

## Serverless workflow using AWS Step Functions to handle errors.

### Tutorial Objectives:

1. Learn to use AWS Step Functions to handle errors in Serverless applications.

### Step 1: Create an AWS Identity and Access Management (IAM) Role

Open the AWS Management Console. When the screen loads, enter your user name and password to get started. Next, Search for IAM, Go to roles and click on create role.

Type of Trusted entity: AWS Service

Choose a use case: **Lambda**

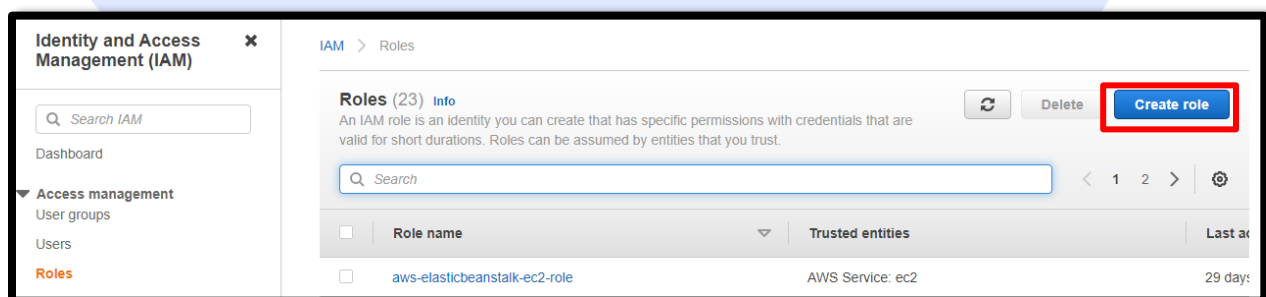
Next: Permissions, Next: tags, Next: Review

Role Name: **lambda\_basic\_execution** and click on Create role.

Similarly create another role for step functions.

In another browser window, navigate to the AWS Management Console and type *IAM* in the search bar. Click IAM to open the service console.

Click Roles, then choose Create Role.



On the Select type of trusted entity page, under AWS service, select Step Functions from the list, and then choose Next: Permissions.

# Cloud Plus Plus Services



Glue	Managed Blockchain	<b>Step Functions</b>
Greengrass	MediaConvert	Storage Gateway
GuardDuty	Migration Hub	Systems Manager
Health Organizational View	Network Firewall	Textract

**Cancel** **Next: Permissions**

On the Attach permissions policy page, choose Next: Tags, Next: Review.

On the Review page, type **step\_functions\_basic\_execution** for Role name and click Create role.

Create role

1 2 3 4

Review

Provide the required information below and review this role before you create it.

Role name\* **step\_functions\_basic\_execution**

Use alphanumeric and '+', '@', '-' characters. Maximum 64 characters.

Role description

Allows Step Functions to access AWS resources on your behalf.

Maximum 1000 characters. Use alphanumeric and '+', '@', '-' characters.

Trusted entities AWS service: states.amazonaws.com

Policies AWSLambdaRole

Permissions boundary Permissions boundary is not set

No tags were added.

\* Required

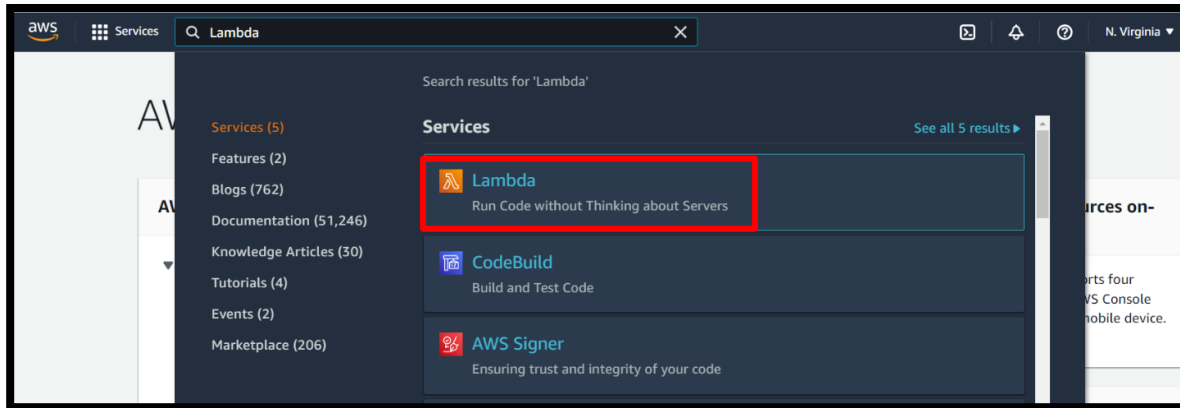
**Cancel** **Previous** **Create role**

Your new IAM role is created and appears in the list beneath the IAM role for your Lambda function.

