

Orchestrate Amazon EMR Serverless jobs with AWS Step functions



Sathish Krishnan · [Follow](#)

10 min read · Jan 13, 2024



44



- [Introduction](#)
- [Overview](#)
- [Proposed Solution](#)
- [Prerequisites](#)
- [Architecture](#)
- [Deployment Steps](#)
 - [Step 1: Create S3 Bucket](#)
 - [Step 2: Create PySpark script](#)

- Step 3: Create AWS IAM roles
- Step 4: Create a Step Functions state machine
- Workflow Validation
- Handling Exceptions
- Resource Utilization
- References

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

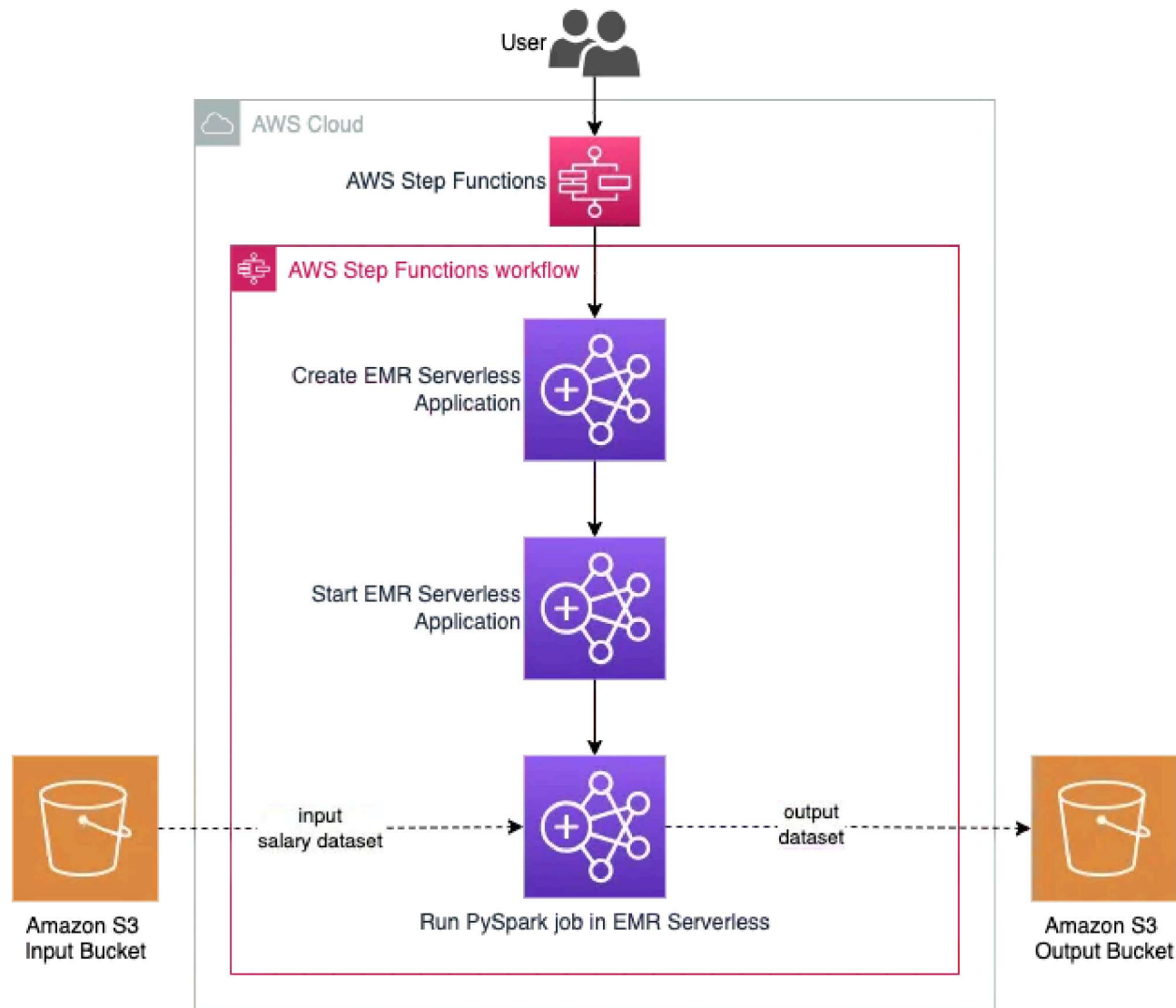
To begin with, make sure the following prerequisites are available

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

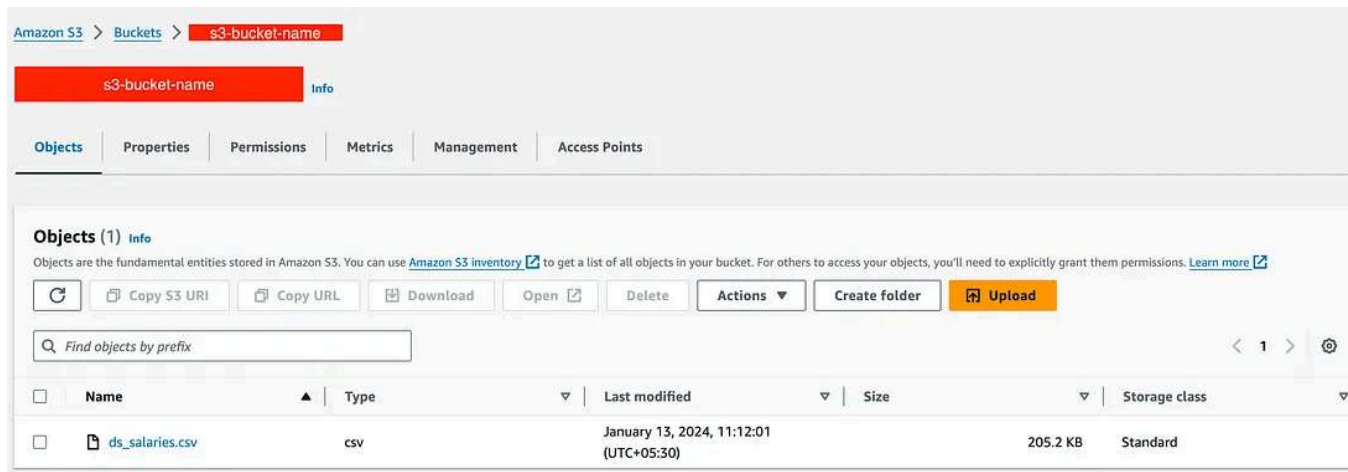
Before beginning this tutorial, ensure that the IAM role being used to deploy has all the relevant permissions to create the required resources as part of

