Build AWS Step Functions Integration with AWS SAM and OpenAPI

Step Functions integrates directly with API Gateway, enabling you to trigger the execution of a state machine with an HTTP request. API Gateway can be used with both Standard and Express workflows. Integrations may be **synchronous** or **asynchronous**. We will create an Express Workflow to be called **synchronous** and a Standard Workflow to be called **asynchronous**.

> **Note**
> A Step Functions **Express** workflow can be called Sync and Async. A **Standard** workflow can only be called Async.

Use AWS SAM and OpenAPI to create an API Gateway REST API with Express Step Functions synchronous and asynchronous integration

1. Using AWS Cloud9 console, return to the folder `module-3/step-functions`
2. This code belongs in your SAM template file `template.yaml`

Review the code and then **copy/paste** it into the `template.yaml` file.

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  API Gateway example for AWS Step Functions Integration

Globals:

    # Enable Logs
    Api:
        MethodSettings:
            ResourcePath: "/*"
            HttpMethod: "*"
            DataTraceEnabled: True
            LoggingLevel: INFO
            MetricsEnabled: True


Resources:

  # Step Functions Express Definition
  StateMachineExpress:
    Type: AWS::Serverless::StateMachine
    Properties:
      Name: StateMachineExpress
      DefinitionUri: state_machine/step_function_asl.json
      Policies:
        - Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - "cloudwatch:*"
                - "logs:*"
              Resource: "*"
      Type: EXPRESS
      Logging:
        Destinations:
          - CloudWatchLogsLogGroup:
              LogGroupArn: !GetAtt StateMachineLogGroup.Arn
        IncludeExecutionData: false
        Level: 'ALL'

  # Log group for Express Step Functions
  StateMachineLogGroup:
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Join [ "/", [ '/stepfunctions', StateMachineExpress]]

  # Step Functions Standard Definition
  StateMachineStandard:
    Type: AWS::Serverless::StateMachine
    Properties:
      Name: StateMachineStandard
      DefinitionUri: state_machine/step_function_asl.json
      Policies:
        - Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - "cloudwatch:*"
                - "logs:*"
              Resource: "*"
      Type: STANDARD

  # APIGW Rest API for Step Functions Integration Example
  RestApiForSyncNF:
    Type: AWS::Serverless::Api
    Properties:
      StageName: dev
      DefinitionBody:  # an OpenApi definition
        'Fn::Transform':
          Name: 'AWS::Include'
          Parameters:
            Location: 'openapi.yaml'
      OpenApiVersion: 3.0.3
      EndpointConfiguration:
        Type: REGIONAL
      Variables:
        SFANEXPRESS: !GetAtt StateMachineExpress.Arn
        SFANSTANDARD: !GetAtt StateMachineStandard.Arn

  # IAM Role to allow APIGW to call SF Express and Standard
  RestApiRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          Effect: Allow
          Principal:
            Service:
              apigateway.amazonaws.com
          Action:
            'sts:AssumeRole'
      Policies:
        PolicyName: AllowSFNExec
        PolicyDocument:
          Version: 2012-10-17
          Statement:
            Effect: Allow
            Action:
              "states:StartSyncExecution"
              "states:StartExecution"
            Resource:
              !GetAtt StateMachineExpress.Arn
              !GetAtt StateMachineStandard.Arn

Outputs:
  ExpressSFIntegrationRestAPI:
    Description: "Express Workflow API endpoint"
    Value: !Sub "https://${RestApiForSyncNF}.execute-api.${AWS::Region}.amazonaws.com/dev"

ExpressStepFunctionsArn:
  Description: "Step Functions Express ARN"
  Value: !GetAtt StateMachineExpress.Arn

StandardStepFunctionsArn:
  Description: "Step Functions Standard ARN"
  Value: !GetAtt StateMachineStandard.Arn
```

The `StateMachineExpress` resource defines a **Step Functions Express workflow**. The `StateMachineStandard` defines a **Step Functions Standard workflow.**

The Type value **(lines 34 and 62)** specifies the type of the Step Functions.

The `RestApiForSyncNF` resource defines a Rest API with two Stage Variables. The variables will be used by the Integration Request transformer to get the Step Functions ARN. In the testing section you will see this topic in more detail.

The `StateMachineLogGroup` resource defines a log group to `StateMachineExpress` log all actions.

3. This code belongs in your OpenAPI definition file `openapi.yaml`

Review the code and then **copy/paste** it into the `openapi.yaml` file

```
openapi: "3.0.1"
info:
  title: "module-3-stepfunctions-integration"
  description: "API Gateway example for AWS Step Functions Integration"
  version: "1.0"

paths:
  /async:
    post:
      produces:
      - "application/json"
      responses:
        "200":
          description: "200 response"
          schema:
            $ref: "#/definitions/Empty"
      x-amazon-apigateway-integration:
        credentials:
          Fn::Sub: "${RestApiRole.Arn}"
        httpMethod: "POST"
        uri: "arn:aws:apigateway:${AWS::Region}:states:action/StartExecution"
        responses:
          default:
            statusCode: "200"
        requestTemplates:
          application/json: "{\n    \"input\": \"$util.escapeJavaScript($input.json('$'))\",\
            \n    \"stateMachineArn\": \"$util.escapeJavaScript(${stageVariables.SFANSTANDARD})\"\
            \n}"
        passthroughBehavior: "when_no_match"
        type: "aws"
  /sync:
    post:
      produces:
      - "application/json"
      responses:
        "200":
          description: "200 response"
          schema:
            $ref: "#/definitions/Empty"
      x-amazon-apigateway-integration:
        credentials:
          Fn::Sub: "${RestApiRole.Arn}"
        httpMethod: "POST"
        uri: "arn:aws:apigateway:${AWS::Region}:states:action/StartSyncExecution"
        responses:
          default:
            statusCode: "200"
        requestTemplates:
          application/json: "{\n    \"input\": \"$util.escapeJavaScript($input.json('$'))\",\
            \n    \"stateMachineArn\": \"$util.escapeJavaScript(${stageVariables.SFANEXPRESS})\"\
            \n}"
        passthroughBehavior: "when_no_match"
        type: "aws"
definitions:
  Empty:
    type: "object"
    title: "Empty Schema"
```

The x-amazon-apigateway-integration object **(lines 17 and 40)** specifies details of the backend integration used for this method.

> **Important**
> For Step Functions, you must use the HTTP method of **POST** for the integration request, according to the specification of the Step Functions service action for workflow invocations. In **x-amazon-apigateway-integration** all methods were defined with `httpMethod: "POST"`.

On **lines 30** and **53** we have `type: "aws"`, responsible for configuring the integration as **Step Functions** type (service integration).

On **line 21** the **uri** is pointing to `StartExecution`. This is the **async** Step Functions API.

On **line 44** the **uri** is pointing to `StartSyncExecution`. This is the **sync** Step Functions API.

With custom Step Functions it is possible to have more control in the request and response workflow. In booth examples we are transforming the input payload before passing it to Step Functions using Apache Velocity Template Language (VTL).

In **Line 26 an 49** we have the request transformation template definition.

> **Note**
> This example does not include the ARN of your state machine in the body of your API Gateway call, we configured a body-mapping template, as shown in the following example.
>
> With this approach, you can specify ARNs of different state machines based on your development stage (for example, dev, test, and prod).