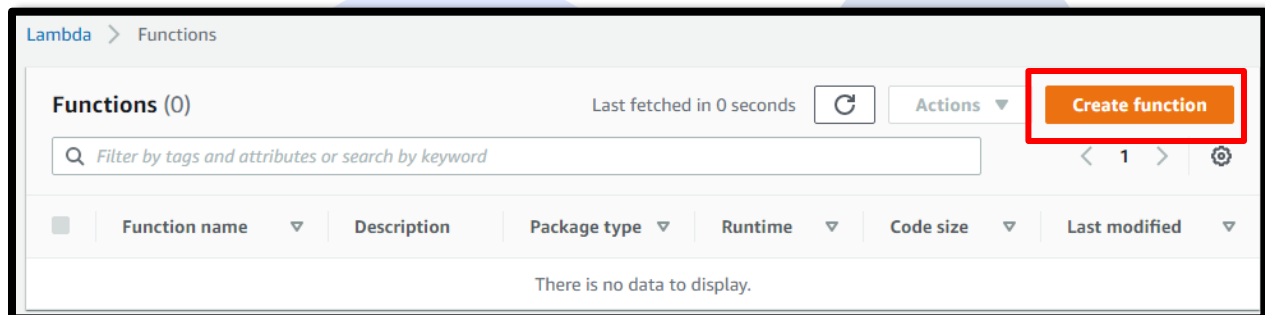
## **Step 2: Create Lambda Function.**

In the new tab, open AWS console and type *Lambda* in the search bar and select Lambda to open the service console.

# Cloud Plus Plus Services



Create a Lambda function by clicking on 'Create Function'.

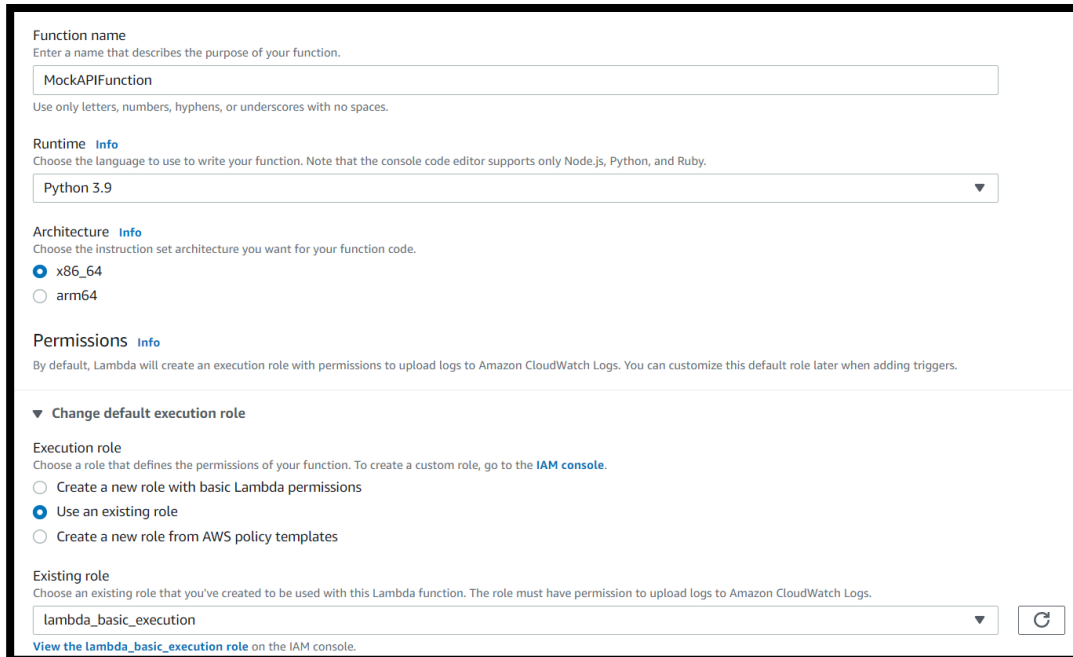


Leave **Author from scratch** selected. Next, configure your first Lambda function as follows:

- For Name, type **MockAPIFunction**.
- For Runtime, choose **Python 3.9**.

In Change default execution role

- For Role, select use an existing role.  
Existing role: **lambda\_basic\_execution**



**Function name**  
Enter a name that describes the purpose of your function.  
  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

**Architecture** [Info](#)  
Choose the instruction set architecture you want for your function code.  
☒ x86\_64  
☐ arm64

**Permissions** [Info](#)  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

**Execution role**  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).  
☐ Create a new role with basic Lambda permissions  
☒ Use an existing role  
☐ Create a new role from AWS policy templates

**Existing role**  
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.  
  
[View the lambda\\_basic\\_execution role](#) on the IAM console.

Click Create function.

On the *MockAPIFunction* screen, scroll down to the Function code section. In the code window, replace all of the code with the following, and choose Save.