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 x
1  from pyspark.sql import SparkSession
2  from pyspark.sql import functions as func
3
4
5  if __name__ == "__main__":
6      spark = (
7          SparkSession.builder.appName("Demo PySpark Job")
8              .enableHiveSupport()
9              .getOrCreate()
10         )
11
12         df = (
13             spark.read.csv("s3://[redacted]aws-s3-bucket-name/ds_salaries.csv", header=True)
14             .select("job_title", "salary")
15             .groupBy("job_title")
16             .agg(
17                 func.max("salary").alias('maximum_salary'),
18                 func.min("salary").alias('minimum_salary')
19             )
20         )
21         print("dataset aggregation complete")
22         df.write.csv("s3://[redacted]aws-s3-bucket-name/output", header=True)
23         print("dataset write complete")
24
25     spark.stop()
26

```

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

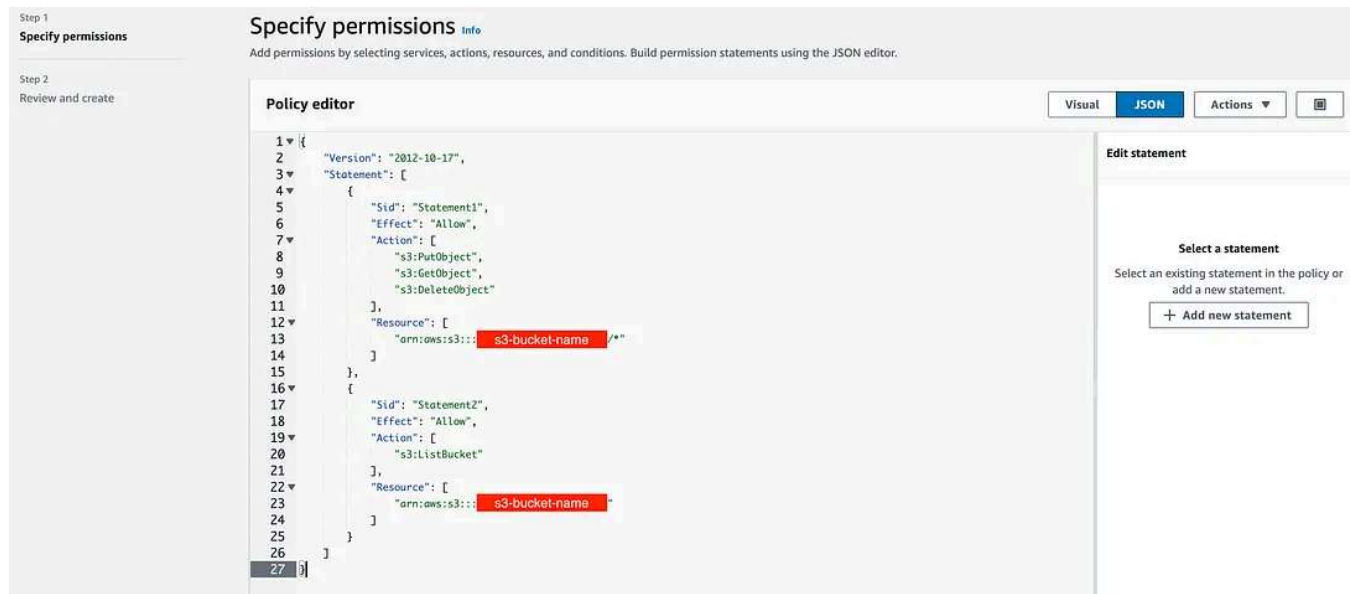
Step 3: Create AWS IAM roles

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**

