Orchestrate Amazon EMR Serverless jobs with AWS Step functions

Sathish Krishnan · Follow

10 min read · Jan 13, 2024

Introduction

Amazon EMR Serverless is a serverless option in Amazon EMR that makes it easy for data analysts and engineers to run open-source big data analytics frameworks such as Apache Spark and Apache Hive, without configuring, managing, and scaling clusters or servers. Amazon EMR Serverless gives all the features and benefits of Amazon EMR without the need for experts to plan and manage clusters.

AWS Step Functions is a serverless orchestration service that enables developers to build visual workflows for applications as a series of event-

driven steps. Step Functions ensures that the steps in the serverless workflow are followed reliably, that the information is passed between stages, and errors are handled automatically.

Integrating AWS Step Functions and Amazon EMR Serverless makes it easier to manage and orchestrate big data workflows. This integration simplifies your architecture by eliminating the need for additional steps to monitor job status, making the whole system more efficient and easier to manage.

In this blog, we explain how you can orchestrate a PySpark application using Amazon EMR Serverless and AWS Step Functions. We run a Spark job on EMR Serverless that processes 2023 Data Scientists Salary dataset in an Amazon Simple Storage Service (Amazon S3) bucket and stores the aggregated results in Amazon S3.

Overview

In this solution, we will explain integrating Amazon EMR Serverless and AWS Step Functions by creating and running a PySpark job using the 2023 Data Scientists Salary dataset. The following dataset contains parameters like work year, experience level, job type, job level and various other components related to Data Scientists Salary. Using this dataset, we will find the maximum and minimum salary for each Engineer job title.

Proposed Solution

The solution is implemented as follows:

1. Create a State Machine to orchestrate Amazon EMR Serverless workflow
2. Create a PySpark job and store it in a S3 location to run in EMR Serverless application
3. Following tasks should be orchestrated to create, start and run a PySpark job in an EMR Serverless Application
   - A Step Function Task to create an EMR Serverless application
   - A Step Function Task to start the application
   - A Step Function Task to run (in synchronous mode) a PySpark Job in the application
4. If the Tasks are completed, set the status of the state machine to “success”
5. In case of any Task (or) Job failure, catch the exception and set its status to “failed”

If any job is failed due to any error during the execution, it will be indicated as a failure. You can inspect the cause of the error in the state machine workflow. You can also check the EMR Serverless application for more detailed error logs in the EMR studio console.

Prerequisites

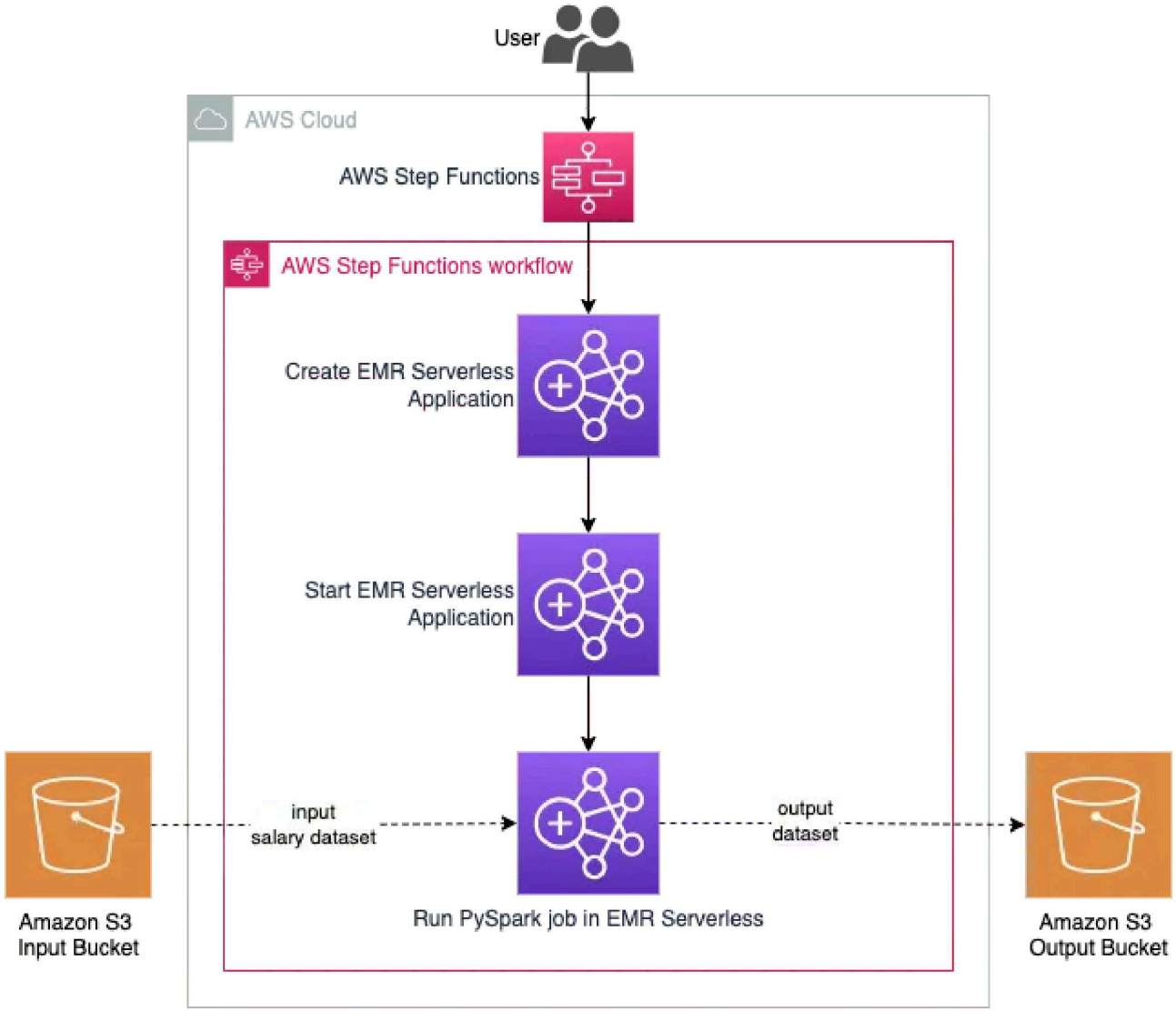
To begin with, make sure the following prerequisites are available

1. An AWS account
2. An IAM user with access to AWS Step Functions, Amazon EMR Serverless and Amazon S3
3. An S3 bucket

Architecture

The proposed solution is developed with the following architecture, which integrates Step Function for orchestration and Amazon EMR Serverless for transformation. The Resultant dataset is written as a CSV file to Amazon S3 bucket.

The below diagram illustrates the architecture for the proposed solution.



Architecture to orchestrate Amazon EMR Serverless job using AWS Step Functions

Deployment Steps

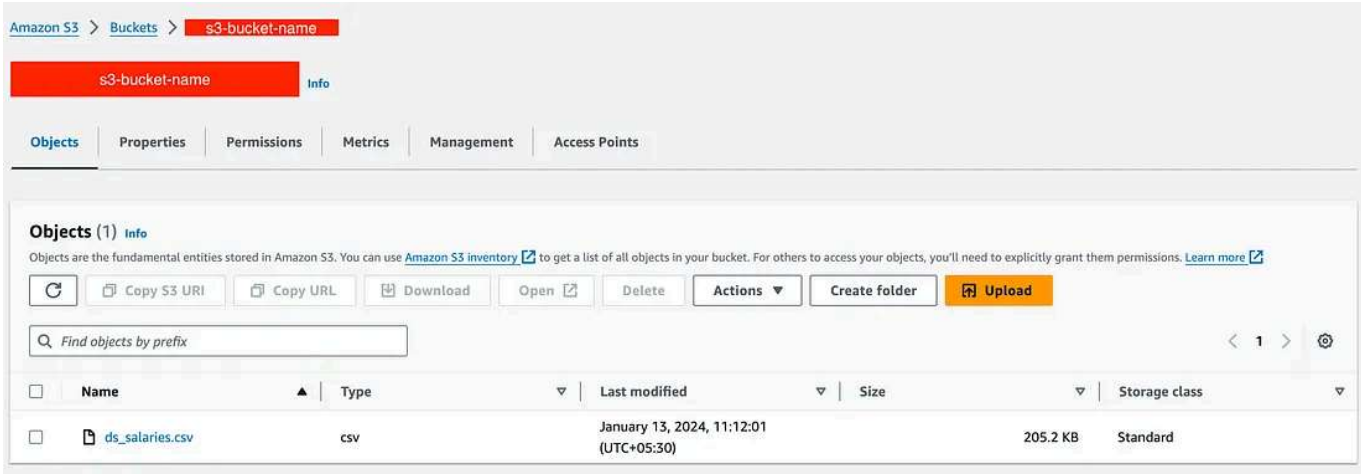
Before beginning this tutorial, ensure that the IAM role being used to deploy has all the relevant permissions to create the required resources as part of

the solution and S3 bucket is created for storing input and output datasets. The roles with the appropriate permissions will be created in AWS IAM (Identity and Access Management) console using the following steps.