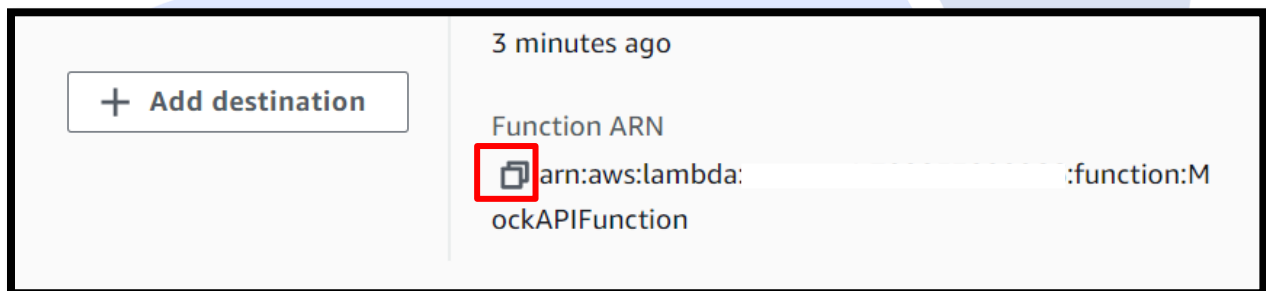
```
class TooManyRequestsException(Exception): pass
class ServerUnavailableException(Exception): pass
class UnknownException(Exception): pass

def lambda_handler(event, context):
    statuscode = event["statuscode"]
    if statuscode == "429":
        raise TooManyRequestsException('429 Too Many Requests')
    elif statuscode == "503":
        raise ServerUnavailableException('503 Server Unavailable')
    elif statuscode == "200":
        return '200 OK'
    else:
        raise UnknownException('Unknown error')
```

# Cloud Plus Plus Services



Once the Lambda function is created, scroll to the top of the window and note its Amazon Resource Name (ARN).



## Step 3. Create a Step Functions State Machine

Open the AWS Step Function console, Click on Create a state machine, select **Author from scratch**,

state machine name: **MyAPIStateMachine**, and then select I will use an existing role.

Application integration

## AWS Step Functions

Assemble functions into business-critical applications

AWS Step Functions is a serverless function orchestrator that makes it easy to sequence AWS Lambda functions and multiple AWS services into business-critical applications. Through its visual interface, you can create and run a series of checkpointed and event-driven workflows that maintain the application state.


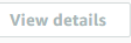

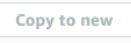


**Get started**


Run a Hello World workflow in just a few clicks.


**Get started**

Step Functions > State machines

**State machines (0)**  
Execution counts are based on the most recent 1000 executions

Any type < 1 > 

Name	Type	Creation date	Status	Logs	Total	Running	Succeeded	Failed	Timed out
No state machines									
									


**Choose authoring method: Write your workflow in code**

**Type: Standard**

**Define state machine**

Design your workflow visually ☐

Drag and drop your workflow together with Step Functions Workflow Studio.



**Write your workflow in code ☒**

Author your workflow using Amazon States Language. You can generate code snippets to easily build out your workflow steps.

Run a sample project ☐

Deploy and run a fully functioning sample project in minutes using CloudFormation.

Replace the contents of the State machine definition section with the following code:

```
{
```

```
"Comment": "An example of using retry and catch to handle API responses",
```

```
"StartAt": "Call API",
```

```
"States": {  
  "Call API": {  
    "Type": "Task",  
    "Resource": "lambda function arn paste here",  
    "Next" : "OK",  
    "Comment": "Catch a 429 (Too many requests) API exception, and resubmit  
the failed request in a rate-limiting fashion.",  
    "Retry" : [ {  
      "ErrorEquals": [ "TooManyRequestsException" ],  
      "IntervalSeconds": 1,  
      "MaxAttempts": 2  
    } ],  
    "Catch": [  
      {  
        "ErrorEquals": ["TooManyRequestsException"],  
        "Next": "Wait and Try Later"  
      }, {  
        "ErrorEquals": ["ServerUnavailableException"],  
        "Next": "Server Unavailable"  
      }, {  
        "ErrorEquals": ["States.ALL"],  
        "Next": "Catch All"  
      }  
    ]  
  },  
  "Wait and Try Later": {
```

```
"Type": "Wait",  
"Seconds" : 1,  
"Next" : "Change to 200"  
},  
"Server Unavailable": {  
"Type": "Fail",  
"Error": "ServerUnavailable",  
"Cause": "The server is currently unable to handle the request."  
},  
"Catch All": {  
"Type": "Fail",  
"Cause": "Unknown error!",  
"Error": "An error of unknown type occurred"  
},  
"Change to 200": {  
"Type": "Pass",  
"Result": {"statusCode" : "200"} ,  
"Next": "Call API"  
},  
"OK": {  
"Type": "Pass",  
"Result": "The request has succeeded.",  
"End": true  
}  
}
```