The screenshot shows the AWS IAM console interface for creating a custom trust policy. At the top, there are two tabs: 'SAML 2.0 federation' (unselected) and 'Custom trust policy' (selected). Below the tabs, the 'Custom trust policy' section is active, displaying a JSON statement in a code editor. The statement is a single statement with 'Effect': 'Allow' and 'Action': 'sts:AssumeRole', where the principal is the EMR Serverless service. To the right of the code editor is a panel titled 'Edit statement' which contains a 'Select a statement' section with a message to select an existing statement or add a new one, and a '+ Add new statement' button.

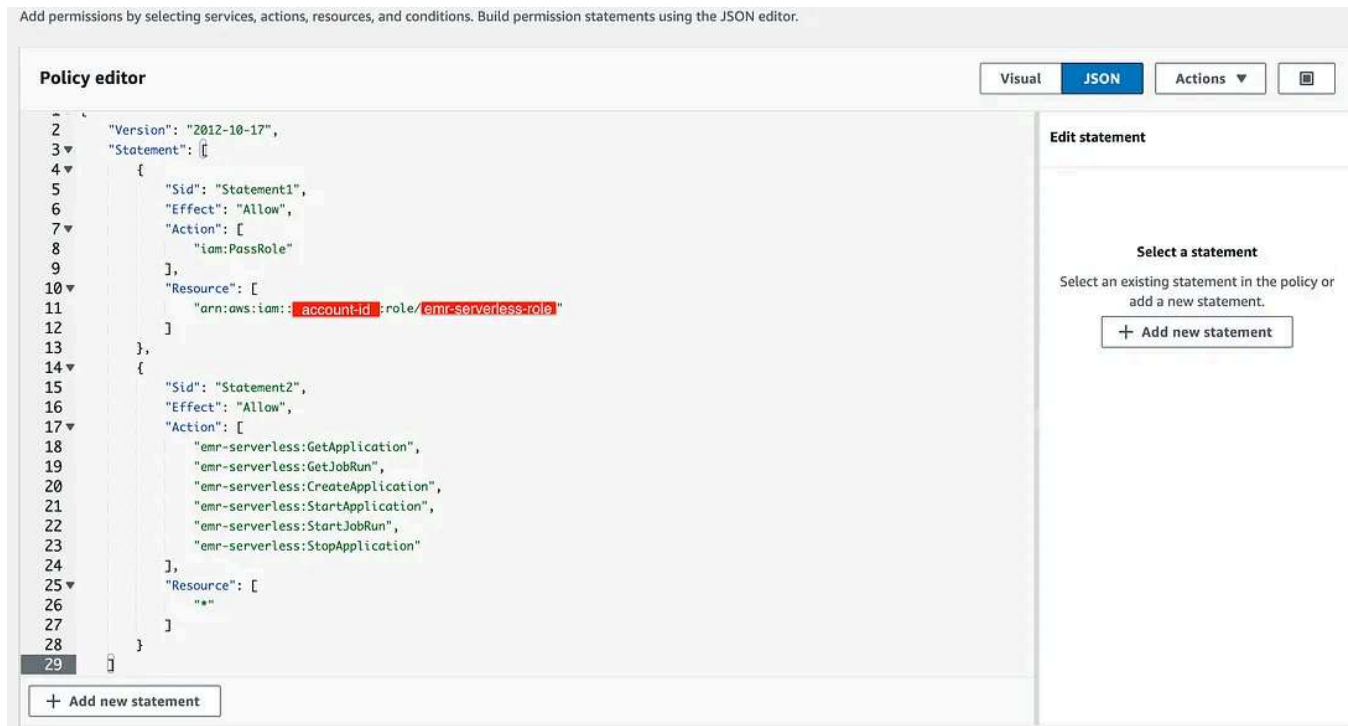
```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Sid": "Statement1",  
6       "Effect": "Allow",  
7       "Principal": {  
8         "Service": "emr-serverless.amazonaws.com"  
9       },  
10      "Action": "sts:AssumeRole"  
11    }  
12  ]  
13 }
```

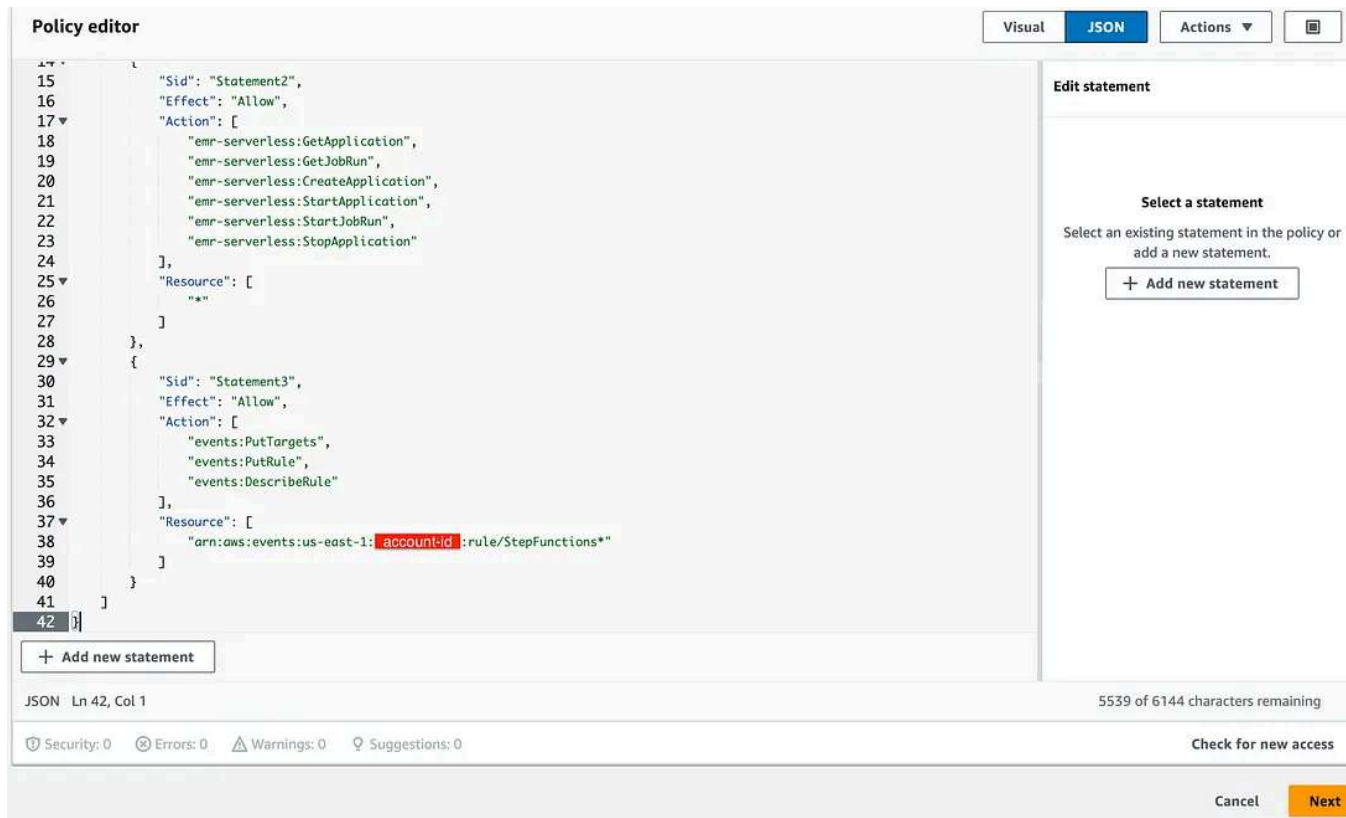
emr-serverless-role IAM Role

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 — 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**

☐ SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

☒ Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Sid": "Statement1",  
6       "Effect": "Allow",  
7       "Principal": {  
8         "Service": "states.amazonaws.com"  
9       },  
10      "Action": "sts:AssumeRole"  
11    }  
12  ]  
13 }
```