```
{
  "input": "$util.escapeJavaScript($input.json('$'))",
  "stateMachineArn": "$util.escapeJavaScript($stageVariables.arn)"
}
```

We will see more details of this transformation in the test section.

Deploy the project

1. To deploy the Amazon API Gateway and the AWS Step Functions to your AWS account, run the following commands from the application root `module-3/step-functions`, where the `template.yaml` file for the sample application is located:

```
sam build && sam deploy --guided
```

The first time that you run the `sam deploy --guided` command, AWS SAM starts an AWS CloudFormation deployment. In this case, you needed to say what are the configurations that you want SAM to have in order to get the guided deployment. You can configure as it is above.

- **Stack Name**: `module-3-step-functions-integration`
- **AWS Region** Put the chosen region to run the workshop, e.g. `us-east-1`
- **Confirm changes before deploy**: Y
- **Allow SAM CLI IAM role creation**: Y
- **Disable rollback**: N
- **Save arguments to configuration file**: Y
- **SAM configuration file** and **SAM configuration environment** leave blank

```
Setering default arguments for 'sam deploy'
=========================================
Stack Name [module-3-step-function-integration]: module-3-step-function-integration
AWS Region [us-west-2]:
#Shows you resources changes to be deployed and require a 'Y' to initiate deploy
Confirm changes before deploy [Y/n]: Y
#SAM needs permission to be able to create roles to connect to the resources in your template
Allow SAM CLI IAM role creation [Y/n]: Y
#Preserves the state of previously provisioned resources when an operation fails
Disable rollback [y/N]: N
Save arguments to configuration file [Y/n]: Y
SAM configuration file [samconfig.toml]:
SAM configuration environment [default]:
```

2. After configuring the deployment, AWS SAM will display assets that will be created. But first, it will automatically upload the template to a temporary bucket it creates. Then, it will ask you to confirm the changes. Type `y` to confirm.

```
CloudFormation stack changeset
-------------------------------------------------------------------------------------------------
Operation                     LogicalResourceId             ResourceType                  Replacement
-------------------------------------------------------------------------------------------------
+ Add                         RestApiRole                   AWS::IAM::Role                N/A
+ Add                         RestApiforExpressAPIDeploymen AWS::ApiGateway::Deployment   N/A
                              tabdc4cd619
+ Add                         RestApiforExpressAPIDevStage  AWS::ApiGateway::Stage        N/A
+ Add                         RestApiforExpressAPI          AWS::ApiGateway::RestApi      N/A
+ Add                         StateMachineExpressRole       AWS::IAM::Role                N/A
+ Add                         StateMachineExpress           AWS::StepFunctions::StateMac  N/A
                                                            hine
+ Add                         StateMachineLogGroup          AWS::Logs::LogGroup           N/A
-------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-west-2:[redacted]:changeSet/samcli-deploy1679965253/2771a07d-4c3b-4c3a-8b47-d38659e18541