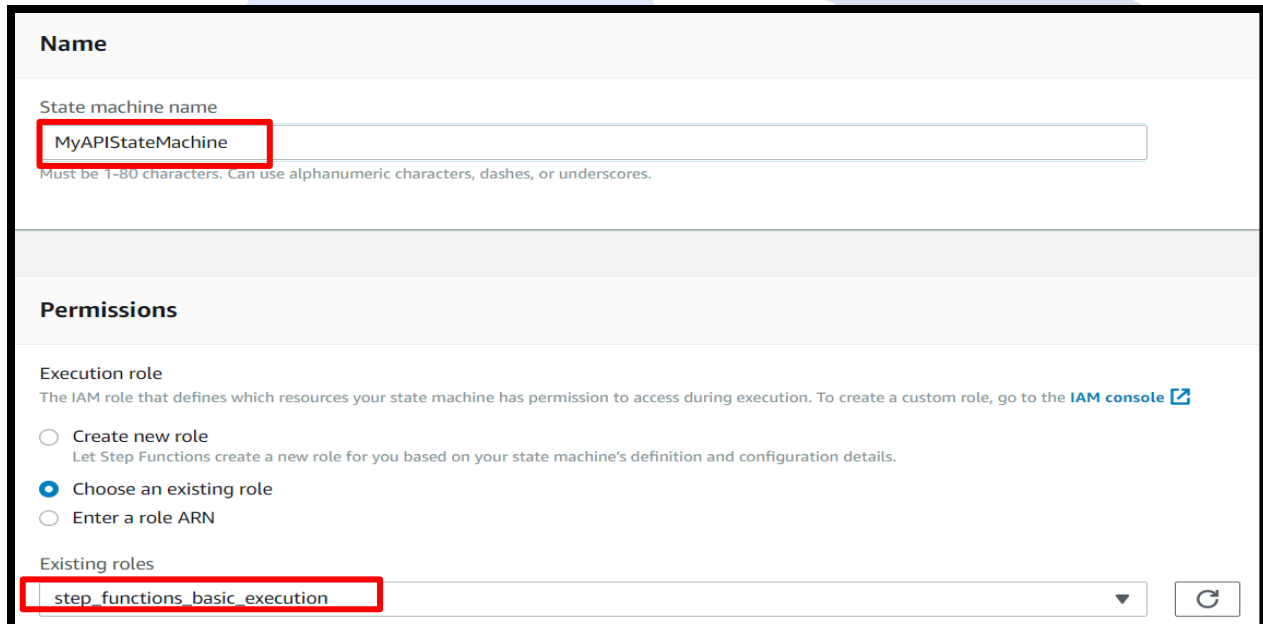


c. Find the "Resource" line in the "Call API" Task state (line 7). To update this ARN to the ARN of the mock API Lambda function you just created, click on the ARN text and then select the ARN from the list.

```
7  "Resource":  
8    "a",  
9    "Next" : "OK",  
    "Comment": "Catch a 429 (Too many requests) API exception  
    and resubmit the failed request in a rate-limiting fashion.",
```

Click the refresh button beside the visual workflow pane to have Step Functions. Review the visual workflow, click Create state machine.

Name: **MyAPIStateMachine** and then select I will use an existing role.



**Name**

State machine name

MyAPIStateMachine

Must be 1-80 characters. Can use alphanumeric characters, dashes, or underscores.

**Permissions**

Execution role

The IAM role that defines which resources your state machine has permission to access during execution. To create a custom role, go to the [IAM console](#).

☐ Create new role  
Let Step Functions create a new role for you based on your state machine's definition and configuration details.