Edit statement **Statement1** [Remove](#)

Add actions for STS

☐ All actions (sts:*)

Access level - read

☐ [GetAccessKeyInfo](#) Info

☐ [GetCallerIdentity](#) Info

☐ [GetFederationToken](#) Info

☐ [GetServiceBearerToken](#) Info

☐ [GetSessionToken](#) Info

Access level - read or write

☒ [AssumeRole](#) Info

☐ [AssumeRoleWithSAML](#) Info

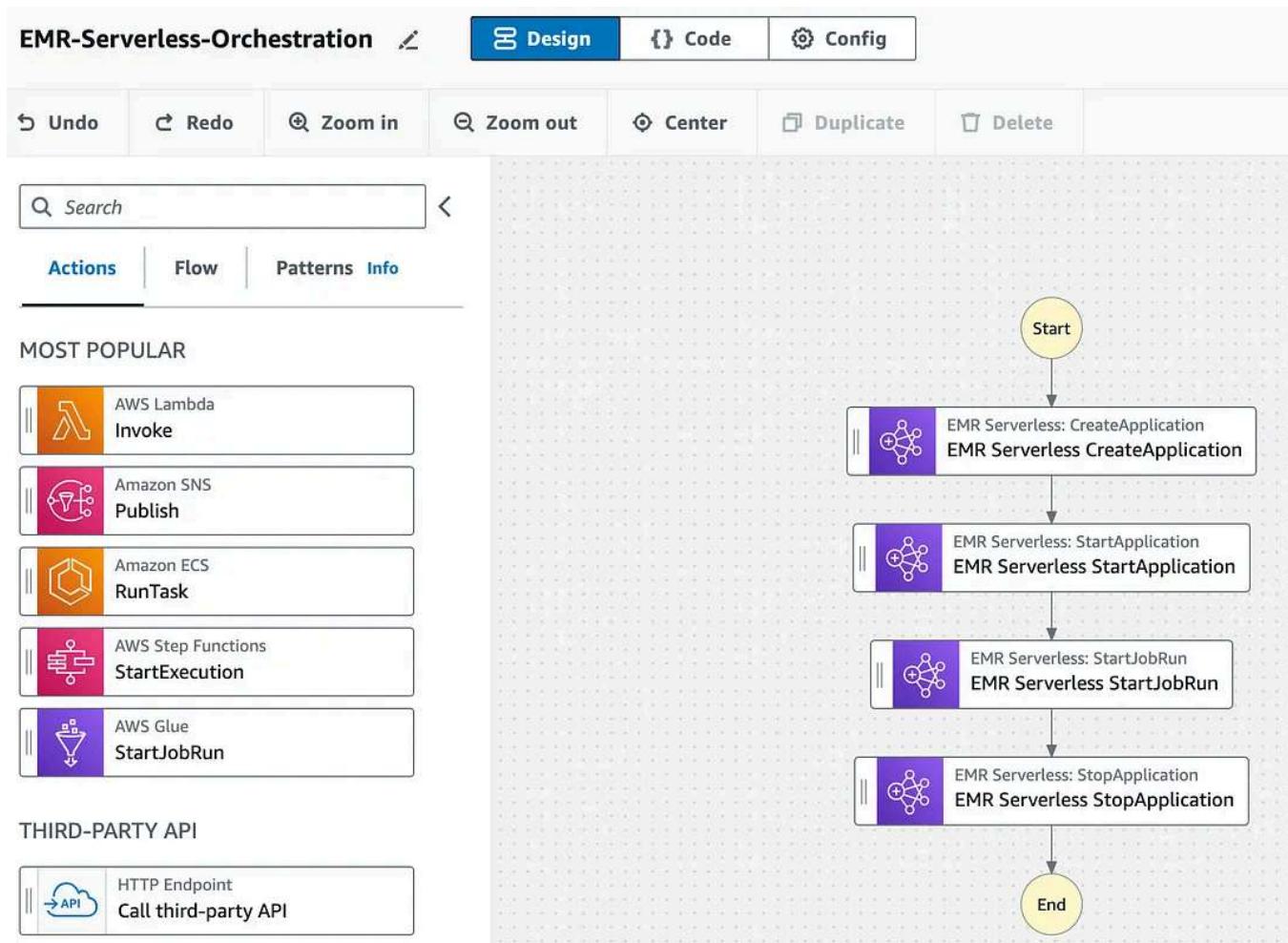
step-functions-role IAM Role

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**

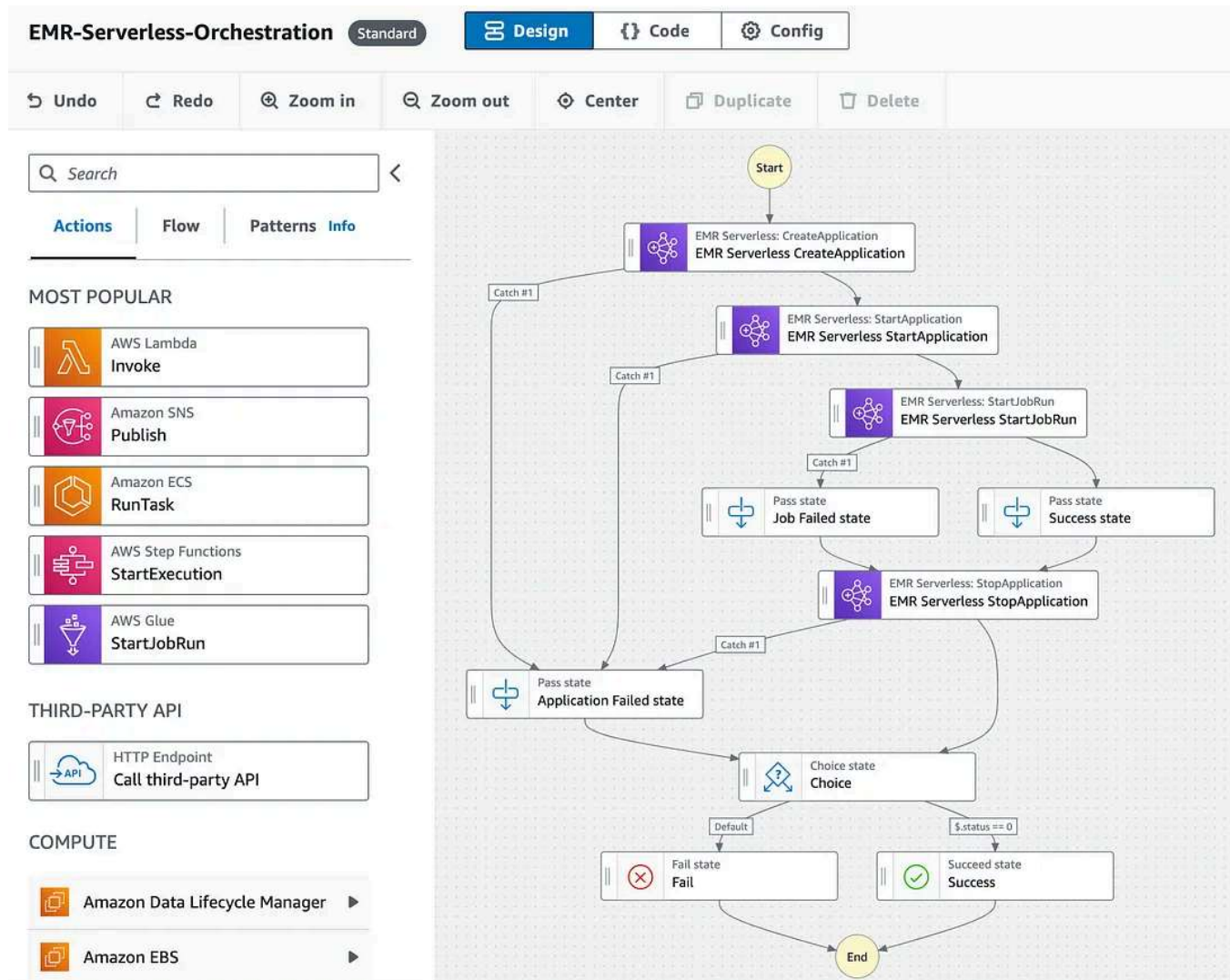


EMR-Serverless-Orchestration State Machine

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.factory.class=com.amazonaws.services.emrserverless.hadoop.HiveMetastoreClientFactory"
      }
    }
  },
  "ResultPath": null,
  "Next": "EMR Serverless StopApplication"
```

Medium

🔍 Search

✍ Write

Sign up

Sign in



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.

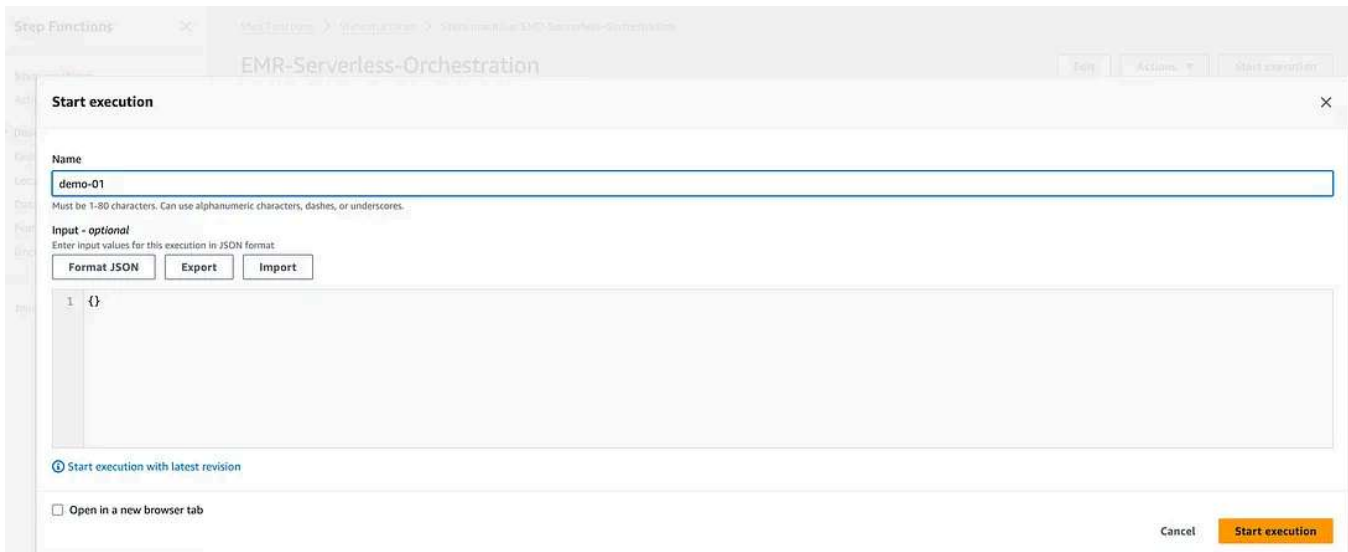
After adding all the above details to the tasks, go to **Config** section and add a name to the state machine (eg: EMR-Serverless-Orchestration) and add the newly created execution role **step-functions-role** and click **Create** button. This will create a new state machine using which we can create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

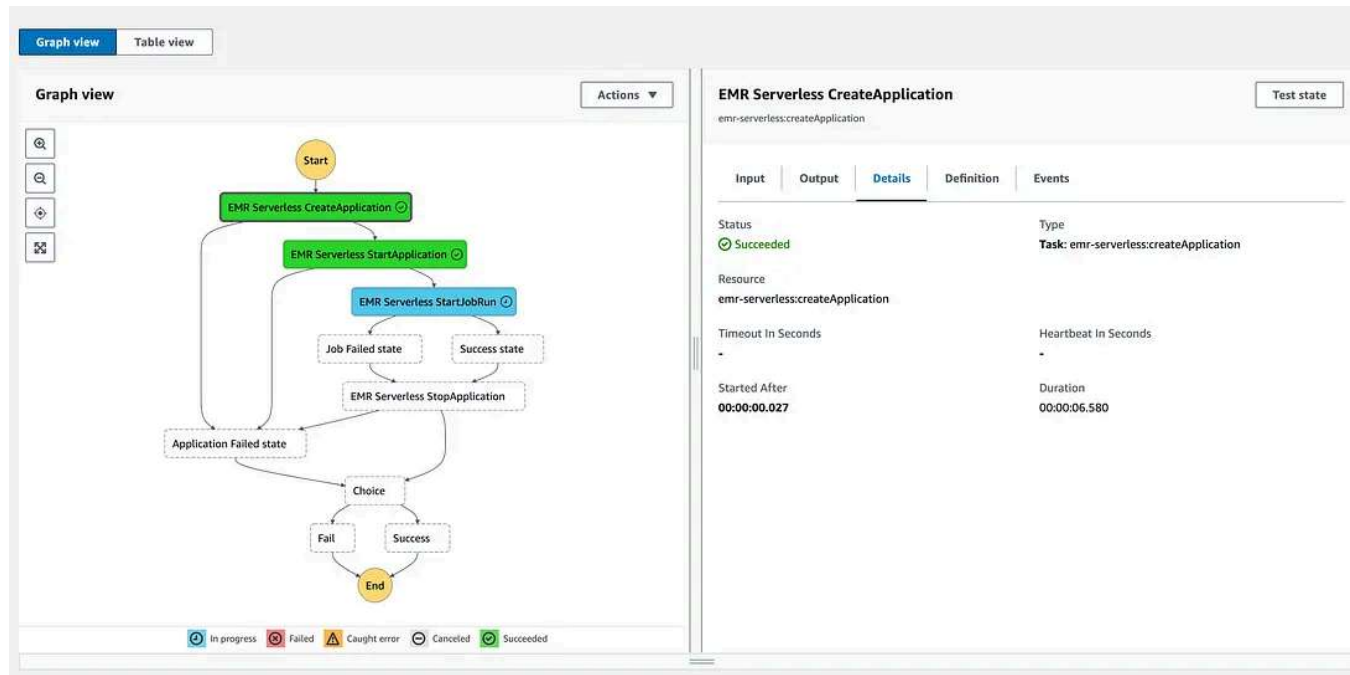