Previewing CloudFormation changeset before deployment
=====================================================
Deploy this changeset? [y/N]: y
```

3. After it was finished, a new stack `module-3-step-functions-integration` will be successfully created. Take note of the following outputs: StandardStepFunctionsArn, ExpressStepFunctionsArn and ExpressSFIntegrationRestAPI as they will be used later in the tests section.

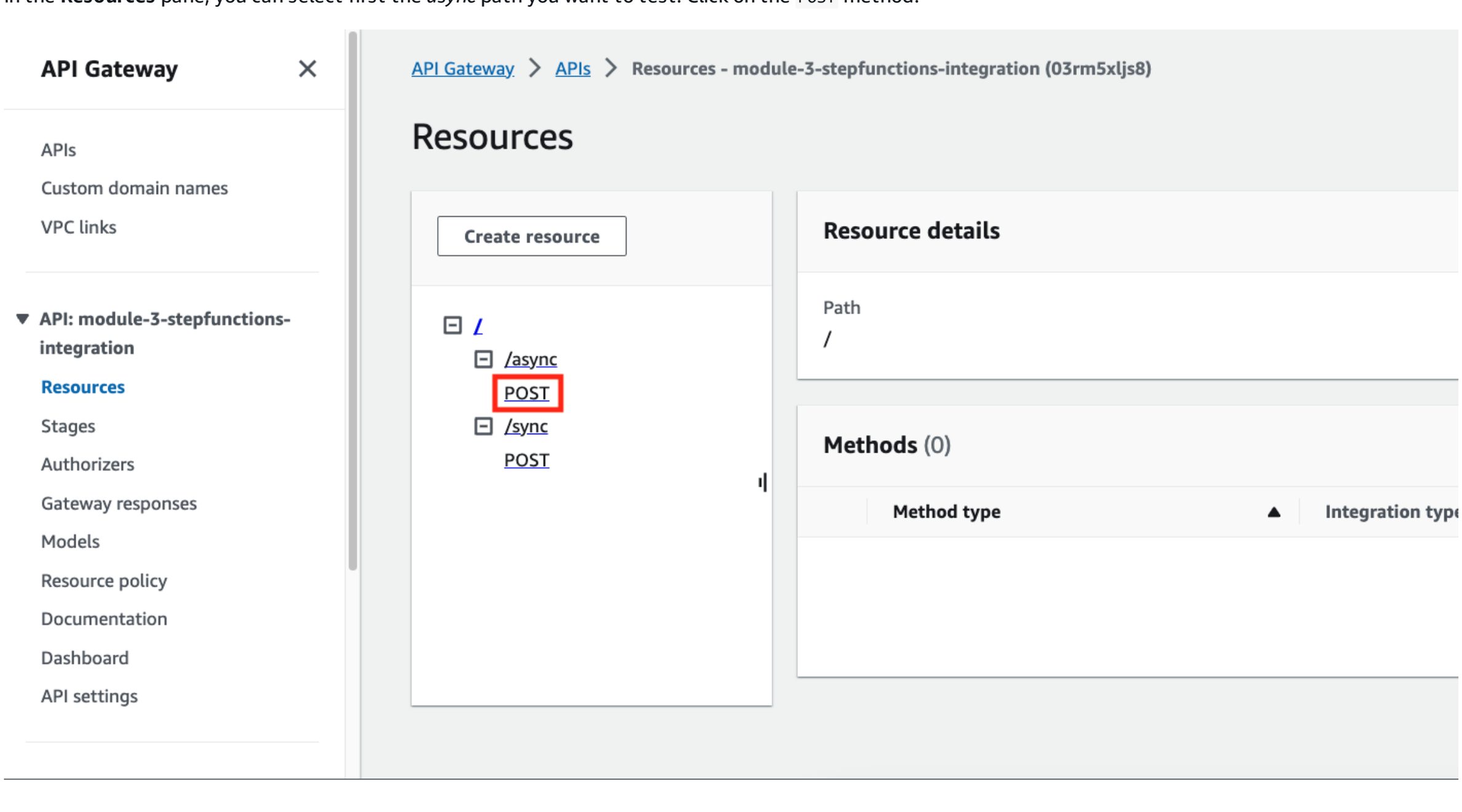
Test Integration

After you create your API Gateway REST API with AWS Step Functions Integration, you can test the API.