Step 1: Create S3 Bucket

Create S3 bucket by the following these steps:

1. In **AWS** console, navigate to **S3** console
2. Click **Create bucket** and give a global unique bucket name
3. Add appropriate **Tags** and set the **Default encryption** type as **Server-side encryption with Amazon S3 managed keys (SSE-S3)** and click **Create bucket**
4. Once the S3 bucket is created, upload the 2023 Data Scientists Salary dataset



AWS S3 bucket for EMR Serverless POC

Step 2: Create PySpark script

Create a Pyspark script to be executed in EMR Serverless application. In this script, we will be reading the 2023 Data Scientists Salary dataset from S3 bucket get the maximum and minimum salary for each job title and store the output dataset into S3 bucket. Upload the following code into S3 bucket.

process_ds_salaries.py

```python
from pyspark.sql import SparkSession
from pyspark.sql import functions as func


if __name__ == "__main__":
    spark = (
        SparkSession.builder.appName("Demo PySpark Job")
        .enableHiveSupport()
        .getOrCreate()
    )

    df = (
        spark.read.csv("s3://aws-s3-bucket-name/ds_salaries.csv", header=True)
        .select("job_title", "salary")
        .groupBy("job_title")
        .agg(
            func.max("salary").alias('maximum_salary'),
            func.min("salary").alias('minimum_salary')
        )
    )
    print("dataset aggregation complete")
    df.write.csv("s3://aws-s3-bucket-name/output", header=True)
    print("dataset write complete")

    spark.stop()

```

PySpark script to get maximum and minimum salary for each job title

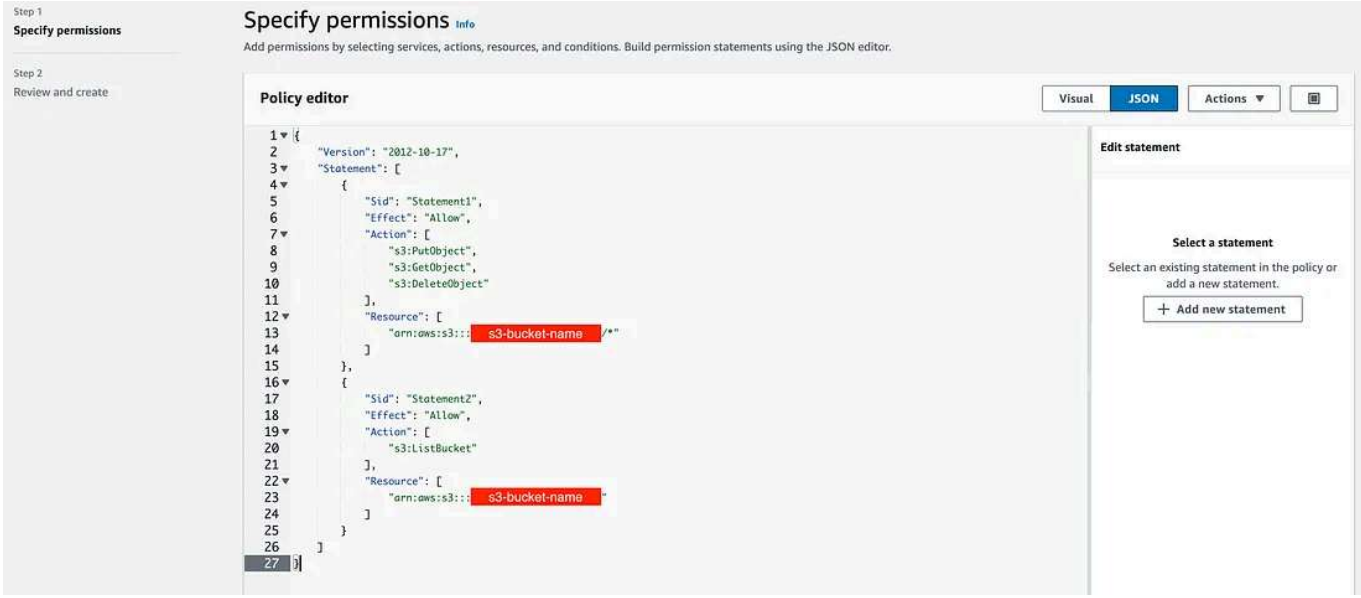
Step 3: Create AWS IAM roles

To manage and run EMR Serverless application through state machine, we require two IAM roles. One IAM role assigned to state machine to manage EMR Serverless application and the other IAM role assigned to EMR Serverless application to run PySpark job with appropriate S3 permissions.

IAM Role 1: EMR Serverless

Create EMR Serverless IAM role by following these steps:

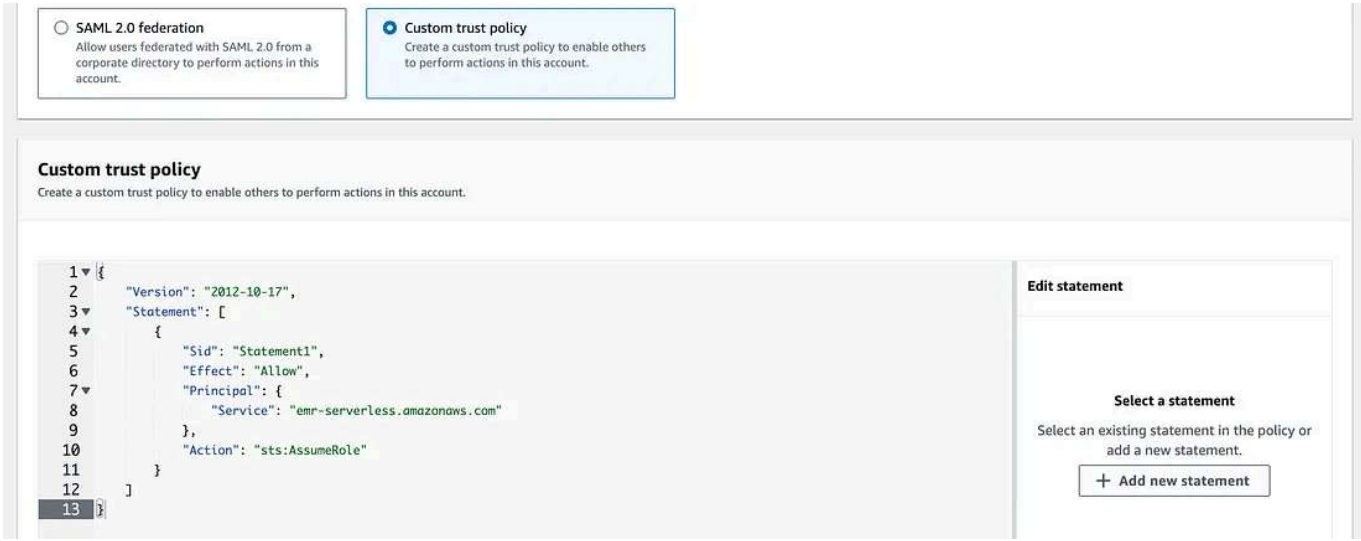
1. In **AWS** console, navigate to **IAM** console
2. Under **Access Management** tab, navigate to **Policies** page and click **Create policy**
3. Add the following permissions and add a meaningful **Policy name** (eg: emr-serverless-policy), **Policy Description** and appropriate **Tags** and click **Create policy** button



emr-serverless-policy IAM Policy

1. Once the policy is created, navigate to **Roles** page

2. Click **Create role** to create a new IAM role for EMR Serverless
3. Choose **Custom trust policy** under **Trusted entity type** section and add the following statement and click **Next** button
4. Add the created policy **emr-serverless-policy**, add a **Role name** (eg: emr-serverless-role), **Description** and appropriate **Tags** and click **Create role**