The screenshot shows the AWS Step Functions console interface. The main heading is "EMR-Serverless-Orchestration". Below it, there are buttons for "Edit", "Actions", and "Start execution". A modal dialog titled "Start execution" is open. It has a "Name" field with the value "demo-01" and a note: "Must be 1-80 characters. Can use alphanumeric characters, dashes, or underscores." Below the name field, there is a section for "Input - optional" with the instruction "Enter input values for this execution in JSON format". There are three buttons: "Format JSON", "Expert", and "Import". The "Format JSON" button is selected. The input field shows a single line with the JSON object "{}". At the bottom of the dialog, there is a checkbox labeled "Start execution with latest revision" which is checked, and another checkbox labeled "Open in a new browser tab" which is unchecked. The "Start execution" button is highlighted in orange.

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**.

EMR Studio > Applications > EMR-Serverless-Demo-01

EMR-Serverless-Demo-01

Start application Stop application Action

How it works
With Serverless applications, you can submit data processing jobs to it or perform interactive analysis in Jupyter notebooks from EMR Studio workspaces.

Application details

Application ID [REDACTED]	ARN [REDACTED]	Type Spark
Status Started	Creation time Jan 13, 2024, 15:17 (UTC +5.5:30)	Release version emr-7.0.0
Interactive endpoint Not enabled	Last updated Jan 13, 2024, 15:18 (UTC +5.5:30)	Architecture x86_64

Properties Job runs Tags