☒ Choose an existing role

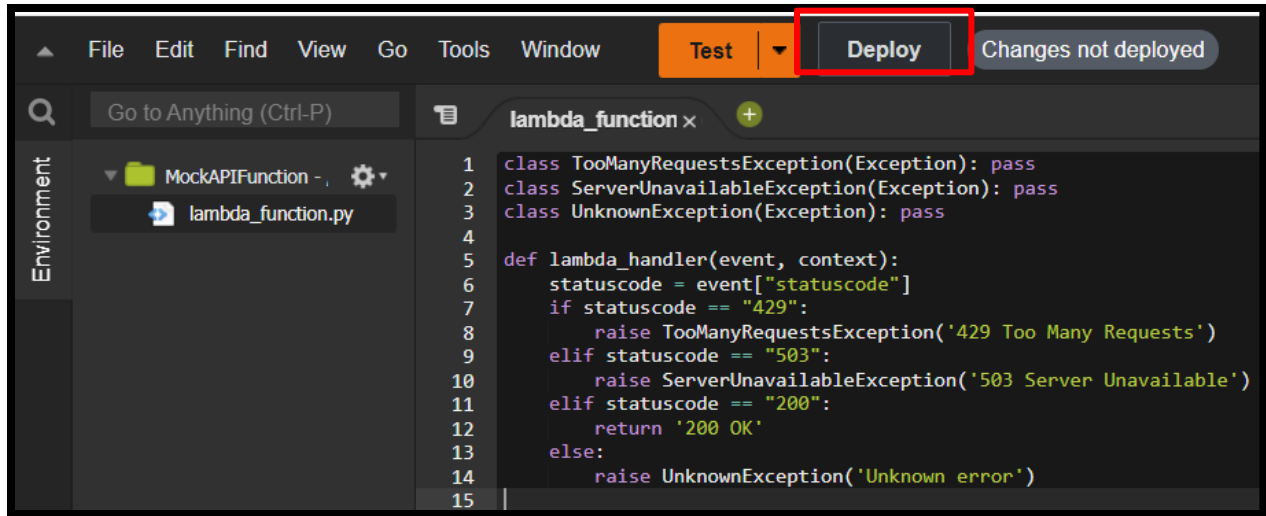
☐ Enter a role ARN

Existing roles

step\_functions\_basic\_execution

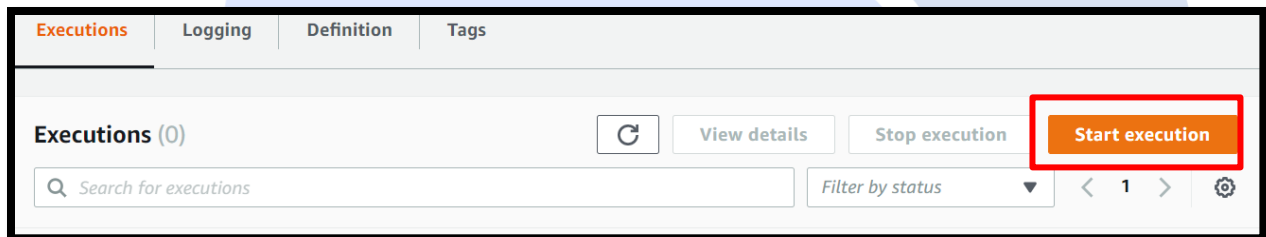
Create State Machines.

Also **Deploy** lambda function.



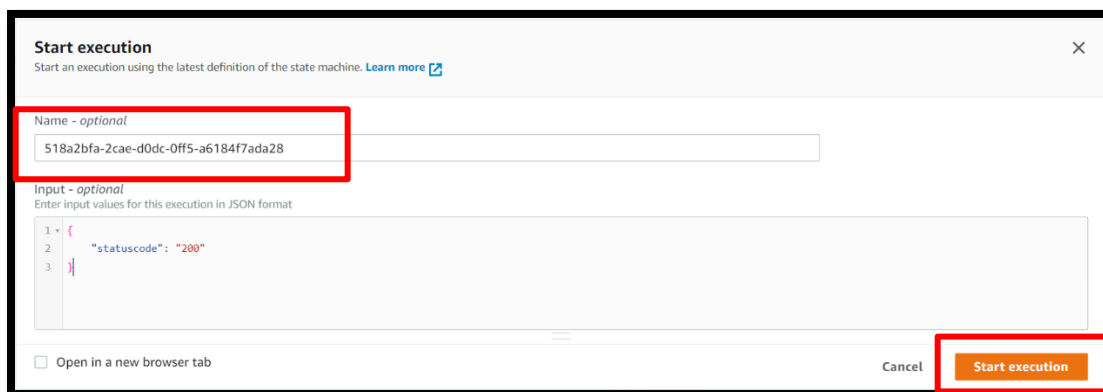
## Step 4. Test your Error Handling Workflow

Click on Start execution.



A new execution dialog box appears, Replace the existing text with the code below, then choose Start execution:

```
{  
  "statusCode": "200"  
}
```



# Cloud Plus Plus Services

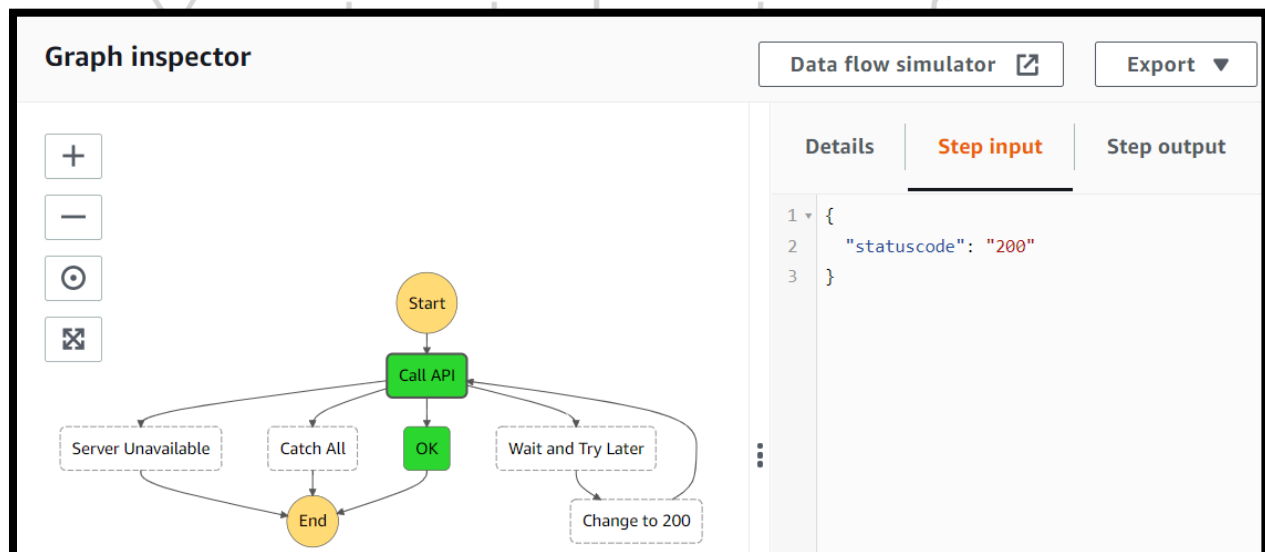


On the Execution details screen, click Input to see the input. Next, click Output to view the result of state machine execution. You can see that the workflow interpreted statuscode 200 as a successful API call.