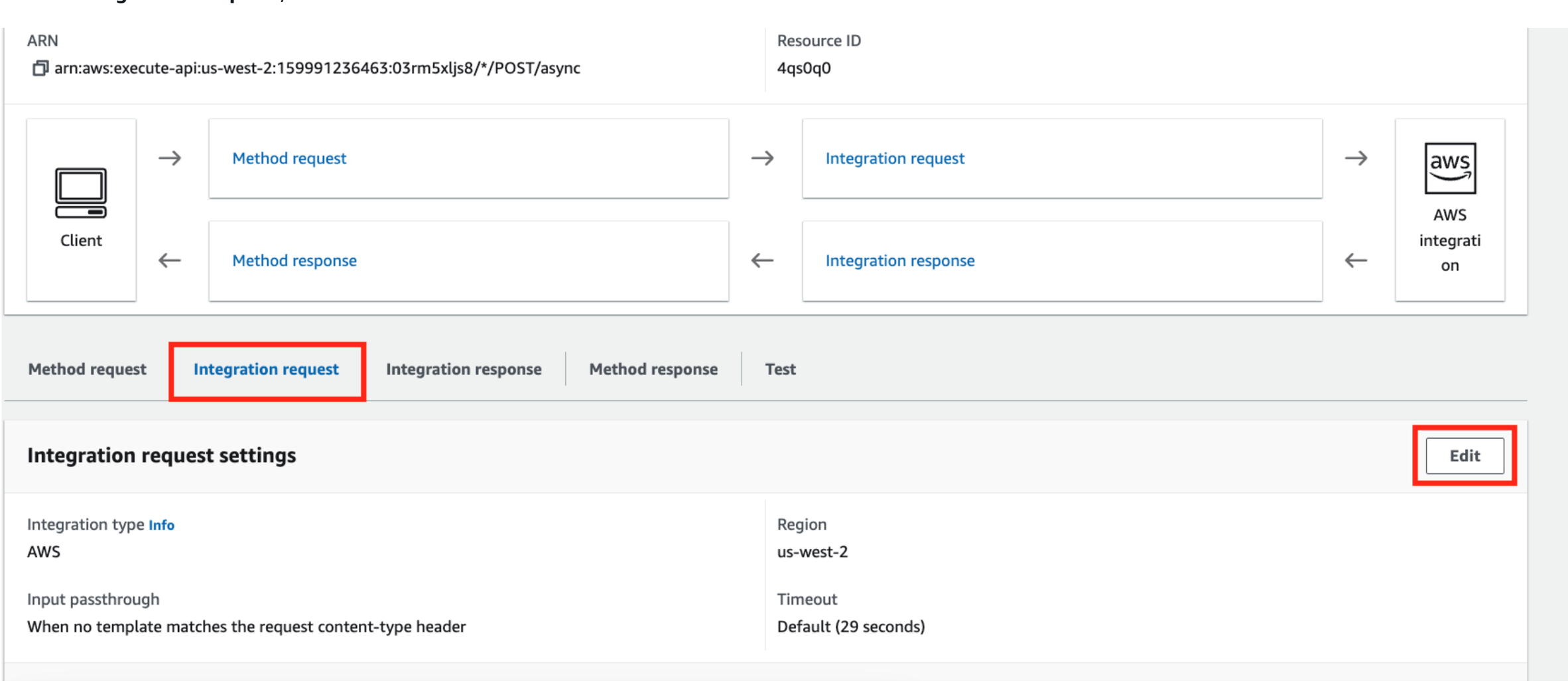
Test the Asynchronous AWS Step Functions Integration using API Gateway console

First let's test the **async** type integration.

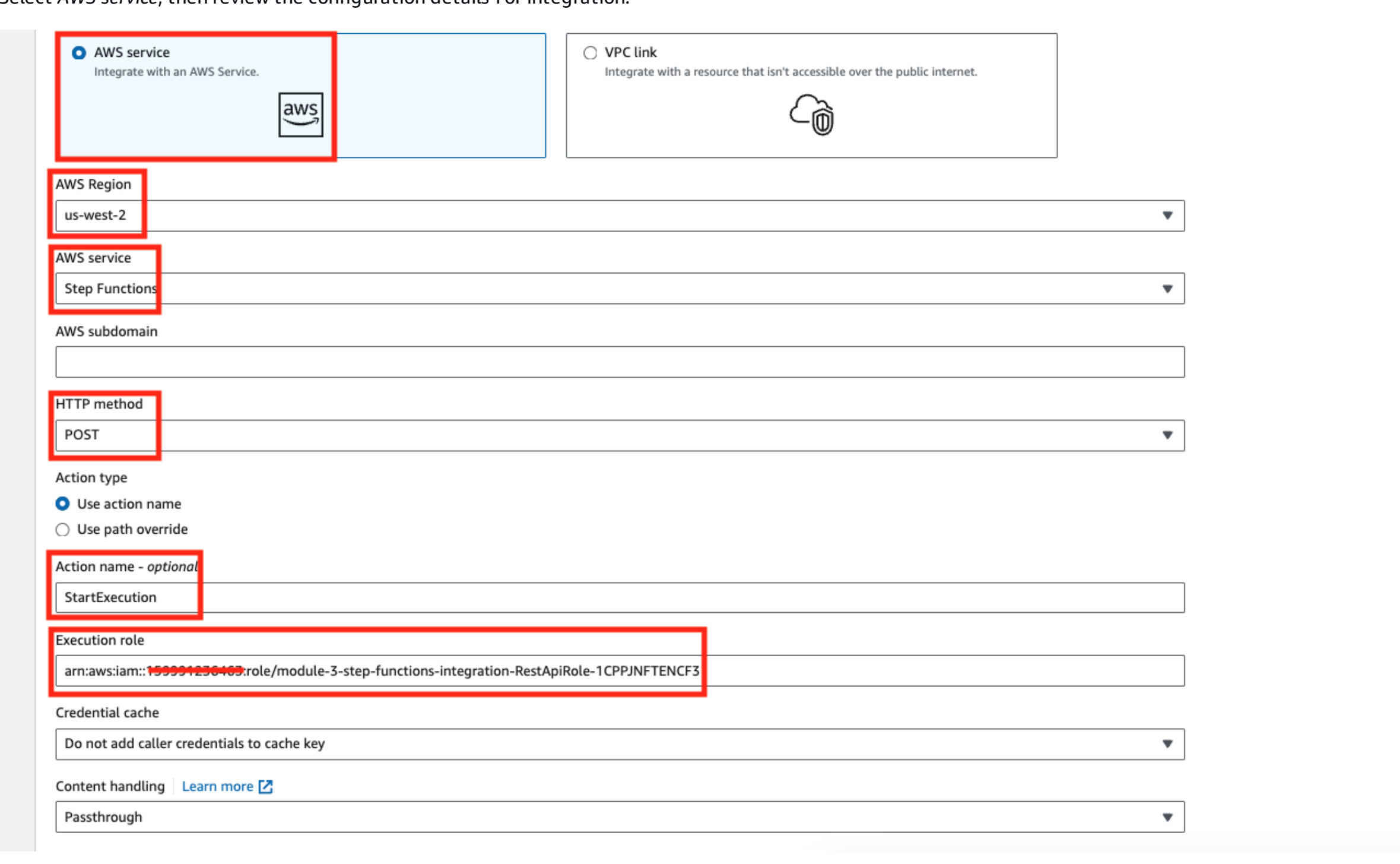
1. Open the Amazon API Gateway console and sign in.
2. Choose your REST API named, `module-3-step-functions-integration`.
3. In the **Resources** pane, you can select first the *async* path you want to test. Click on the `POST` method.



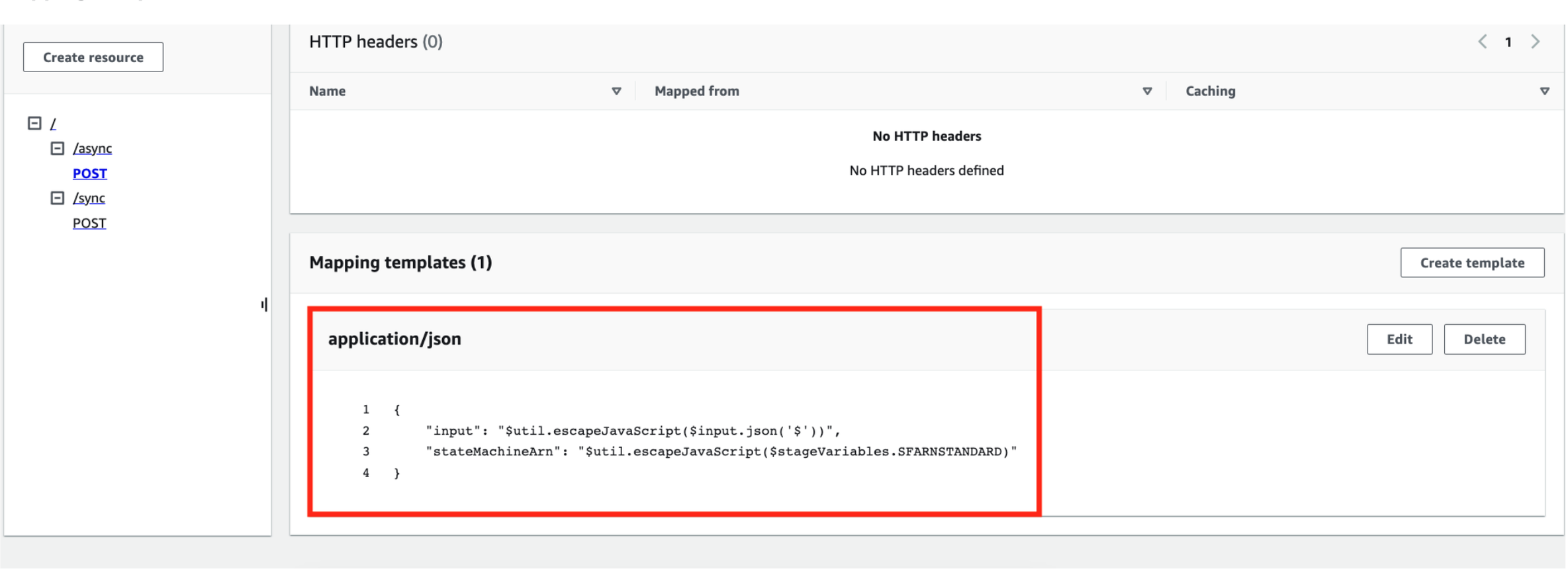