Job runs (1) Info View application UIs Actions Submit job

Find job runs by name, ID, or status Any run status Start time in last 30 days

Job run name	Run status	Job run ID	Driver log files	Start time (UTC +5.5:30)	Run time	Resource utilization
[REDACTED]	Running	[REDACTED]	View logs	Jan 13, 2024, 15:18	-	View details

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.

SQL query

Add SQL from templates

Run SQL query

Amazon S3 Select supports only the SELECT SQL command. Using the S3 console, you can extract up to 40 MB of records from an object that is up to 128 MB in size. To work with larger files or more records, use the AWS CLI, AWS SDK, or Amazon S3 REST API. For more complex SQL queries, use [Amazon Athena](#).

1 /* To create reference point for writing SQL queries, you can display the first 5 records of input data by running the following SQL query: SELECT * FROM s3object s LIMIT 5 */

2 SELECT * FROM s3object s LIMIT 5

SQL

Ln 1, Col 1

Errors: 0

Warnings: 0

Query results

Download results

Query results are not available after you choose Close or navigate away. Choose Download results to download a copy of the following query results.

Status

Successfully returned 5 records in 3676 ms

Bytes returned: 159 B

Raw

Formatted

1 job_title,maximum_salary,minimun_salary

2 3D Computer Vision Researcher,50000,100000

3 AI Developer,80000,100000

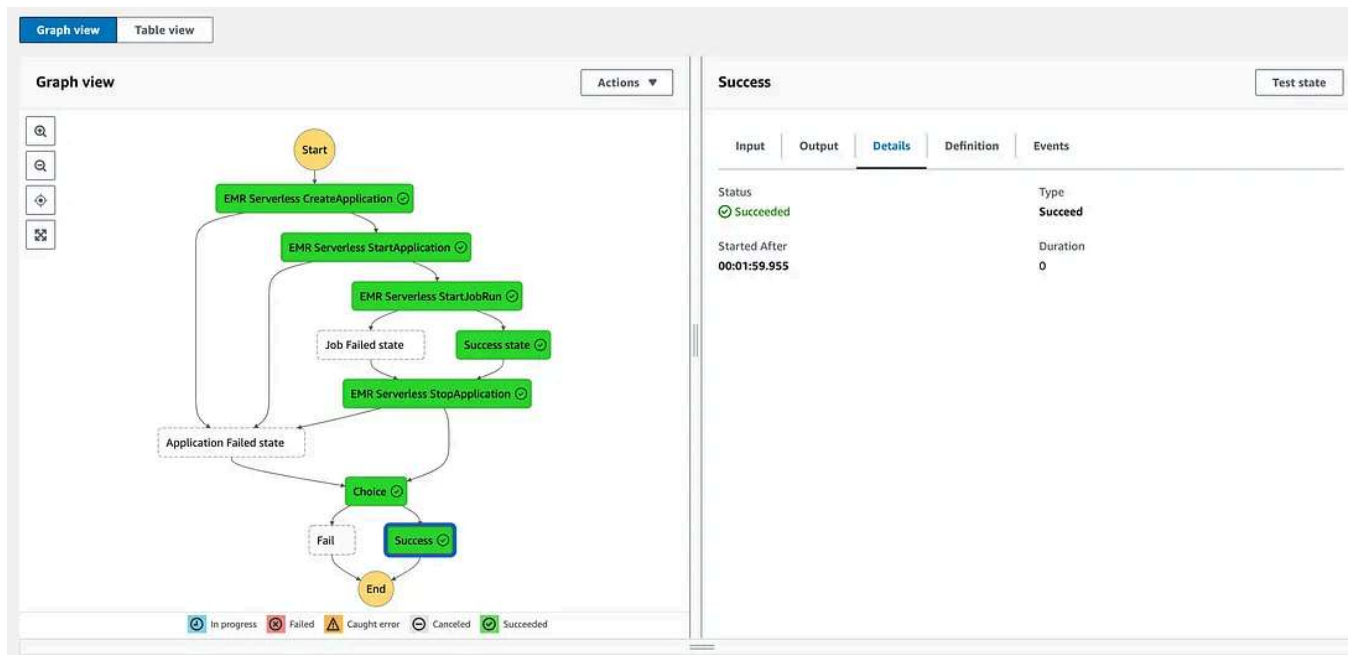
4 AI Programmer,70000,400000