Details	Execution input	Execution output	Definition
1 ▾	{		
2	"statusCode": "200"		
3	}		

Details	Execution input	Execution output	Definition
1	"The request has succeeded."		

This Task state successfully invoked mock API Lambda function with the input you provided, and captured the output of that Lambda function, "200 OK".

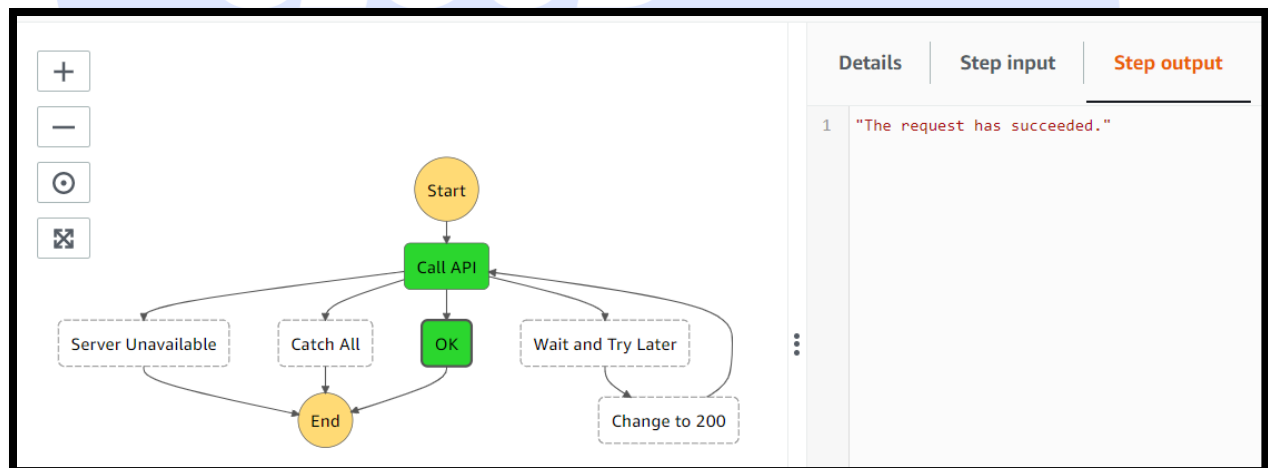
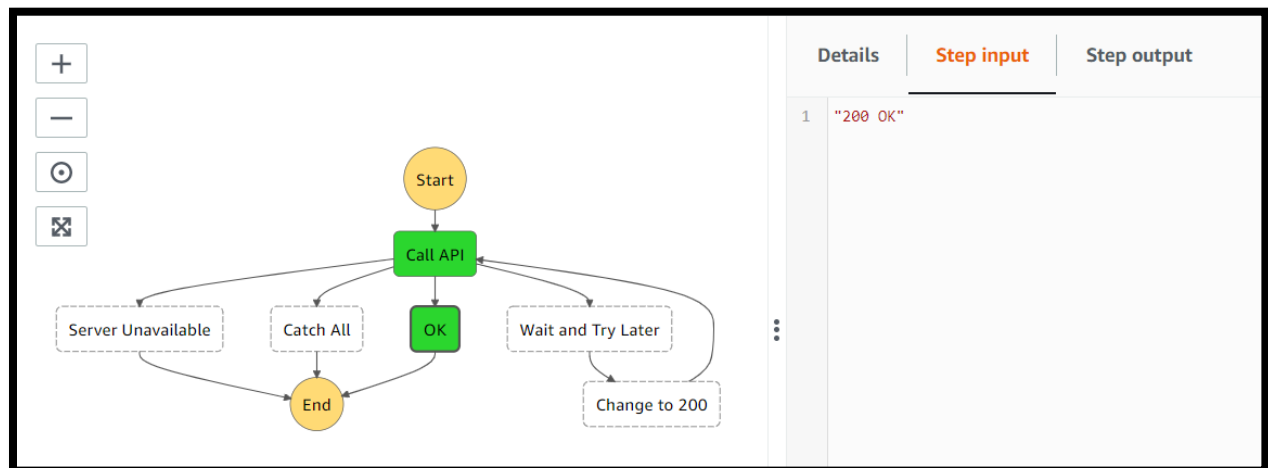


# Cloud Plus Plus Services



Details	Step input	Step output
1	"200 OK"	

