Functions* state machine code is on the left, and the workflow diagram is on the right.

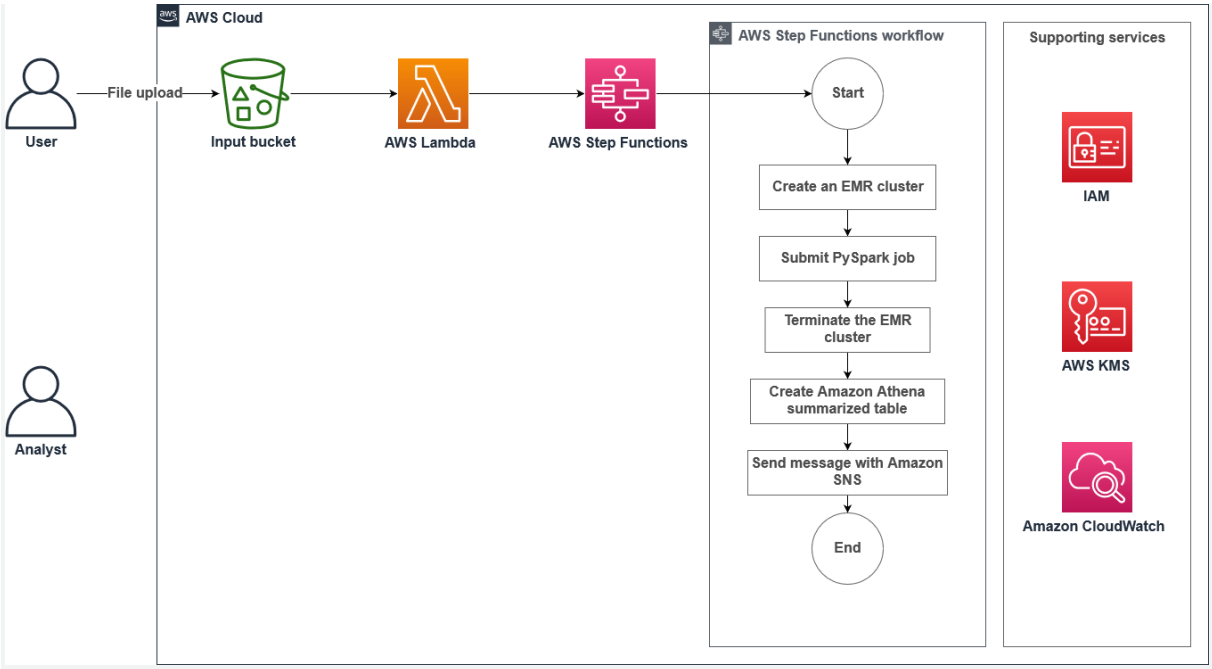
Challenge task: Take a moment to review what the state machine does to automatically provision resources, process the data, and then terminate the resources to clean up the batch process. Look at the *Amazon EMR* cluster configurations and think about how the cluster can be optimized for cost and efficiency. Try to answer the following questions:

- What applications are selected for the *Amazon EMR* cluster?
- What kind of instances are used for the leader and core fleet?
- What Amazon SNS message will be sent to the stakeholder when the steps are complete?

You can review an explanation of the *Step Functions* state machine code **here**.

Task 2: Run the Step Functions state machine task

Before you actually run the *Step Functions* steps, you need to complete some configurations. Once the configurations are set, you can upload the 2020 data and the batch processing will start.



CONFIGURE S3 EVENT NOTIFICATIONS

In this task, you use the *Amazon S3* Event Notifications feature to run the *Lambda* function when a new file is uploaded to the *S3* folder. The *Lambda* function initiates the *Step Functions* run.

You can read more about S3 Event Notifications here.