4. Click in **Integration Request**, then in *Edit*.



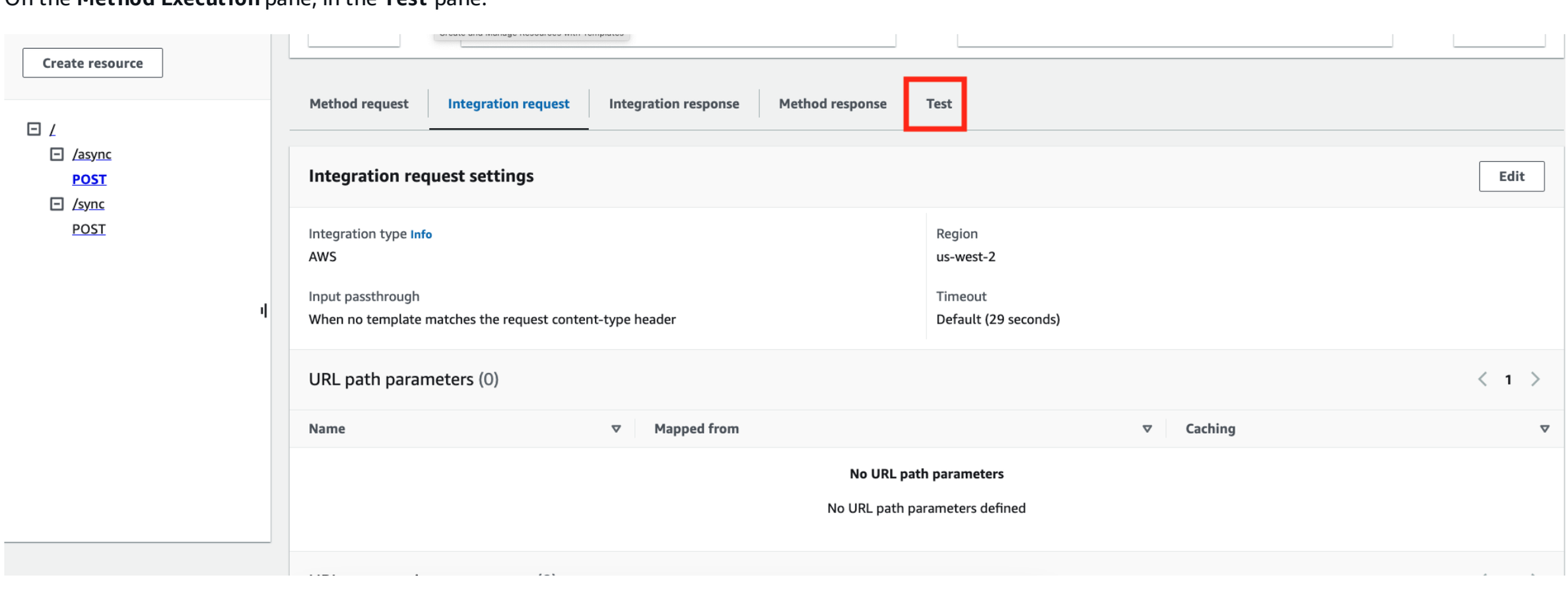
5. Select *AWS service*, then review the configuration details for integration.



6. Click on Save at the end of the page. Back on the **Integration Request** table, at the end of the page, see the input transformation request template from the **Mapping Template**:



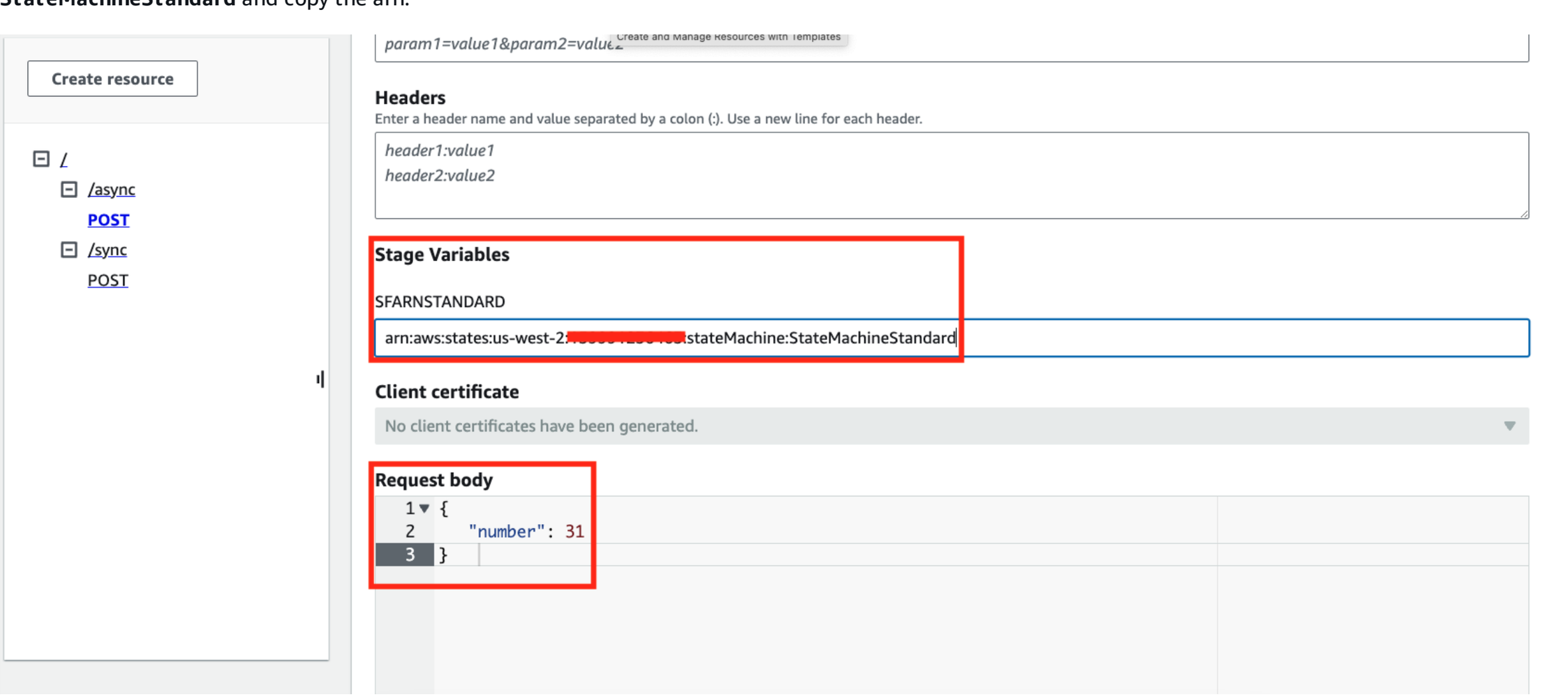
7. On the **Method Execution** pane, in the **Test** pane.



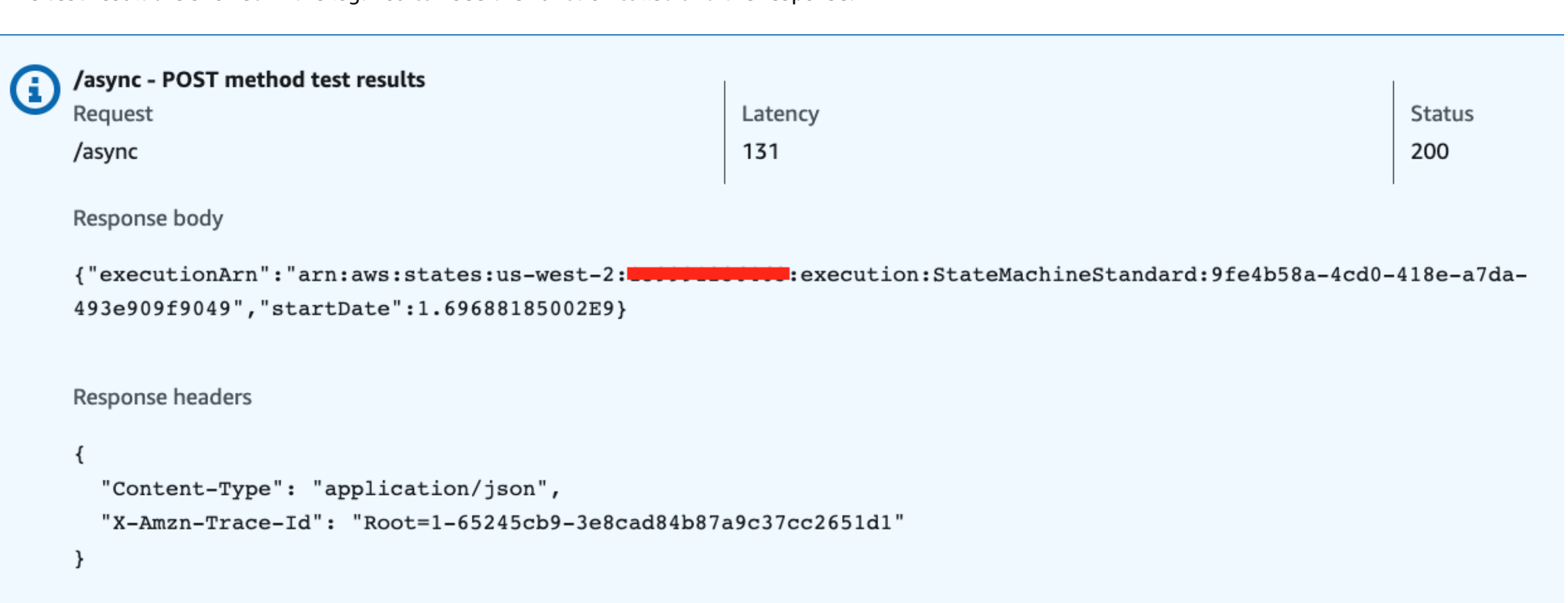
8. For Request Body, **copy/paste** the event test below.