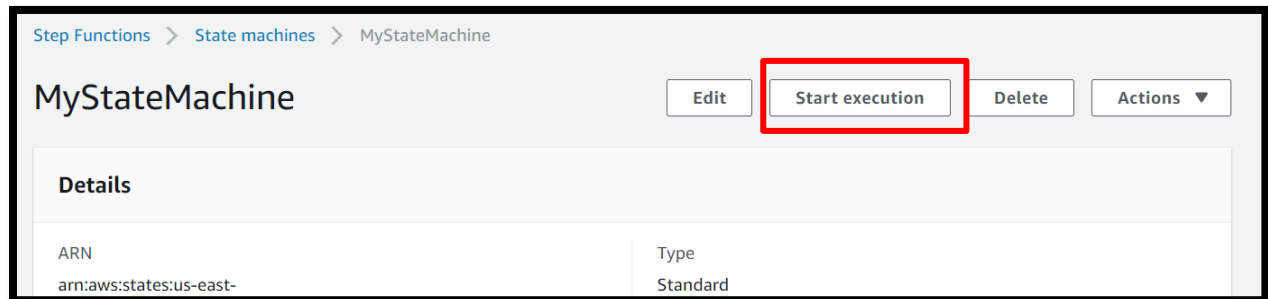
Next, click on the "OK" Task state in the visual workflow.



## Step 5. Inspect the Execution of your State Machine

Scroll to the top of the Execution details screen and click on **MyAPIStateMachine**.

Click on Start execution again, and this time provide the following input and then click Start execution.



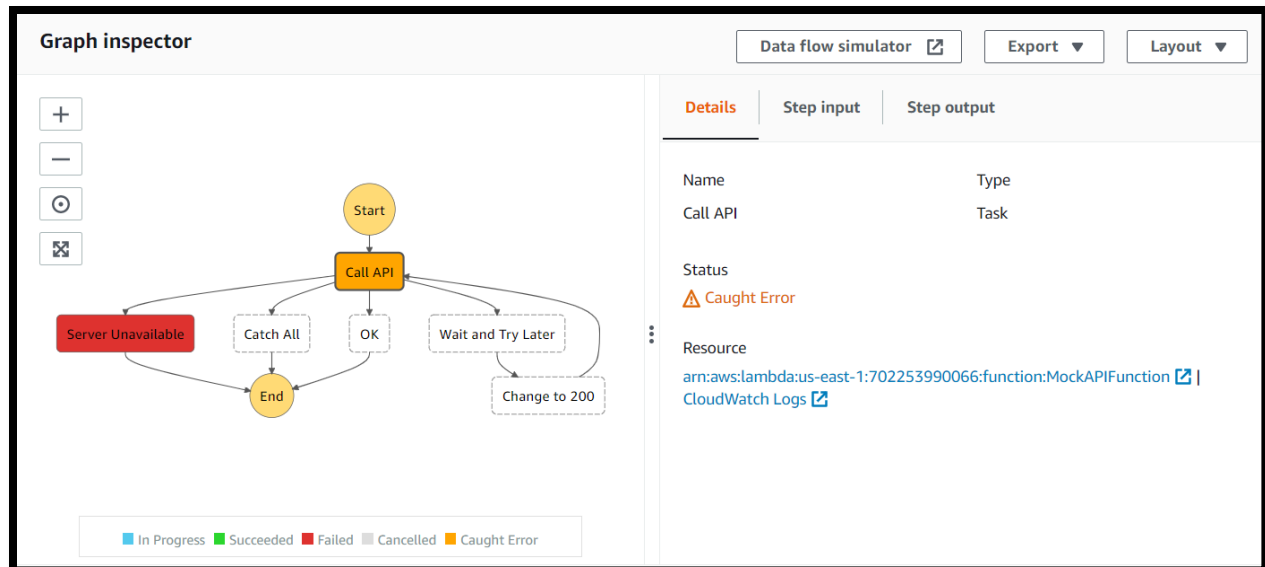
Click on Start execution again, and this time provide the following input and then click Start execution.

```
{  
  "statusCode": "503"  
}
```

In the Execution event history section, expand each execution step to confirm that your workflow behaved as expected.



# Cloud Plus Plus Services



**Execution event history**

ID	Type	Step	Resource	Elapsed Time (ms)	Timestamp
▶ 1	ExecutionStarted		-	0	Nov 24, 2021 09:38:31.616 PM
▶ 2	TaskStateEntered	Call API	-	32	Nov 24, 2021 09:38:31.648 PM
▶ 3	LambdaFunctionScheduled	Call API	<a href="#">Lambda</a>   <a href="#">CloudWatch Logs</a>	32	Nov 24, 2021 09:38:31.648 PM
▶ 4	LambdaFunctionStarted	Call API	<a href="#">Lambda</a>   <a href="#">CloudWatch Logs</a>	67	Nov 24, 2021 09:38:31.683 PM
▼ 5	LambdaFunctionFailed	Call API	<a href="#">Lambda</a>   <a href="#">CloudWatch Logs</a>	130	Nov 24, 2021 09:38:31.746 PM
<pre>1 { 2   "error": "ServerUnavailableException", 3   "cause": { 4     "errorMessage": "503 Server Unavailable", 5     "errorType": "ServerUnavailableException", 6     "requestId": "dc8b9732-d83c-42b5-9a12-58ad552052bd", 7     "stackTrace": [ 8       "  File \"/var/task/lambda_function.py\", line 10, in lambda_handler\n    raise ServerUnavailableException('503 Server Unavailable')\n" 9     ] 10  } 11 }</pre>					
▶ 6	TaskStateExited	Call API	-	130	Nov 24, 2021 09:38:31.746 PM
▶ 7	FailStateEntered	Server Unavailable	-	138	Nov 24, 2021 09:38:31.754 PM
▶ 8	ExecutionFailed		-	138	Nov 24, 2021 09:38:31.754 PM

Next, simulate a 429 exception. Scroll to the top of the Execution details screen and click on **MyAPIStateMachine**. Click on Start execution, provide the following input, and click Start execution.