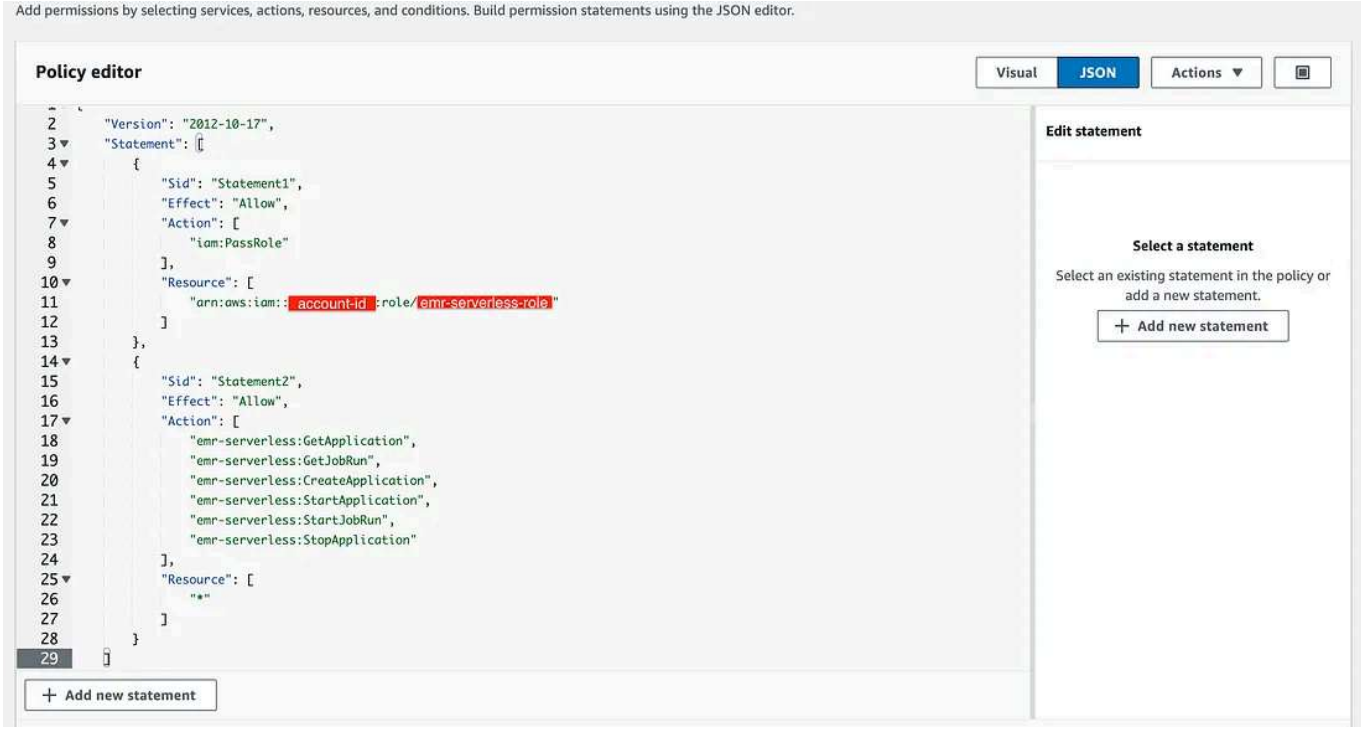


emr-serverless-role IAM Role

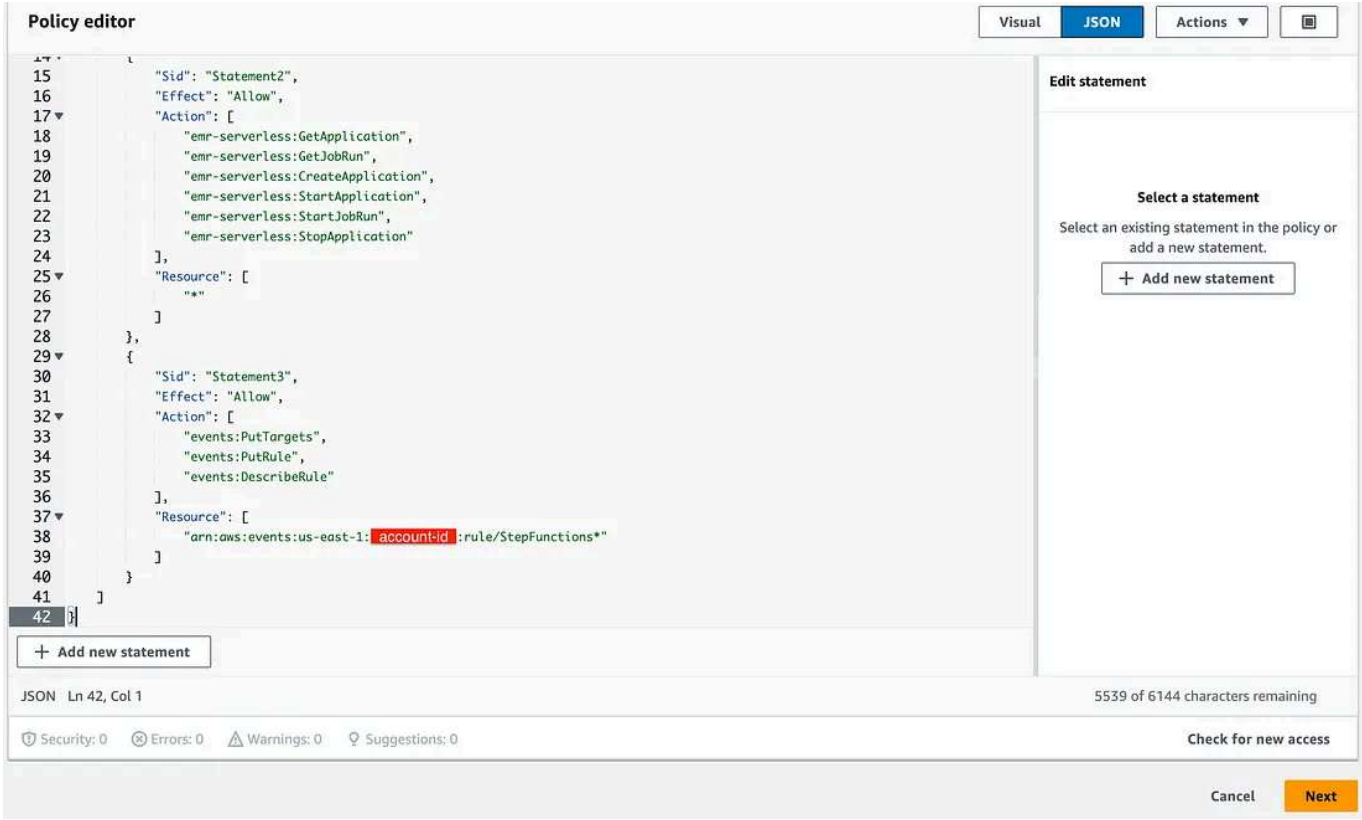
IAM Role 2: Step Functions state machine

Create Step Functions state machine IAM role by following these steps:

1. In **AWS** console, navigate to **IAM** console
2. Under **Access Management** tab, navigate to **Policies** page and click **Create policy**
3. Add the following permissions and add a meaningful **Policy name** (eg: step-functions-policy), **Policy Description** and appropriate **Tags** and click **Create policy** button