```
{
  "number": 31
}
```

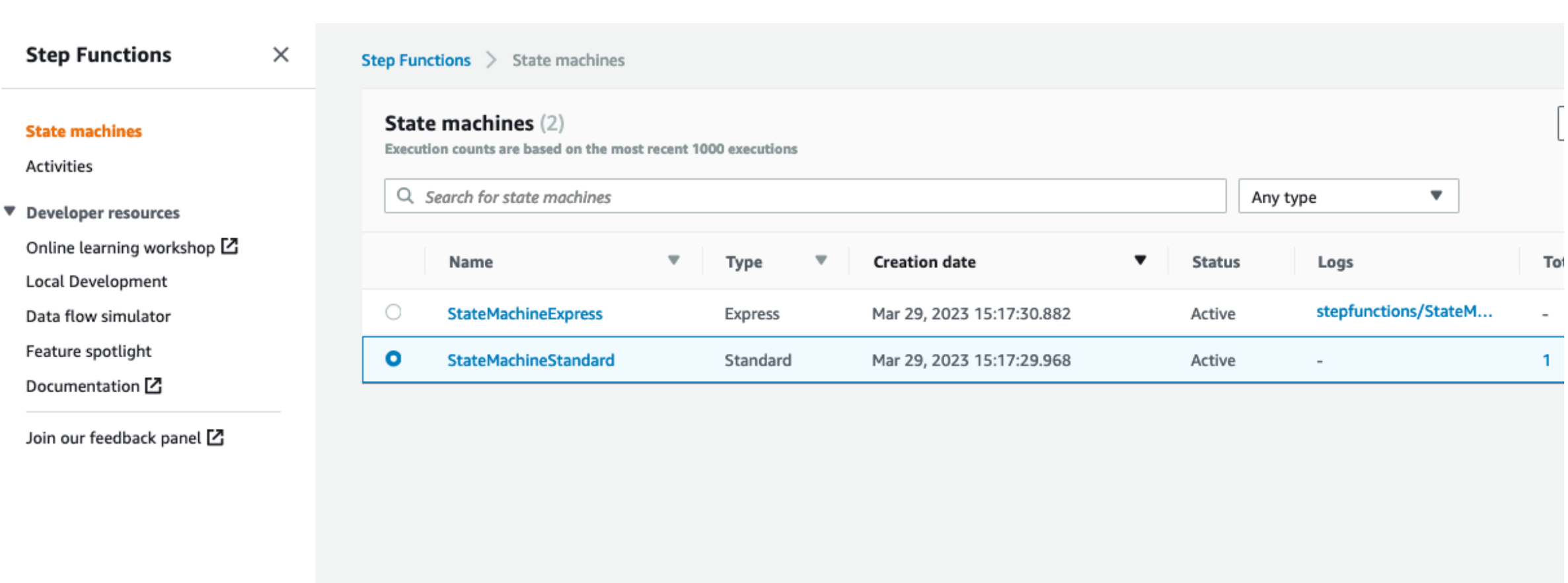
9. For Stage Variable **SFARNSTANDARD**, **copy/paste** the output ARN **StandarStepFunctionsArn** from Cloud9 Console, or go to Step Functions console, click on **StateMachineStandard** and copy the arn.



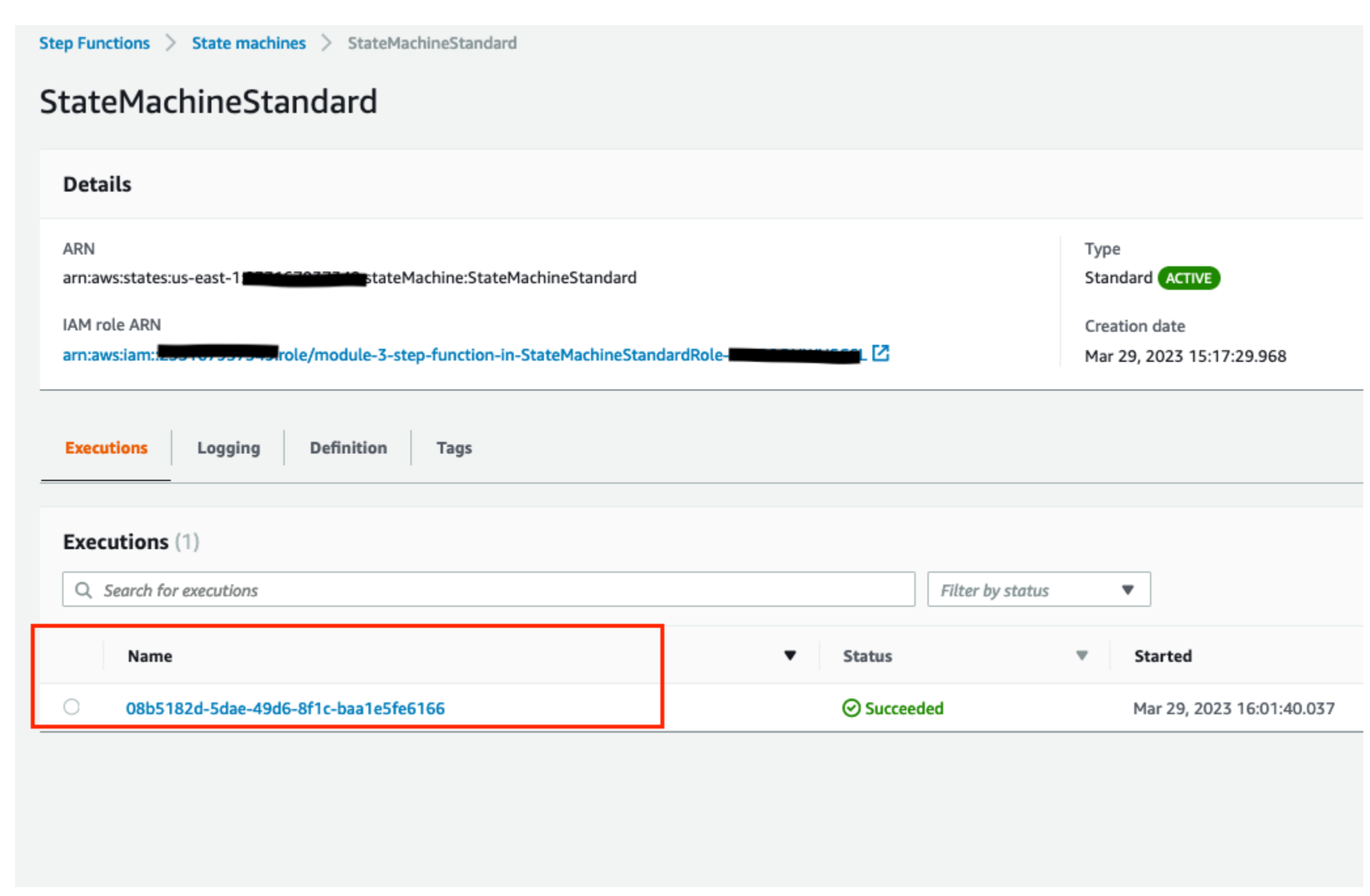
10. The test result are showed in the log. You can see the function called and the response:



11. Go to Step Functions Console, click on State machines

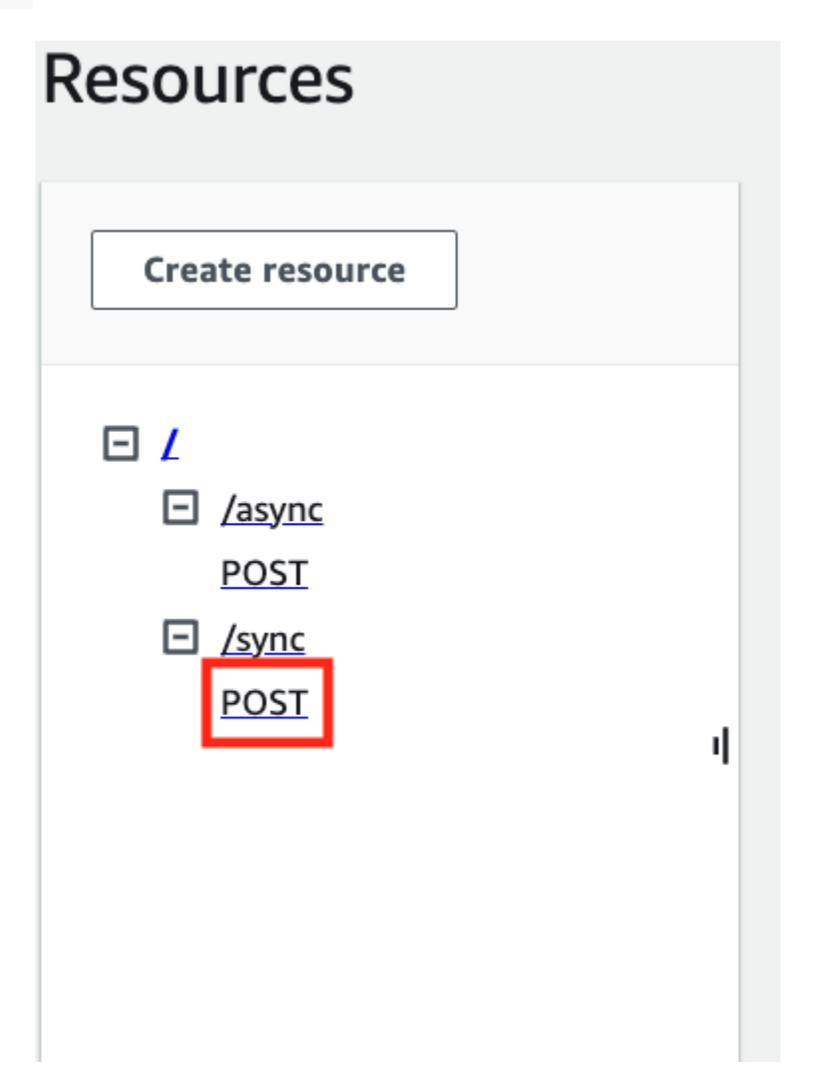