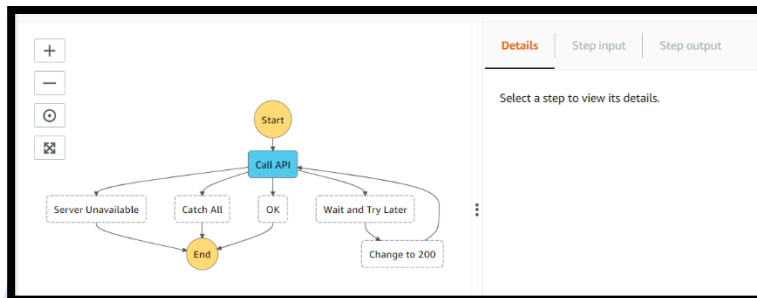
```
{
  "statusCode": "429"
}
```

# Cloud Plus Plus Services



**Details** | Execution input | Execution output | Definition

Execution Status  
 Running



Next, the Wait state used brute force to change the response code to 200, and your workflow completed execution successfully.

**Details** | Execution input | **Execution output** | Definition

Execution Status  
 Succeeded

**Graph inspector** Data flow simulator Export Layout

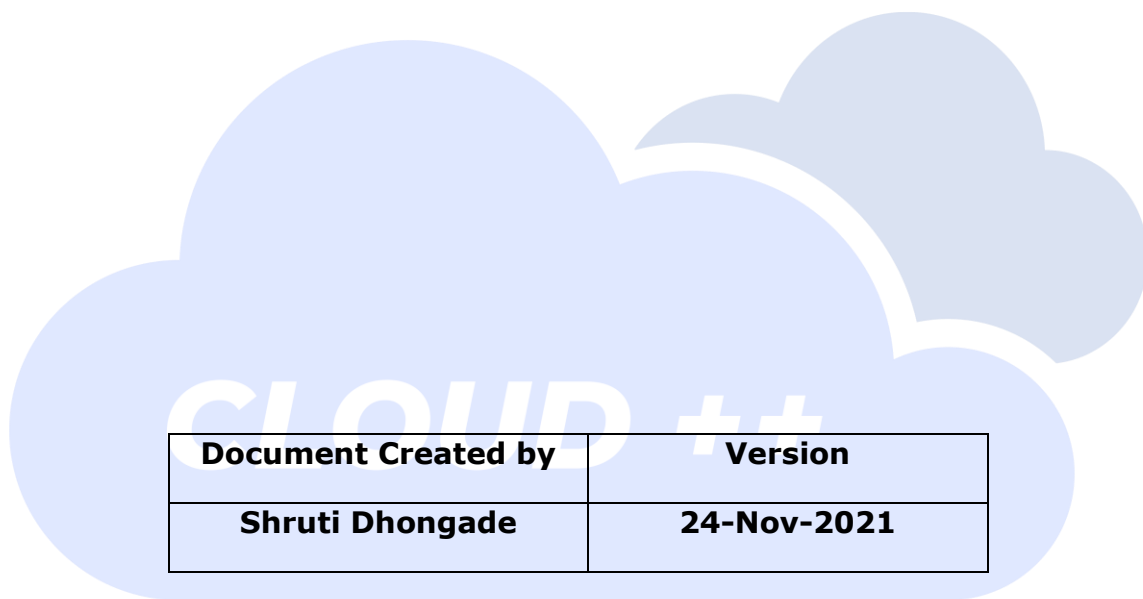
```
graph TD; Start((Start)) --> CallAPI[Call API]; CallAPI --> ServerUnavailable[Server Unavailable]; CallAPI --> CatchAll[Catch All]; CallAPI --> OK[OK]; CallAPI --> Wait[Wait and Try Later]; ServerUnavailable --> End((End)); CatchAll --> End; OK --> End; Wait --> Change200[Change to 200]; Change200 --> End;
```

**Details** | Step input | Step output  
  
Name: Call API      Type: Task  
  
Status: Succeeded  
  
Resource: [arn:aws:lambda:us-east-1:702253990066:function:MockAPIFunction](#) | [CloudWatch Logs](#)

# Cloud Plus Plus Services



**Note:** If you no longer need the resources, you may delete the Lambda Function, the state machine, IAM role



Document Created by	Version
Shruti Dhongade	24-Nov-2021

Your trusted partner for  
cloud enablement