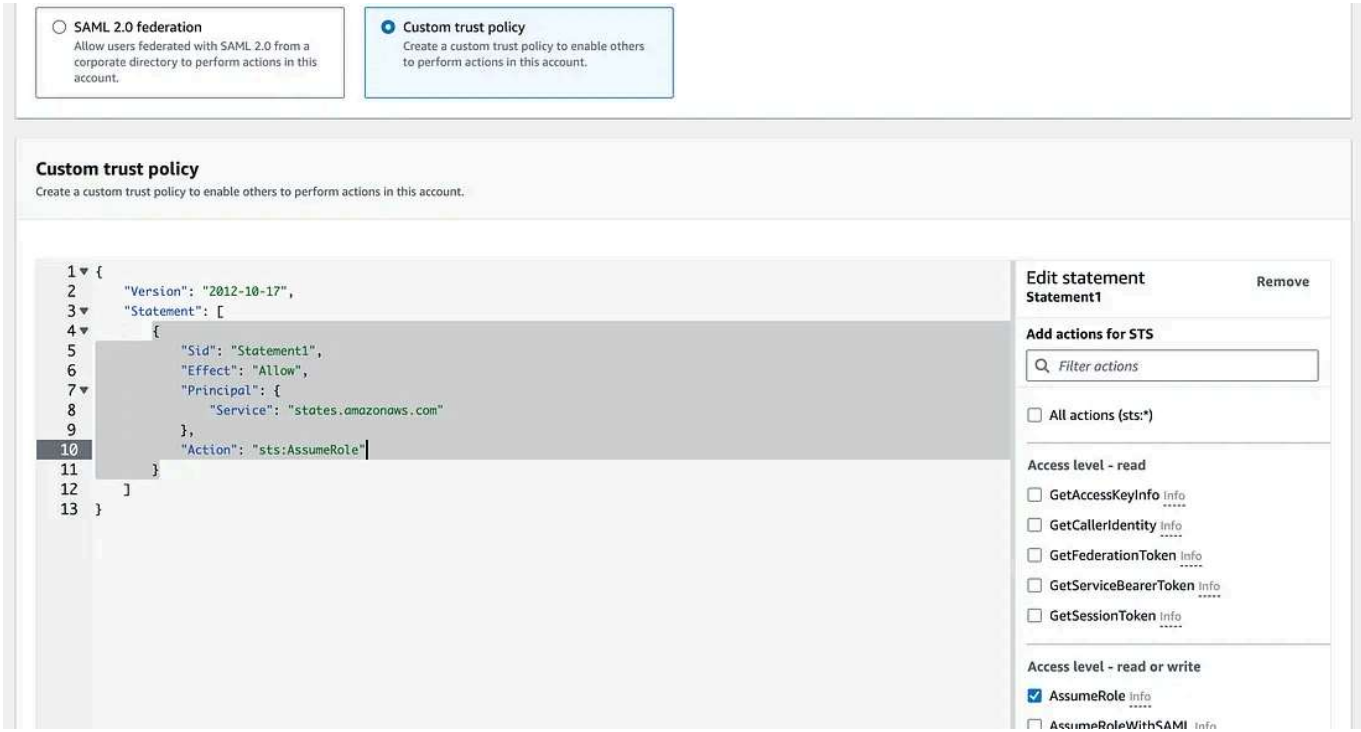
step-functions-policy IAM Policy — 01



step-functions-policy IAM Policy — 02

1. Once the policy is created, navigate to **Roles** page
2. Click **Create role** to create a new IAM role for Step Functions state machine
3. Choose **Custom trust policy** under **Trusted entity type** section and add the following statement and click **Next** button

4. Add the created policy **step-functions-policy**, add a **Role name** (eg: step-functions-role), **Description** and appropriate **Tags** and click **Create role**



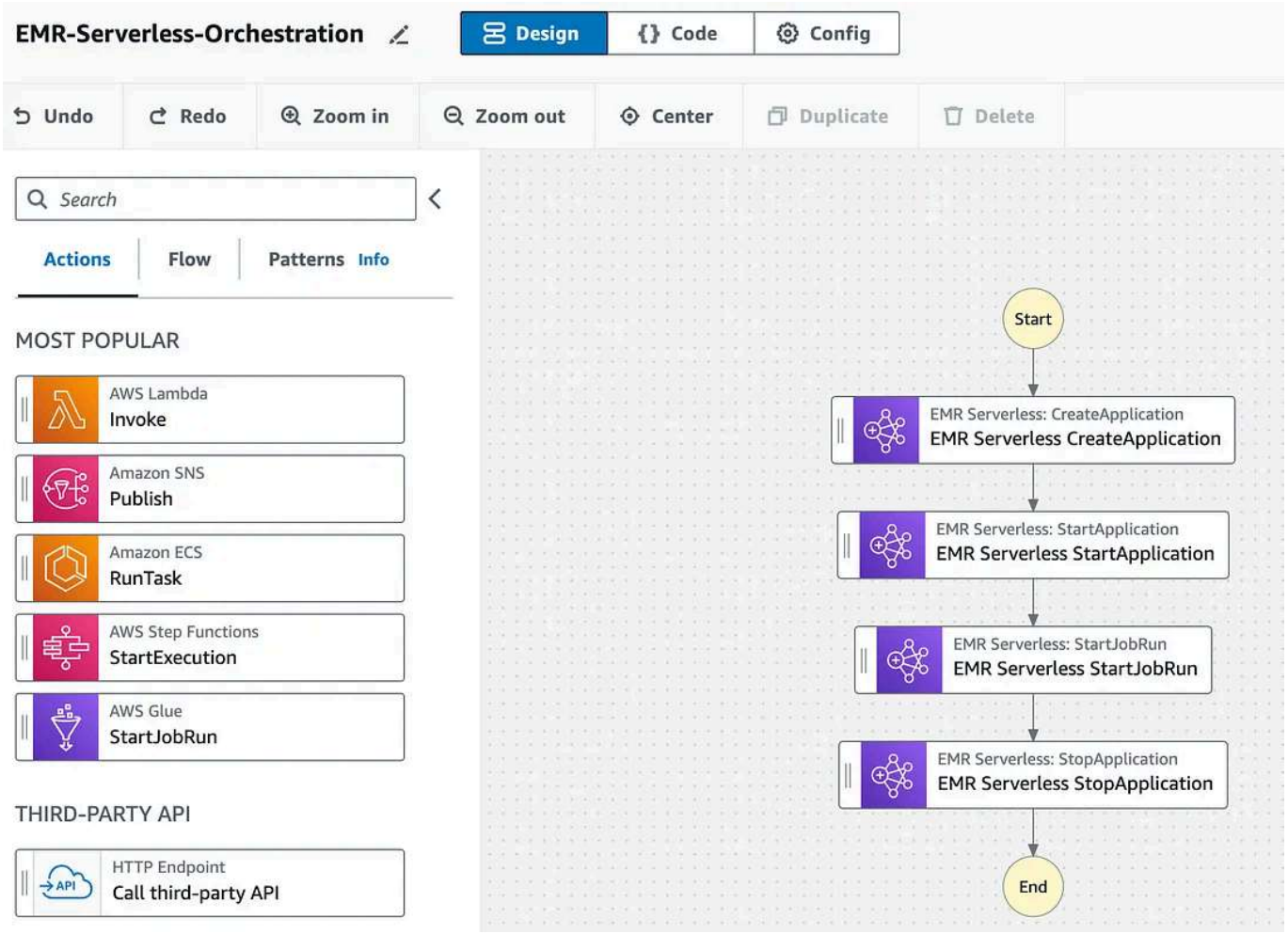
step-functions-role IAM Role

Step 4: Create a Step Functions state machine

The Step Functions state machine can be created in two ways — either directly through the code or through the Step Functions studio graphical interface. In this tutorial, we will create the Step Functions state machine through Workflow Studio.

Follow below steps to create a state machine:

1. In **AWS** console, navigate to **AWS Step Functions** console and click **Create state machine**
2. Choose a blank template, and orchestrate the following tasks in order: **EMR Serverless CreateApplication**, **EMR Serverless StartApplication**, **EMR Serverless StartJobRun** and **EMR Serverless StopApplication**