15. At the top of the page, in the unified search bar, search for and choose

    S3

16. Select the bucket with **databucket** in its name.
17. Choose the **Properties** tab and scroll down to the **Event notifications** section.
18. Choose **Create event notification**
19. On the **Create event notification** page, configure:

- **Event name**:

  file_upload

- **Prefix**:

  data/

- **Suffix**:

  .csv

- **Event types**: Select **Put**.
- **Destination**: Select **Lambda function**.
- Choose **Choose from your Lambda functions**.
- **Lambda function**: Select the function with **runFunction** in its name.

20. Choose **Save changes**

UPDATE AND UPLOAD THE PYSPARK SCRIPT IN THE AMAZON S3 DATABUCKET

Each night, your company receives the stock feed from various sources, but you want to identify high-volume trades so that you can serve a specific group of investors. This script identifies the trades with a volume greater than 100,000 shares. You need to add the script that completes this work to the scripts folder in your *S3* bucket so that the state machine can complete the stock data processing.

21. Paste the following script in a text editor.

Do not use MS Word to edit the code.

```
import sys
import time
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

bucket_name = "<dataBucket>"

spark =  SparkSession.builder.appName("stock-summary").getOrCreate()

stockDF =  spark.read.option("header",True).csv("s3://"+bucket_name+"/data/")

stockDF.registerTempTable("stock_data_view")

StockSummaryDF = spark.sql("SELECT `Trade_Date`, `Ticker`, `Close` FROM
stock_data_view WHERE Volume > 100000 ORDER BY Close DESC")

StockSummaryDF.write.mode('overwrite').parquet("s3://"+bucket_name+"/output/")

spark.stop()
```

22. Replace *<dataBucket>* with the **dataBucket** value shown to the left of these instructions.
23. Save the file as ***script.py***.
24. At the top of the page, in the unified search bar, search for and choose

    S3

25. Select the bucket with **databucket** in its name.
26. Select the **scripts/** folder and then choose **Upload**
27. On the **Upload** page, choose **Add files**
28. Navigate to the folder in your local computer and select the **script.py** file you saved in a previous step.
29. Choose **Upload**

UPDATE AWS LAKE FORMATION PERMISSIONS

As part of the *Step Functions* state machine, you use *AWS Lake Formation* to create a database in your *AWS Glue* Data Catalog. You need to grant the *Step Functions* state machine permission to create a database.

30. At the top of the page, in the unified search bar, search for and choose

AWS Lake Formation

31. If you are presented with a **Welcome to Lake Formation** pop-up window, configure below:

- Select **Add myself**
- Choose **Get Started**

32. On the **AWS Lake Formation** page, under **Administration** in the left pane, choose **Administrative roles and tasks**.
33. In the **Database creators** section, choose **Grant** and configure:

- From the **IAM users and roles** dropdown menu, select the IAM role with -*stepFunctionRole*- in its name.

You can also find the role by entering

stepFunctionRol

in the search box.

- Under **Catalog permissions**, select **Create database**.
- Choose **Grant**

SUBSCRIBE TO AN AMAZON SNS TOPIC

As part of this lab build, we have created an *Amazon SNS* topic for you. In this task, you subscribe to the topic and confirm the subscription. This will tell you when your batch processing job is complete.

34. At the top of the page, in the unified search bar, search for and choose

Simple Notification Service

35. In the left pane, choose **Topics**.
36. Select the topic with **-TaskCompleteSNS-** in its name.
37. In the **Subscriptions** tab, choose **Create subscription** and configure:

- **Protocol**: **Email**
- **Endpoint**: An email address that can receive notifications from Amazon SNS

38. Choose **Create subscription** to confirm.

You should receive an email to confirm your subscription. Confirm the subscription by selecting the **Confirm subscription** link.

39. In the left pane, choose **Topics**.
40. Select the topic with **TaskCompleteSNS** in its name.

In the **Subscriptions** section, you should see your subscription status as **Confirmed**.

UPLOAD A FILE TO THE AMAZON S3 BUCKET

In this task, you upload the 2020 stock data file to invoke the S3 event notification you created earlier, which runs a Lambda function to initiate a *Step Functions* step.