12. Click on **StateMachineStandard**, and then click on the execution name to see details about the workflow execution

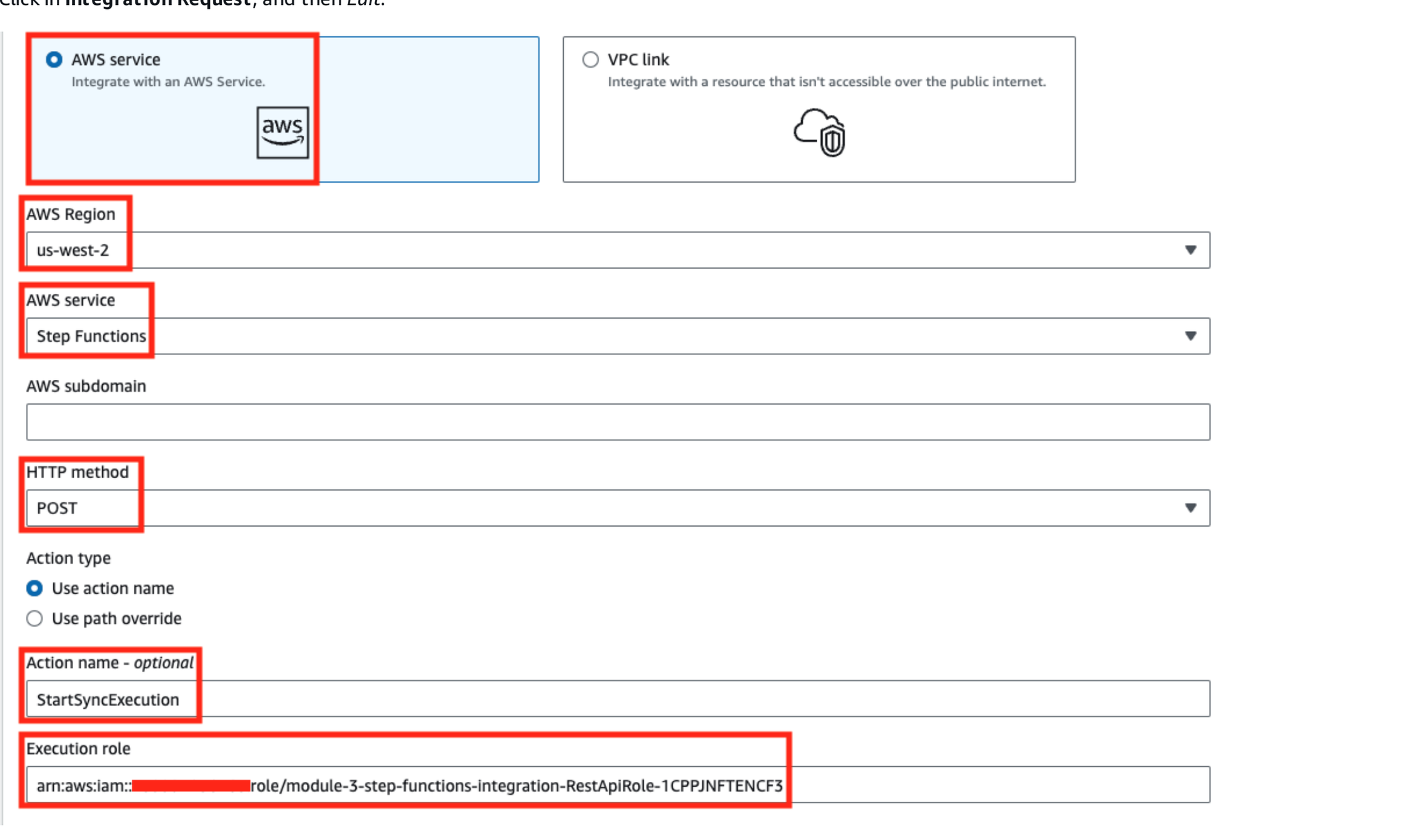


Test the Synchronous AWS Step Functions Integration using API Gateway console

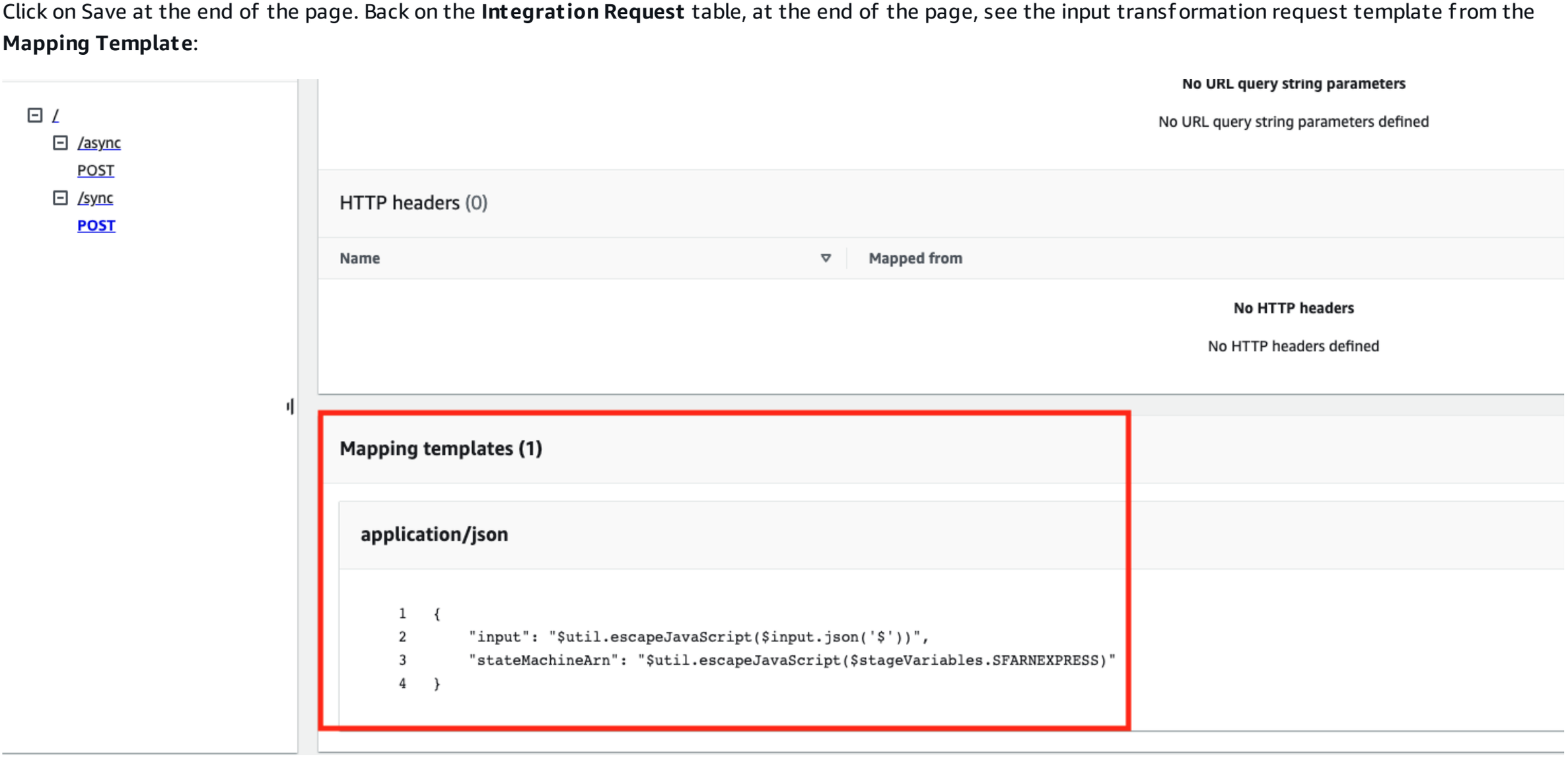
1. Back on the **API Gateway**. Choose your REST API named, `module-3-step-functions-integration`.
2. Select the *sync* path you want to test. Click on the `POST` method.