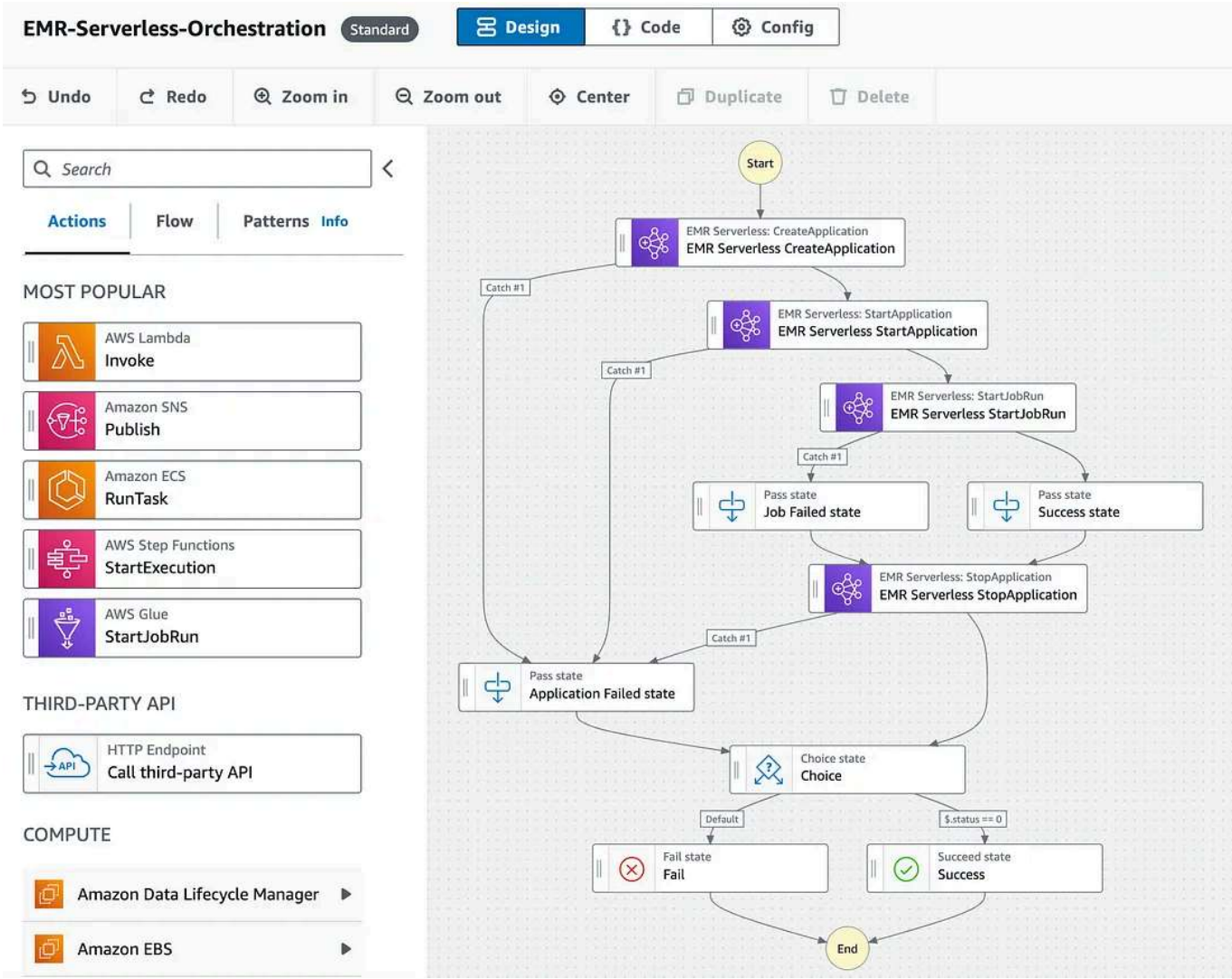


EMR-Serverless-Orchestration State Machine

Handling State Machine Exceptions

You can even handle exceptions for each tasks and set the status of the workflow based on the tasks completion. In the below screenshot, if the state machine is not completed successfully till **EMR Serverless StartJobRun**

task then the workflow will be marked as Failed. If the state machine is completed successfully till **EMR Serverless StartJobRun** task then the workflow will be marked as Success. Based on this status the state machine will be marked success or failed accordingly.



In this deployment, let's add application details only to the tasks for running a Pyspark job. Follow the below steps to add the configurations. Note that each task will have a suffix (.sync) which runs the tasks synchronously. This allows the state machine to wait for each tasks completion.

Task 1: EMR Serverless CreateApplication

Add the following JSON statement to the task **EMR Serverless CreateApplication**

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-serverless:createApplication.sync",
  "Parameters": {
    "Name": "EMR-Serverless-Demo-01",
    "ReleaseLabel": "emr-7.0.0",
    "Type": "SPARK",
  },
  "ResultPath": "$.emr",
  "Next": "EMR Serverless StartApplication"
}
```

You can even add parameters like InitialCapacity, MaximumCapacity, NetworkConfiguration, Tags and so on according to your jobs requirement. For more details related to parameters refer EMR-Serverless API Create Application.

Task 2: EMR Serverless StartApplication

Add the following JSON statement to the task **EMR Serverless StartApplication**

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-serverless:startApplication.sync",
  "Parameters": {
    "ApplicationId.$": "$.emr.ApplicationId"
  },
  "ResultPath": null,
  "Next": "EMR Serverless StartJobRun"
}
```

This task starts the newly created EMR Serverless Application. From the next task, we can submit Pyspark job in the application.

Task 3: EMR Serverless StartJobRun

Add the following JSON statement to the task **EMR Serverless StartJobRun**

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-serverless:startJobRun.sync",
  "Parameters": {
    "ApplicationId.$": "$.emr.ApplicationId",
    "Name": "PySpark Job Run",
    "ExecutionRoleArn": "arn:aws:iam::{aws-account-id}:role/{emr-serverless-role
    "JobDriver": {
      "SparkSubmit": {
        "EntryPoint": "s3://{aws-s3-bucket-name}/process_ds_salaries.py",
        "EntryPointArguments": [],
        "SparkSubmitParameters": "--conf spark.hadoop.hive.metastore.client.fact
      }
    }
  },
  "ResultPath": null,
  "Next": "EMR Serverless StopApplication"
```


This task will submit a PySpark job in the newly created application and wait for its run to complete. For more details related to parameters refer EMR-Serverless API Start Job Run.

Task 4: EMR Serverless StopApplication

Add the following JSON statement to the task **EMR Serverless StopApplication**