5 AI Scientist,55000,12000

6

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

EMR Studio > Applications > EMR-Serverless-Demo-01 > PySpark Job Run

PySpark Job Run

View application UIs Actions

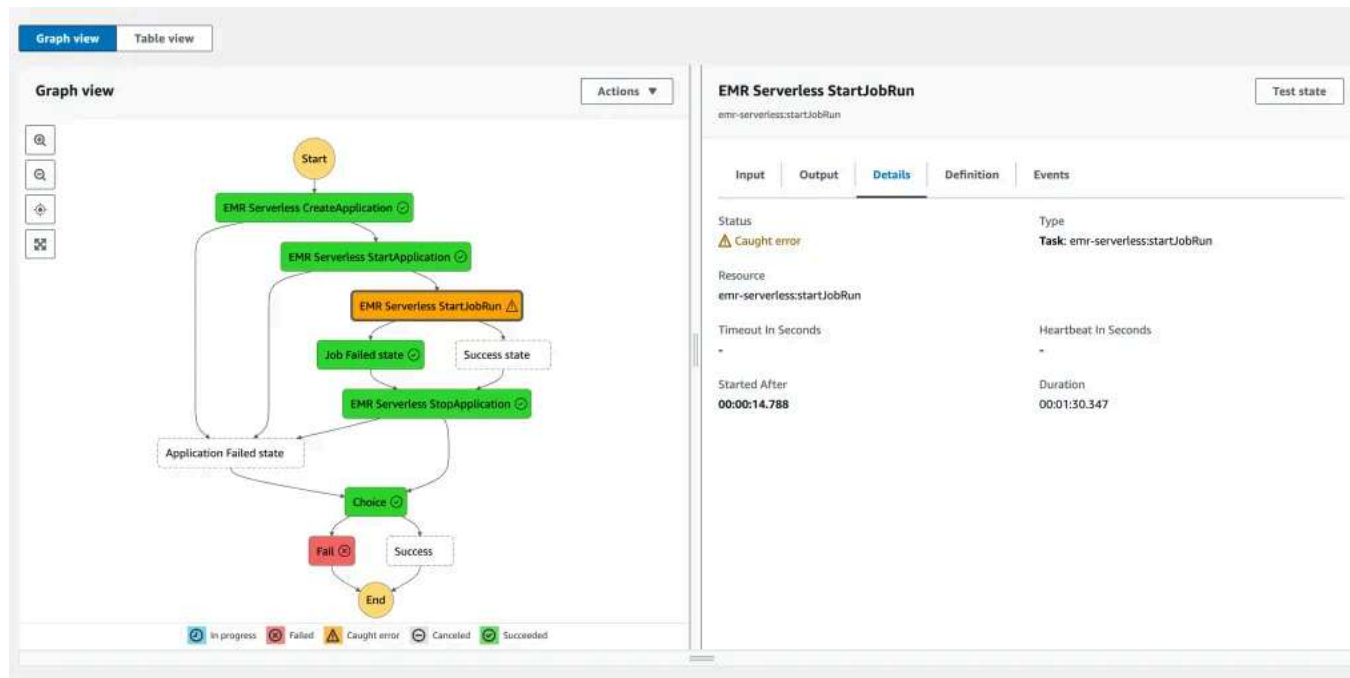
Job details

Job run ID	ARN	Start time
		Jan 13, 2024, 17:14 (UTC +5.5:30)
Status		End time
Success		Jan 13, 2024, 17:16 (UTC +5.5:30)
Status details	IAM role	Last modified time
-		Jan 13, 2024, 17:16 (UTC +5.5:30)
	Script location	Run time
	s3:// /process_ds_salaries.py	1 minutes 37 seconds
	Script arguments	Driver log files
	-	View logs
		Resource utilization
		View details

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.

EMR Studio > Applications > EMR-Serverless-Demo-01 > PySpark Job Run

PySpark Job Run

View application UIs Actions

Job details

Job run ID

Status

Failed

Status details

Job failed, please check complete logs in configured logging destination. ExitCode: 1. Last few exceptions: pyspark.errors.exceptions.captured.AnalysisException: [PATH_ALREADY_EXISTS] Path already exists. Set mode as "overwrite" to overwrite the existing path. File "/usr/lib/spark/python/lib/pyspark.zip/pyspark/errors/exceptions/captured.py", line 185, in deco...

ARN

IAM role

Script location

s3:// /process_ds_salaries.py

Script arguments

-

Start time

Jan 13, 2024, 17:09 (UTC +5.5:30)

End time

Jan 13, 2024, 17:10 (UTC +5.5:30)

Last modified time

Jan 13, 2024, 17:10 (UTC +5.5:30)

Run time