41. To download the stock_prices_2020.csv file, choose the text link.

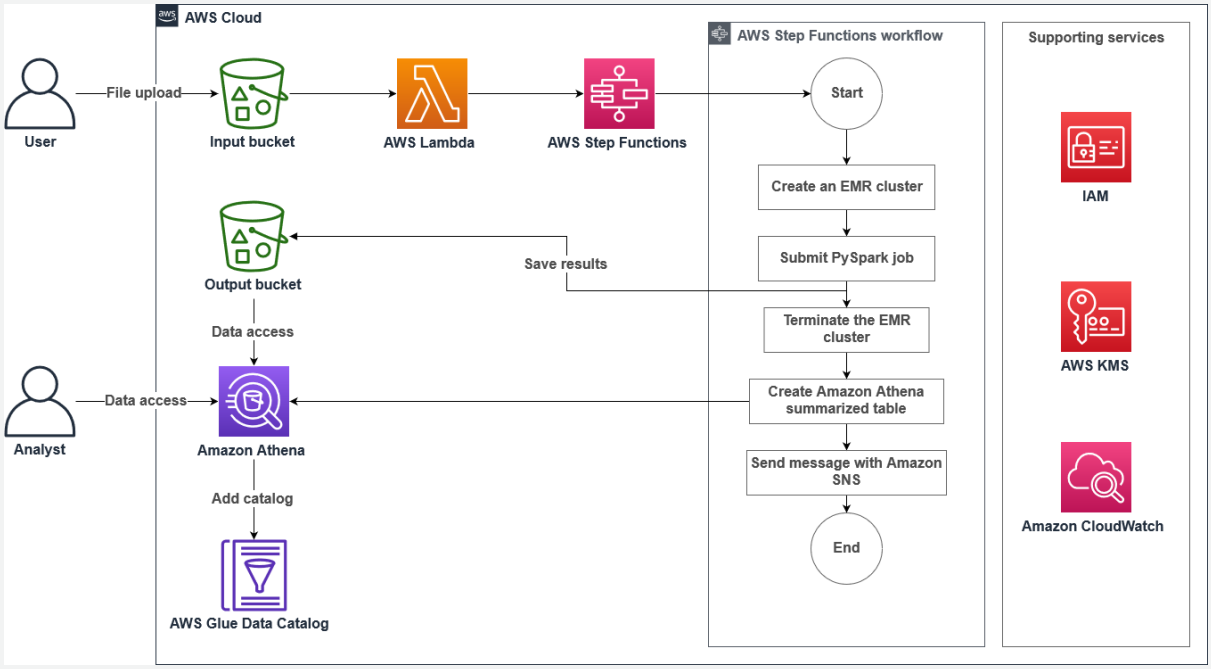
42. At the top of the page, in the unified search bar, search for and choose

S3

43. Select the bucket with **databucket** in its name.
44. Select **data/** and choose **Upload**.
45. On the **Upload** page, choose **Add files**.
46. Navigate to the folder in your local computer and select the **stock_price_2020.csv** file you saved in a previous step.
47. Choose **Upload**.

Task 3: Validate the Step Functions run

In this task, you review and validate the *Step Functions* state machine code run status to ensure the batch processing is operating correctly. Once the run is complete, you review the AWS Glue table and run a query on *Amazon Athena* to validate the processed data.



VALIDATE STEP FUNCTIONS

48. At the top of the page, in the unified search bar, search for and choose

Step Functions

If prompted to leave or stay on the page, choose **Leave Page**.

49. In the left pane, choose **State machines**.
50. On the **State machines** page, select the state machine with **BatchProcessingStep** in its name.
51. In the **Executions** section, select the name of the running state machine.

If you do not see a state machine running, you might have to re-upload the sample file in the S3 bucket.

52. Choose the **Details** tab to view the state machine status. You should see the state machine in a **Running** status.