```
{
  "Type": "Task",
  "Resource": "arn:aws:states:::emr-serverless:stopApplication.sync",
  "Parameters": {
    "ApplicationId.$": "$.emr.ApplicationId"
  },
  "ResultPath": null,
  "End": true
}
```

Once all the above tasks are complete, this task will stop the EMR serverless application to avoid extra cost.

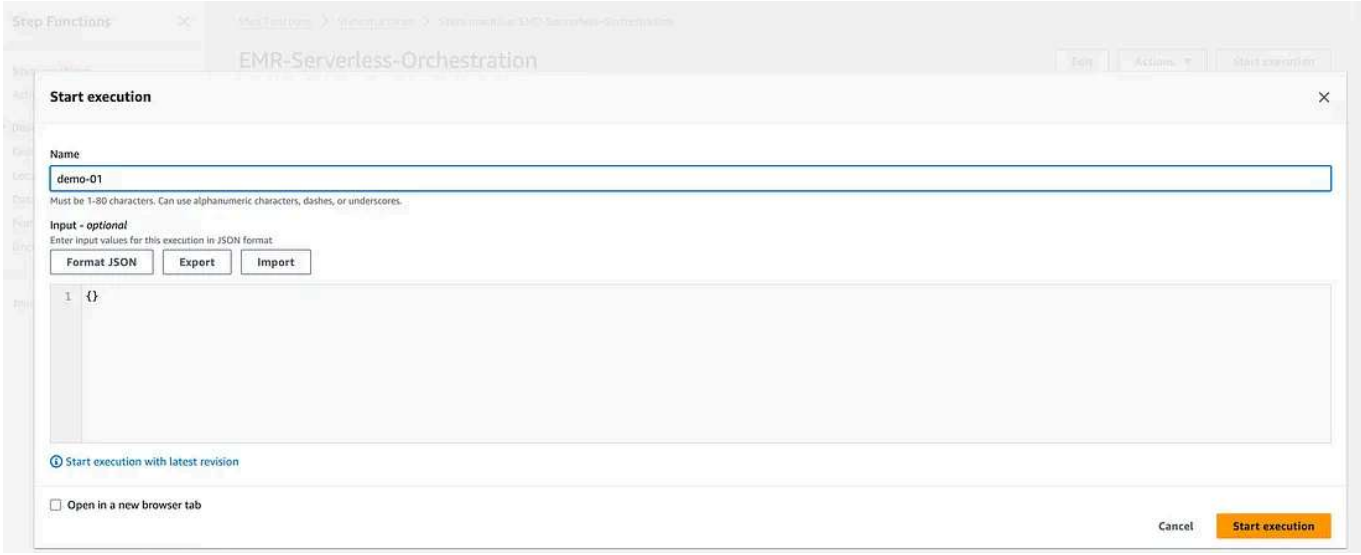
After adding all the above details to the tasks, go to **Config** section and add a name to the state machine (eg: EMR-Serverless-Orchestration) and add the newly created execution role **step-functions-role** and click **Create** button. This will create a new state machine using which we can create and run our EMR Serverless Application.

Workflow Validation

Once all the above deployment steps are completed, we are good to execute the state machine. Follow these steps to execute **EMR-Serverless-Orchestration** state machine.

State Machine Execution

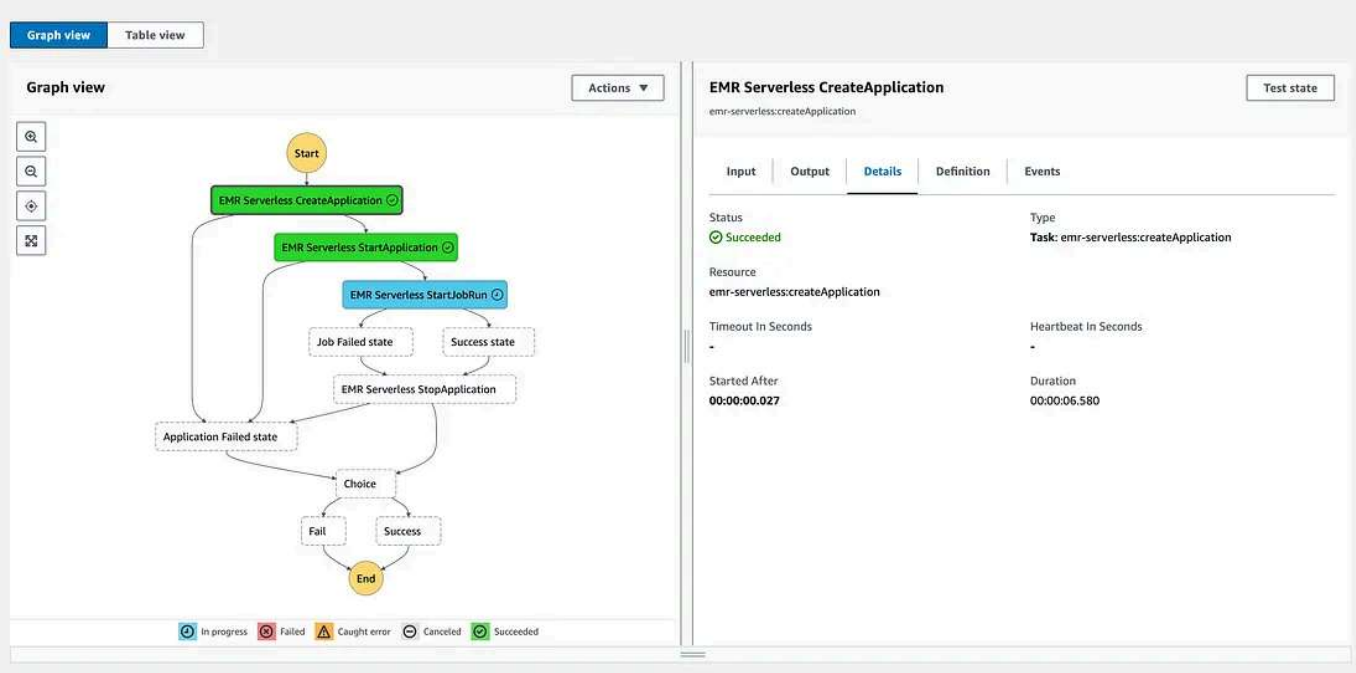
Navigate in to **AWS Step Functions** console and click **EMR-Serverless-Orchestration** state machine. Under **Executions** tab, click **Start execution** with empty JSON statement as input.



state machine execution

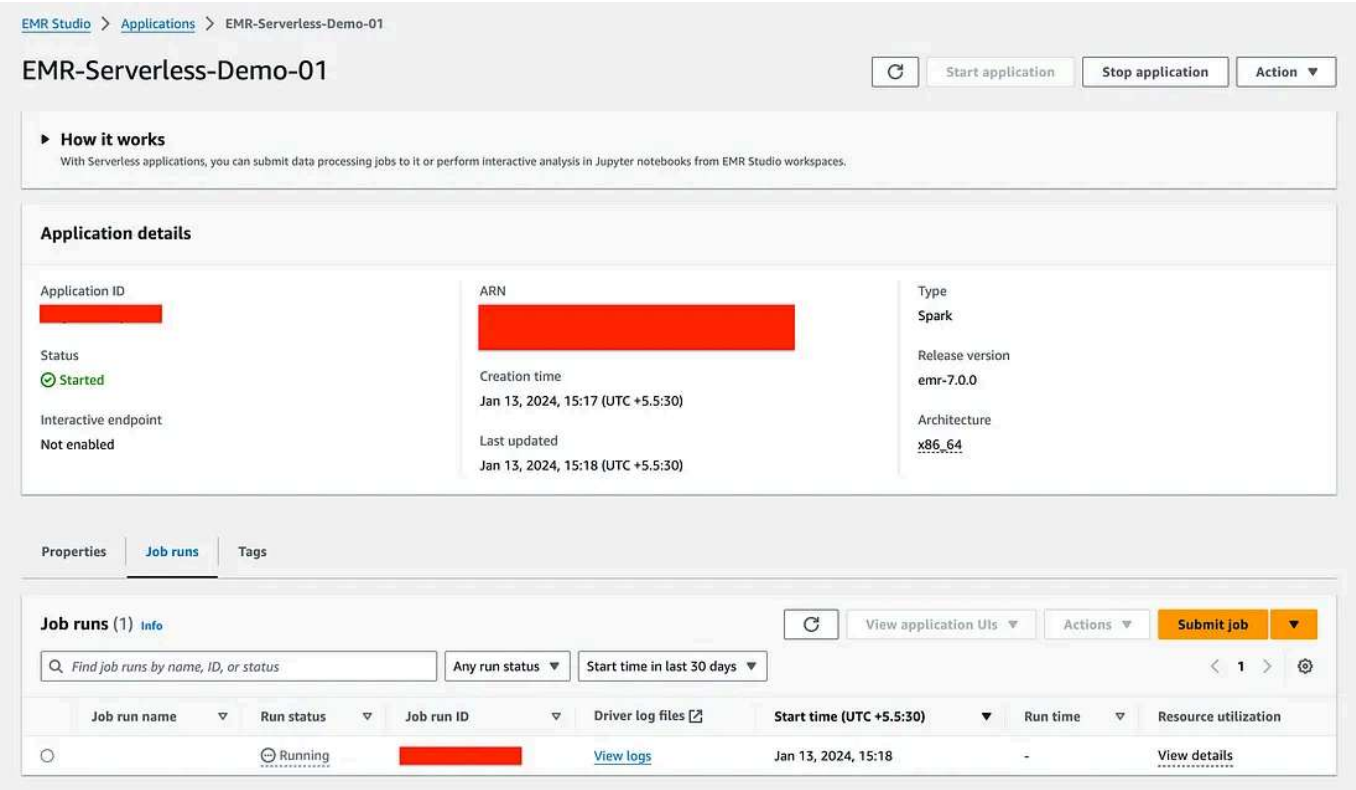
Monitor Application

You can monitor the state machine workflow by navigating into the recent execution and check the status of each completed (or) running tasks.



monitor state machine workflow

Once the EMR Serverless Application is created, navigate into the **EMR** console and open **EMR Serverless** Studio. You will find a new application created with a name **EMR-Serverless-Demo-01.**

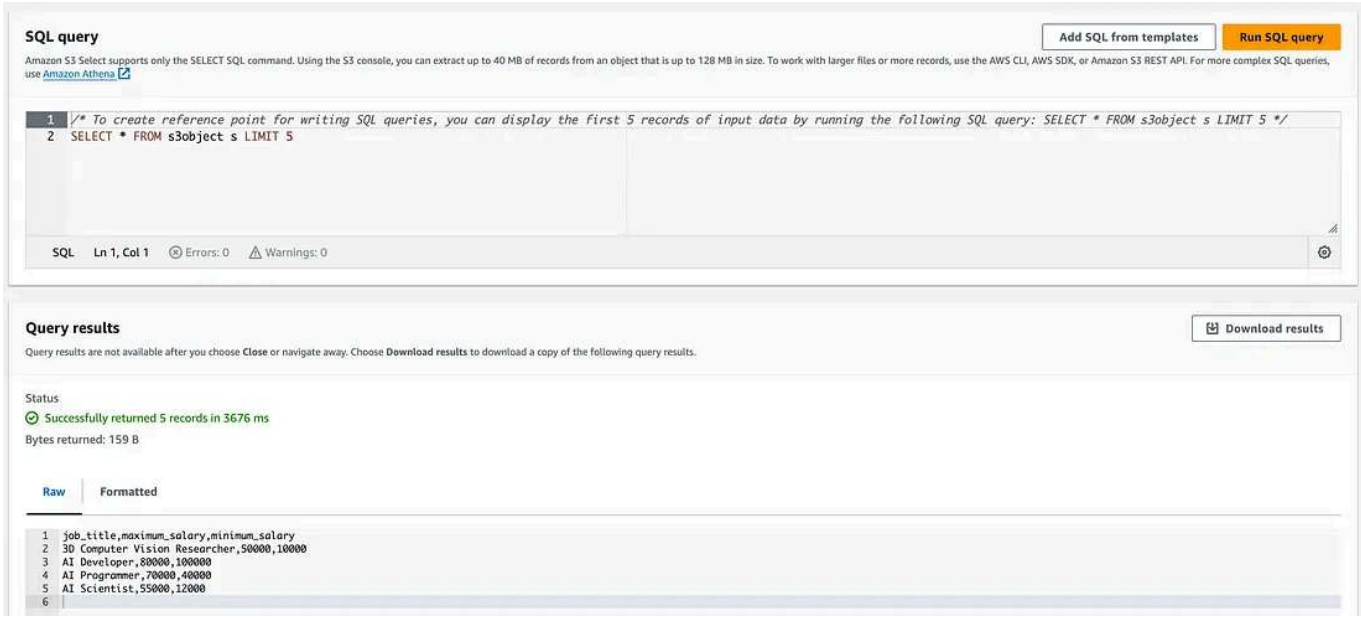


monitor emr serverless application

If the job is failed due to some error, You can debug the errors by clicking **View logs** link under **Driver logs files** tab.