1 minutes 25 seconds

Driver log files

View logs

Resource utilization

View details

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.

Properties Job runs Tags

Job runs (1) Info

Find job runs by name, ID, or status Any run status Start time in last 30 days

Job run name	Run status	Job run ID	Driver log files	Start time (UTC +5.5:30)
PySpark Job Run	Success		View logs	Jan 13, 2024, 17:14

Total resource utilization

- 0.116 vCPU-hours
- 0.462 memoryGB-hours
- 0.578 storageGB-hours

Billed resource utilization

- 0.267 vCPU-hours
- 1.067 memoryGB-hours
- 0 storageGB-hours

Submit Job

Resource utilization

View details

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

Follow



No responses yet



Write a response

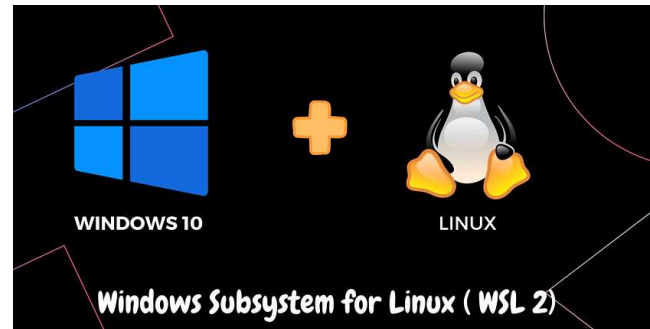
What are your thoughts?

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

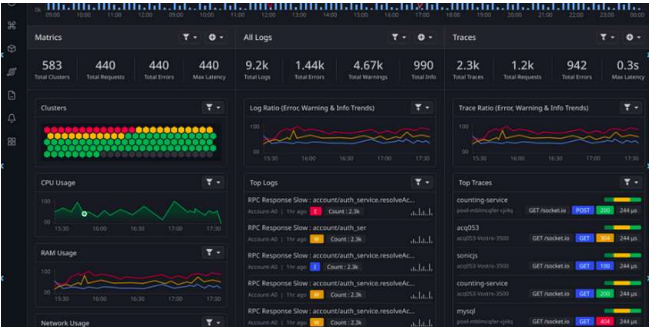


AWS Cloud Architect

Concurrency in AWS Lambda: A Detailed Guide

Practical aws for cloud-data & backend engineers

★ Dec 24, 2024 🖱 17



AWS In AWS in Plain English by Venkatramanan C S

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

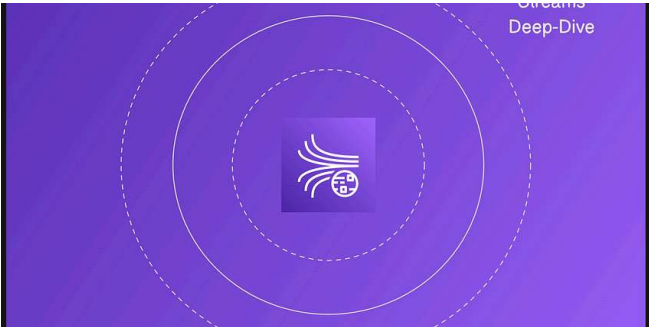
During peak usage, cloud infrastructure requires reliable monitoring tools to ensure...

★ Nov 3, 2024 🖱 10



What is Concurrency in AWS Lambda?

★ Apr 21

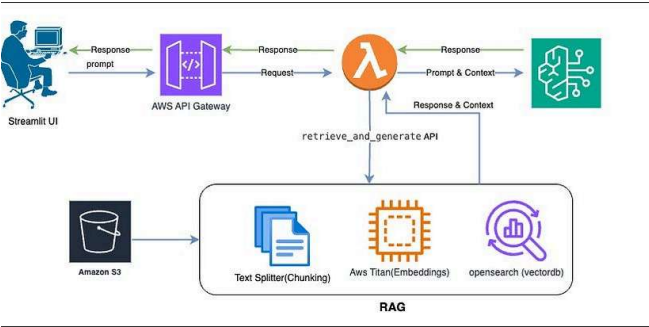


Joud W. Awad

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