Wait until all the steps are green and the state machine is in a **Succeeded** status. It typically takes 10-15 minutes to complete the run.

After the task is complete, you will receive an email that confirms **The Task is complete!**

While you are waiting for the task to complete, learn more about the Amazon States Language here.

UPDATE LAKE FORMATION PERMISSIONS TO VIEW THE TABLE

Once you receive notification that the task is complete, you can grant yourself permissions to access the **stock_summary** table so you can view the processed data.

53. At the top of the page, in the unified search bar, search for and choose

AWS Lake Formation

54. In the left navigation pane, choose **Data catalog settings**.
55. Clear both checkboxes for **Use only IAM access control….**

Note: If you see the error "You don't have permissions to access this resource", ignore the error.

56. Choose **Save**.
57. In the left navigation pane, choose **Data lake permissions** from the **Permissions** section.
58. Choose **Grant** and configure:

- Choose **IAM users and roles**.
- **IAM users and roles**: *Choose the Federated user you are logged in as (see instructions below for help finding this user)*.
- **LF-Tags or catalog resources**: *Named data catalog resources*
- **Databases**: *default*
- **Tables - optional**: *stock_summary*
- **Table permissions**: *Select*

Hint: Open the dropdown menu in the upper-right hand corner of your screen to copy the name of your *Federated user*. You only need the string preceding the slash (/). The role name at the end of the string should be removed when pasted into the **IAM users and roles** field.

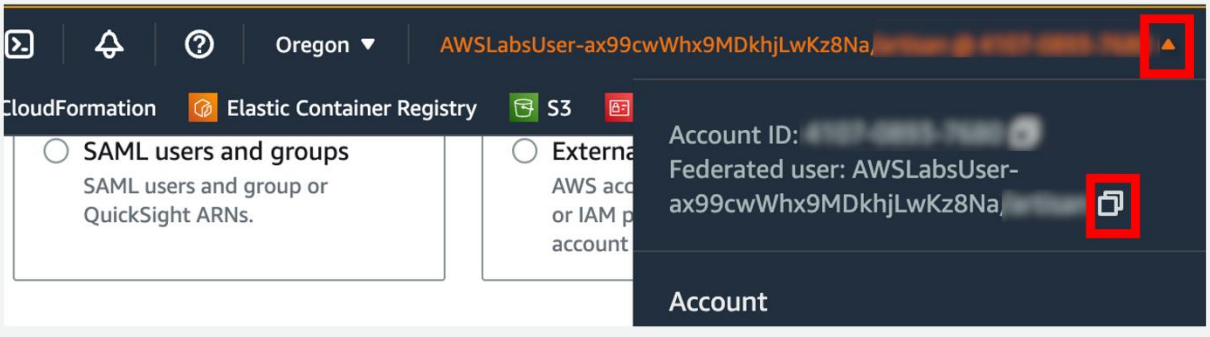


Image description: The preceding image shows how the Federated user can be copied from the AWS console

Note: With these selections, you are allowing your user or role to perform *select* operations on the **stock_summary** table. The user or role can also grant *select* permissions to other users or roles.

59. Choose **Grant**.

VALIDATE THE AWS GLUE TABLE AND RUN A QUERY ON ATHENA TO VALIDATE THE DATA

In this task, you review the **stock_summary** table in the AWS Glue Data Catalog and run a query in Amazon Athena to validate the data. In AWS Glue, a **table** is the metadata definition that represents your data, including its schema.

60. At the top of the page, in the unified search bar, search for and choose

AWS Glue

61. In the left pane **Data catalog** section, under **Databases**, choose **Tables**.
62. On the **Tables** page, choose **stock_summary**.

You should see the schema and details of the table you created with Step Functions.

If no tables appear, delete anything that is in the search box.

63. At the top of the page, in the unified search bar, search for and choose

Athena

Note: If you are brought to the *Amazon Athena* homepage, select **Launch query editor** to navigate to the *Query editor*.