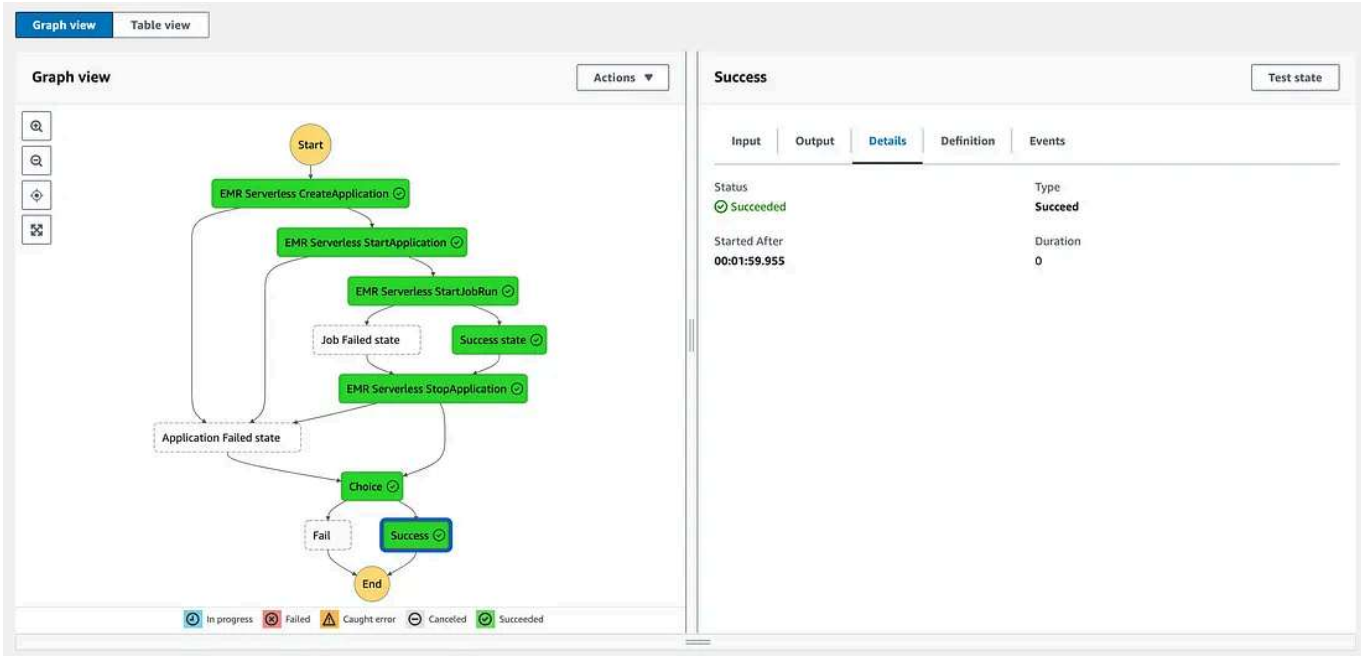
Validation

Once the PySpark job is complete, the result dataset will be stored in the S3 bucket. To view the output, the result dataset can be queried using **Query with S3 select** option under **Actions** tab.



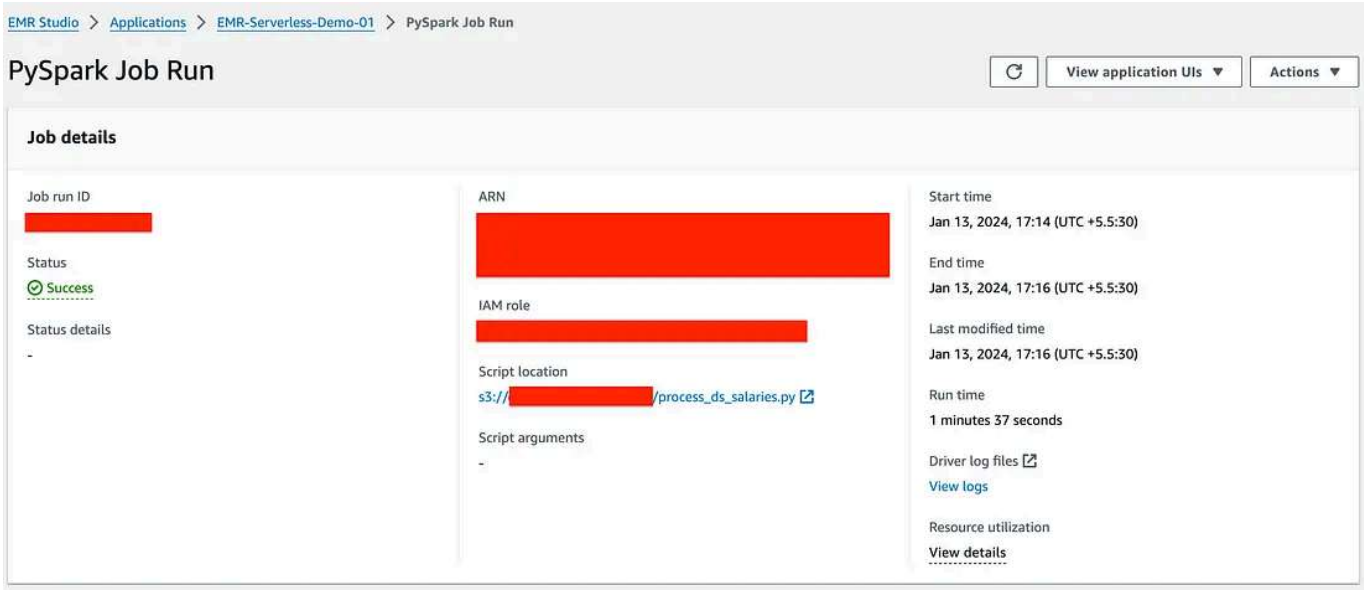
Result

After the completion of **EMR Serverless StartJobRun** task, the **EMR Serverless StopApplication** task will stop the running application to avoid any extra cost. Once all the tasks are completed successfully, the tasks of the state machine will look green in colour.



successful state machine execution

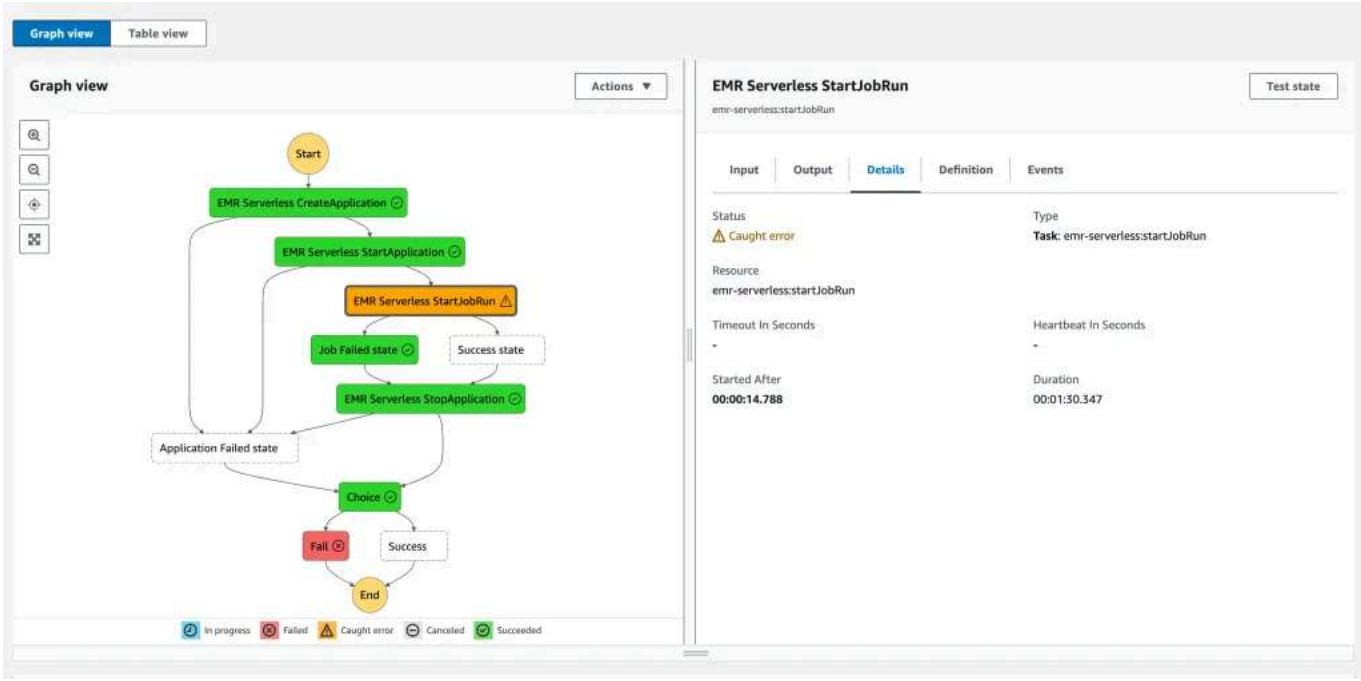
The completed PySpark Job run can be viewed in the EMR Serverless Application



successful emr serverless job run

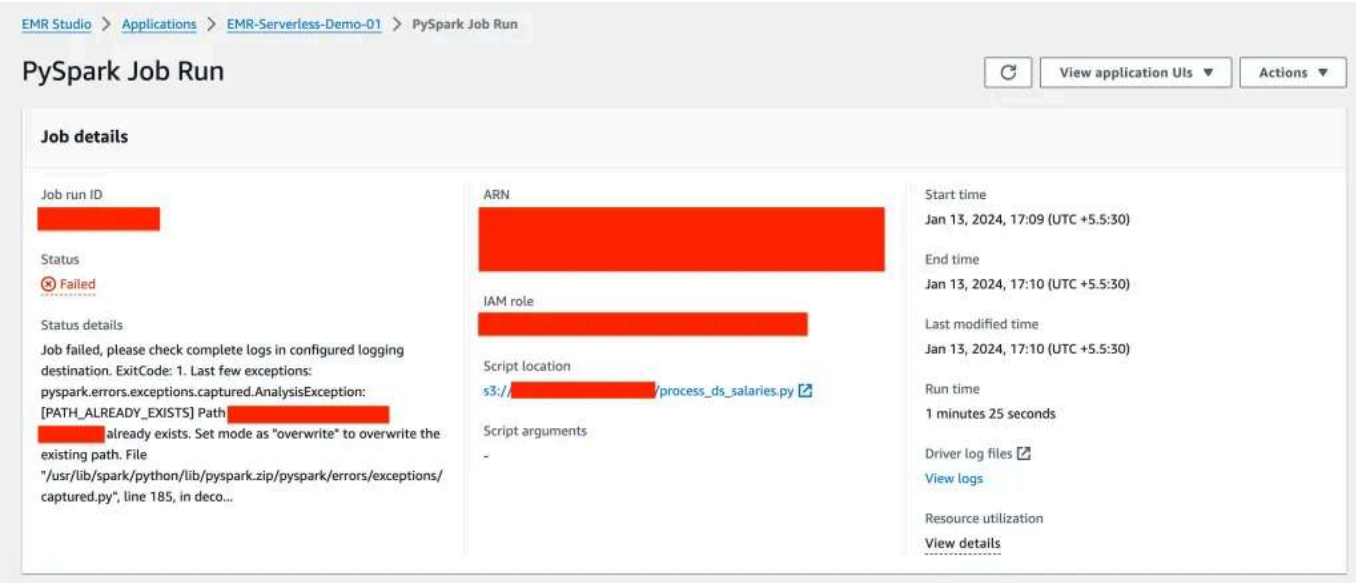
Handling Exceptions

If any of the tasks in state machine is failed, the exceptions will be caught and you can inspect the cause of the error in the state machine workflow.



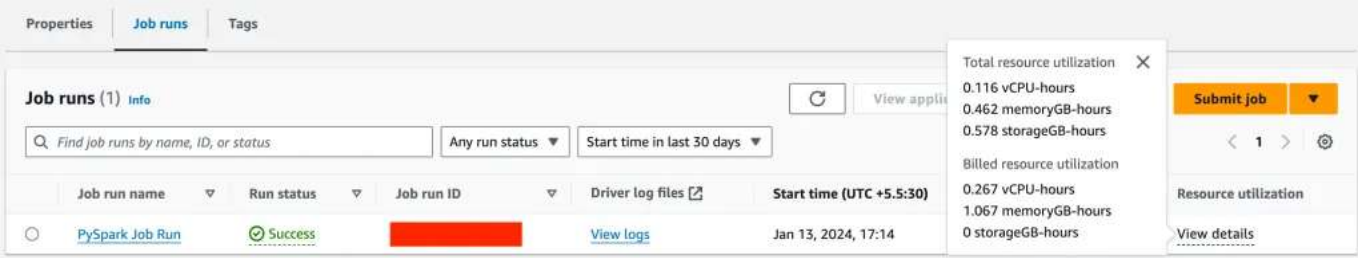
state machine exception handling

You can also check the EMR Serverless application for more detailed error logs in the EMR studio console.



Resource Utilization

Amazon EMR Serverless provides us with a feature which allows to understand how much resources are utilised by each job run. These details can be viewed under the **Resource utilization** tab of each job run.



References

- https://aws.amazon.com/step-functions/
- https://aws.amazon.com/emr/serverless/
- https://aws.amazon.com/emr/
- https://www.kaggle.com/datasets/henryshan/2023-data-scientists-salary
- https://docs.aws.amazon.com/emr-serverless/latest/APIReference/API_CreateApplication.html
- https://docs.aws.amazon.com/emr-serverless/latest/APIReference/API_StartJobRun.html

AWS | Emr Serverless | Aws Step Functions | Serverless Computing | Big Data

Written by Sathish Krishnan

4 Followers · 9 Following

AWS enthusiast and Big Data Developer

No responses yet

Write a response

What are your thoughts?

More from Sathish Krishnan

Sathish Krishnan

AWS S3 static website hosting using Route 53, CloudFront, ACM...

Sathish Krishnan

Install Apache Hadoop on Windows Subsystem for Linux...

AWS S3 website hosting using Route 53, CloudFront and GoDaddy

Jan 13, 2024 7 2

Deploy Apache Hadoop on Windows Subsystem for Linux (WSL) Ubuntu

Mar 30, 2024 74 4

See all from Sathish Krishnan

Recommended from Medium

In Towards Data Engineering by Ritam Mukherjee

Hands-on Cloud : Build a Serverless To-Do List App on AWS

AWS Cloud Architect

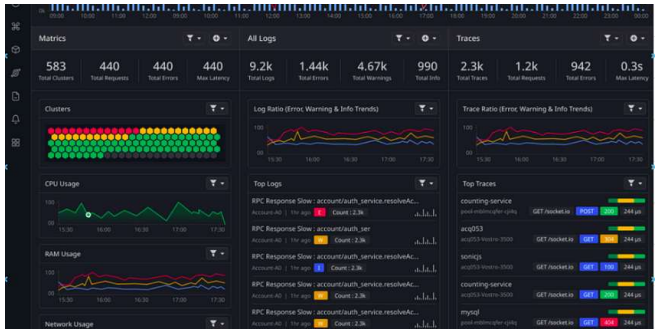
Concurrency in AWS Lambda: A Detailed Guide

Practical aws for cloud-data & backend engineers

Dec 24, 2024 17

What is Concurrency in AWS Lambda?

Apr 21



In AWS in Plain English by Venkatramanan C S

121)11 Cloud Infrastructure Monitoring tools Which are Highl...

During peak usage, cloud infrastructure requires reliable monitoring tools to ensure...

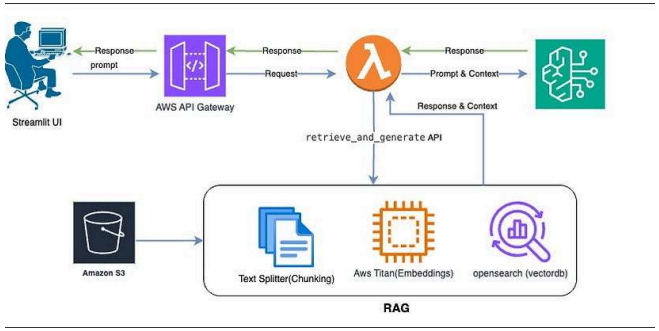
Nov 3, 2024 10

Joud W. Awad

AWS Kinesis Data Streams Deep Dive

In-depth guide to AWS Kinesis Data Streams: learn architecture, consumer models, and...

Nov 5, 2024 289 3



 Souren Stepanyan

Building RealTime Stream Analytics with AWS Lambda...

In today's data-driven world, real-time analytics is crucial for many enterprise...

Nov 22, 2024

 Charan Kumar

Building a Serverless RAG based Q&A app with AWS Bedrock,...

In today's world, knowledge is power, but retrieving the right knowledge at the right...

Jan 10 2

See more recommendations