Before querying your data, you must first configure it for use as an *Amazon Athena* database.

64. Configure the following:

- **Data Source**: **AwsDataCatalog**
- **Database**: default

If a **Workgroup primary settings** window opens, choose **Acknowledge**.

Under **Tables**, you can see the **stock_summary** table listed.

65. Paste the following command in the query page:

```
SELECT * FROM "default"."stock_summary" ORDER BY trade_date DESC LIMIT 10;
```

66. Choose **Run**

You should see 10 results in the **Results** section.

If you receive a **No output location provided** error, use the following steps to update the query result location.

- Choose **Settings** tab at the top right corner of the **Athena** page and then choose **Manage**:
- **Query result location**:

```
s3://<dataBucket>/results/
```

- Choose **Save** and run the query again.
- Output of the query shall get saved in the above mentioned query result location.

Replace <*dataBucket*> with the **dataBucket** value shown to the left of these instructions.

Conclusion

Congratulations! You now have successfully:

- Used S3 Event Notifications and AWS Lambda to automate the batch processing of data
- Reviewed the AWS Step Functions state machine language to:
  - Create an on-demand EMR cluster
  - Add a Spark step job in Amazon EMR and create an Amazon Athena table to query the processed job
  - Add an Amazon SNS topic to send a notification
- Validated a Step Functions state machine run
- Reviewed an AWS Glue table and validated the processed data using Athena

End lab

Follow these steps to close the console and end your lab.

67. Return to the **AWS Management Console**.
68. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
69. Choose **End lab** and then confirm that you want to end your lab.

For more information about AWS Training and Certification, see *https://aws.amazon.com/training/*.

Your feedback is welcome and appreciated.
If you would like to share any feedback, suggestions, or corrections, please provide the details in our *AWS Training and Certification Contact Form*.

Appendix

STEP FUNCTIONS STATE MACHINE CODE REVIEW

Let's review all five different states in the Step Functions state machine definition. The following section contains sample code. It is for reference only.

CREATE AN EMR CLUSTER

In this section, the example code is defining a few important features of the EMR cluster.

- **Resource**: Define the

  `Create an EMR cluster` task.

- **ReleaseLabel**: The release version of the EMR cluster.
- **Applications**: The list of applications to be installed.
- **ServiceRole**: IAM EMR Default Role name.
- **JobFlowRole**: IAM EC2 Instance Profile name.
- **LogUri**: Amazon S3 URI to store the EMR logs.
- **Instances**: Details of leader node and core nodes.
- **ResultPath**: Location to store the output of this state.
- **Next**: The name of the next state to be run.

```
"Create an EMR cluster": {
      "Type": "Task",
      "Resource": "arn:aws:states:::elasticmapreduce:createCluster.sync",
      "Parameters": {
        "Name": "labCluster",
        "VisibleToAllUsers": true,
        "ReleaseLabel": "emr-6.3.0",
        "Applications": [
          {
            "Name": "Spark"
          },
          {
            "Name": "Flink"
          },
          {
            "Name": "Livy"
          }
        ],
        "ServiceRole": "EMRDefaultRole",
        "JobFlowRole": "EMREC2InstanceProfile",
        "LogUri": "s3://lab3-databucket/logs/",
        "Instances": {
          "Ec2SubnetId": "subnet-015218b49b33dc30e",
          "KeepJobFlowAliveWhenNoSteps": true,
          "InstanceFleets": [
            {
              "Name": "MyLeaderFleet",
              "InstanceFleetType": "MASTER",
              "TargetOnDemandCapacity": 1,
              "InstanceTypeConfigs": [
                {
                  "InstanceType": "m5.xlarge"
                }
              ]
            },
            {
              "Name": "MyCoreFleet",
              "InstanceFleetType": "CORE",
              "TargetOnDemandCapacity": 2,
              "InstanceTypeConfigs": [
                {
                  "InstanceType": "m5.xlarge"
```

```
                }
              ]
            }
          ]
        }
      },
      "ResultPath": "$.cluster",
      "Next": "Submit PySpark Job"
    }
```

SUBMIT PYSPARK JOB

In this section, the example code is adding a step task to the EMR cluster.

- **Resource**: Define EMR and add step task.
- **ClusterId.$**: The cluster ID value is retrieved from the previous result path.
- **HadoopJarStep**: This section contains the

  `spark-submit` command. In the following example, the Spark step is defined to run in cluster mode with run memory allocation of 1 GB. The location of the PySpark script is also defined here.

- **ResultPath**: Location to store the output of this state.
- **Next**: The name of the next state to be run.

```
"Submit PySpark Job": {
      "Type": "Task",
      "Resource": "arn:aws:states:::elasticmapreduce:addStep.sync",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId",
        "Step": {
          "Name": "pyspark-job",
          "HadoopJarStep": {
            "Jar": "command-runner.jar",
            "Args": [
              "spark-submit",
              "--deploy-mode",
              "cluster",
              "--executor-memory",
              "1g",
              "s3://lab3-databucket/scripts/script.py"
            ]
          }
        }
      },
      "ResultPath": "$.sparkJob",
      "Next": "Terminate Cluster"
    }
```

TERMINATE THE EMR CLUSTER

In this section, the example code is adding a step task to terminate the EMR cluster.

- **Resource**: Define EMR decommission step task.

- **ClusterId.$**: The cluster ID value is retrieved from the previous result path.
- **ResultPath**: Location to store the output of this state.
- **Next**: The name of the next state to be run.

```
"Terminate the EMR Cluster": {
      "Type": "Task",
      "Resource": "arn:aws:states:::elasticmapreduce:terminateCluster",
      "Parameters": {
        "ClusterId.$": "$.cluster.ClusterId"
      },
      "ResultPath": "$.terminateCluster",
      "Next": "Create Amazon Athena Summarized Output Table"
    }
```

CREATE A SUMMARIZED OUTPUT TABLE IN ATHENA

In this section, the example code is adding a step task to create an Athena table for the processed data in the Amazon S3 bucket.

- **Resource**: Define Athena

  `startQueryExecution` step task.

- **Parameters**: This section contains a
  `query` string,
  `WorkGroup` in Athena and a

  `result` path to save the query results.

- **ResultPath**: Location to store the output of this state.
- **Next**: The name of the next state to be run.

```
"Create Amazon Athena summarized table": {
      "Type": "Task",
      "Resource": "arn:aws:states:::athena:startQueryExecution.sync",
      "Parameters": {
        "QueryString": "CREATE EXTERNAL TABLE IF NOT EXISTS
default.stock_summary(`Trade_Date` string,`Ticker` string,`Close` string) ROW
FORMAT SERDE   'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat' OUTPUTFORMAT
'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat' LOCATION
's3://lab3-databucket/output/' TBLPROPERTIES ('classification'='parquet',
'compressionType'='none', 'typeOfData'='file')",
        "WorkGroup": "primary",
        "ResultConfiguration": {
          "OutputLocation": "s3://lab3-databucket/results/"
        }
      },
      "ResultPath": "$.athenaTable",
      "Next": "Send message to SNS"
    }
```

SEND COMPLETION MESSAGE USING AMAZON SNS

In this section, the example code is adding a step task to send an Amazon SNS notification.

- **Resource**: Define SNS publish step task.
- **Parameters**: This section contains your SNS and message ARN.
- **End**: You define the value

  `true` to end the states.

```
"Send message with Amazon SNS": {
      "Type": "Task",
      "Resource": "arn:aws:states:::sns:publish",
      "Parameters": {
        "TopicArn": "arn:aws:sns:us-west-2:1234567890:TaskCompleteAgain",
        "Message": {
          "Input": "The Task is complete!"
        }
      },
      "End": true
    }
```

To continue this lab, move on to **Task 2**.