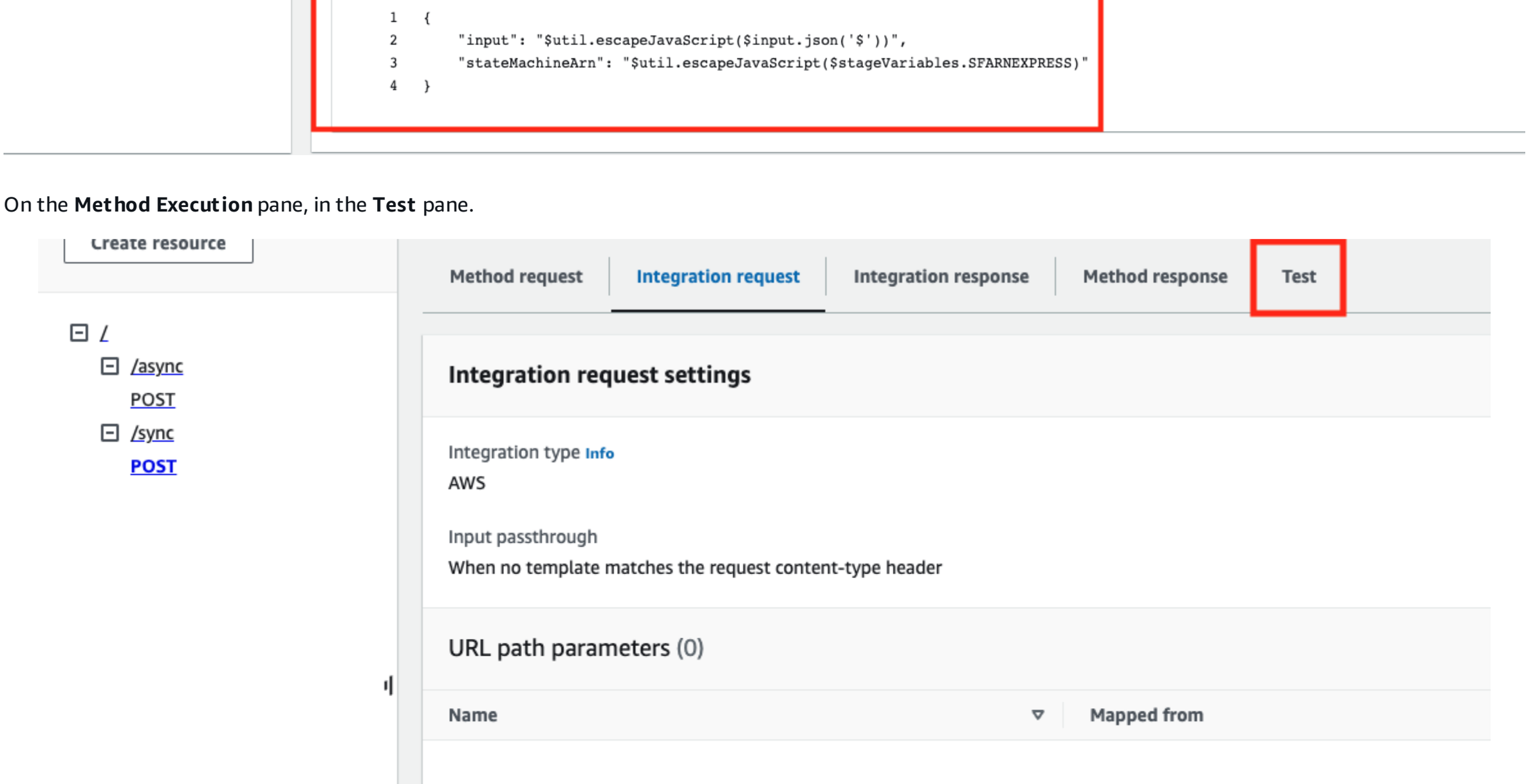
3. Click in **Integration Request**, and then *Edit*.



4. Click on Save at the end of the page. Back on the **Integration Request** table, at the end of the page, see the input transformation request template from the **Mapping Template**:



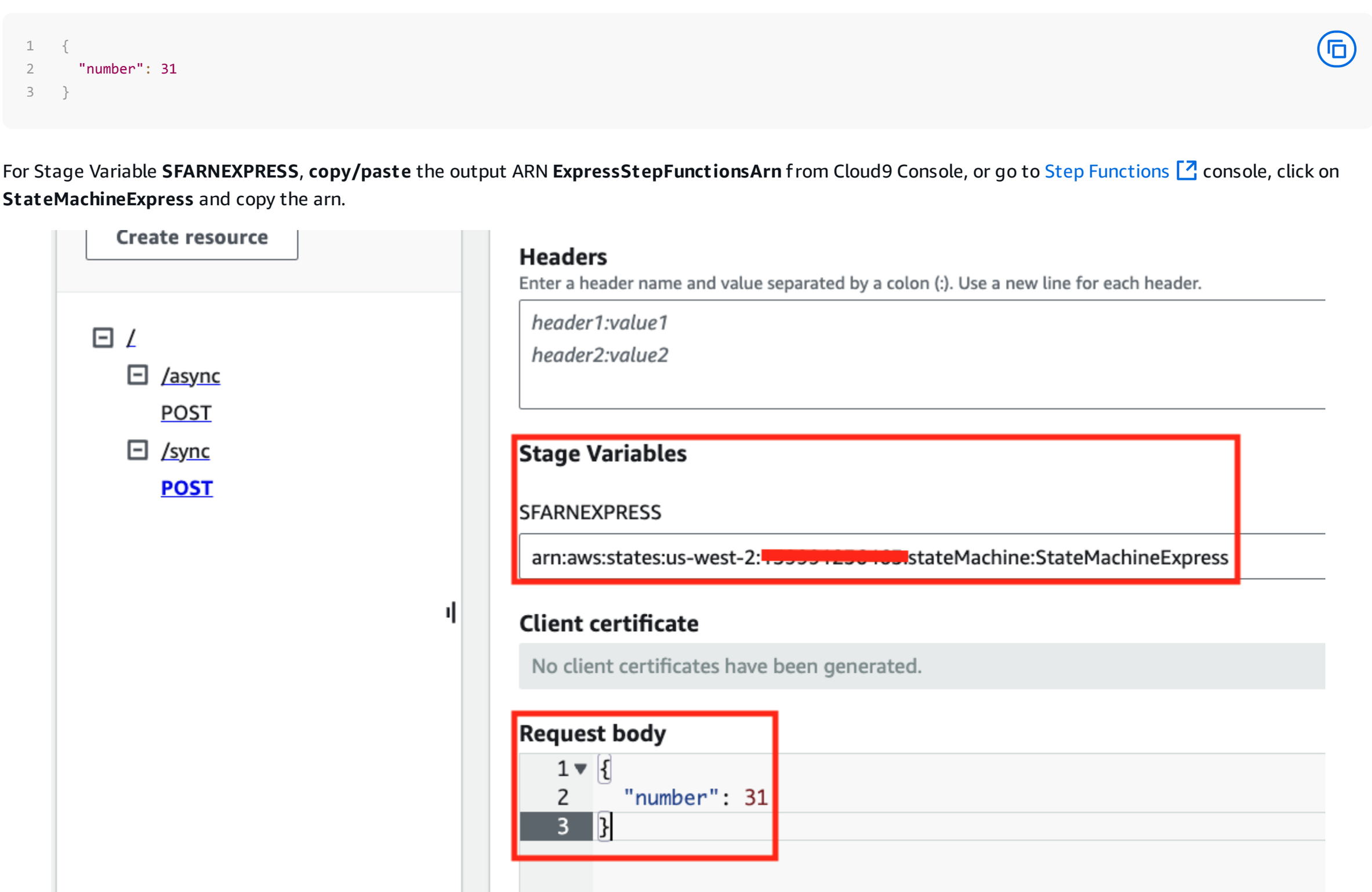
5. On the **Method Execution** pane, in the **Test** pane.



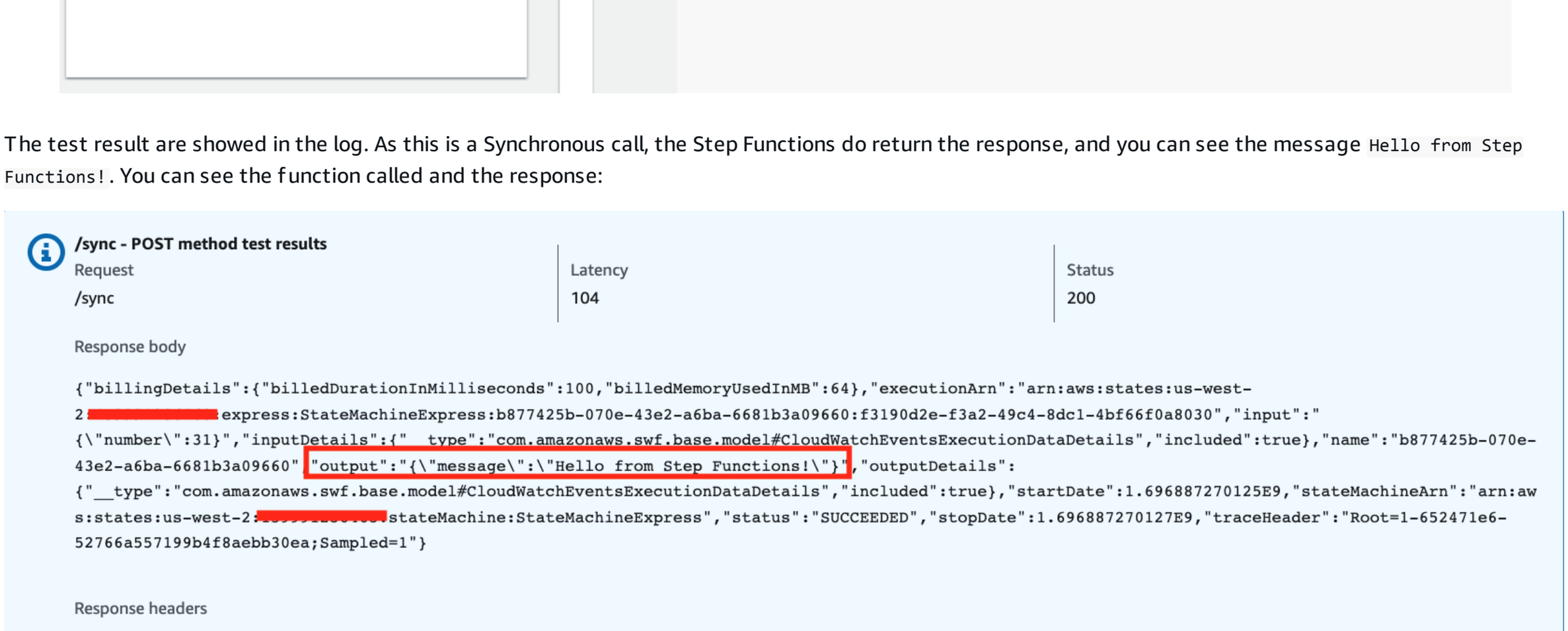
6. For Request Body, **copy/paste** the event test below.

```
{
  "number": 31
}
```

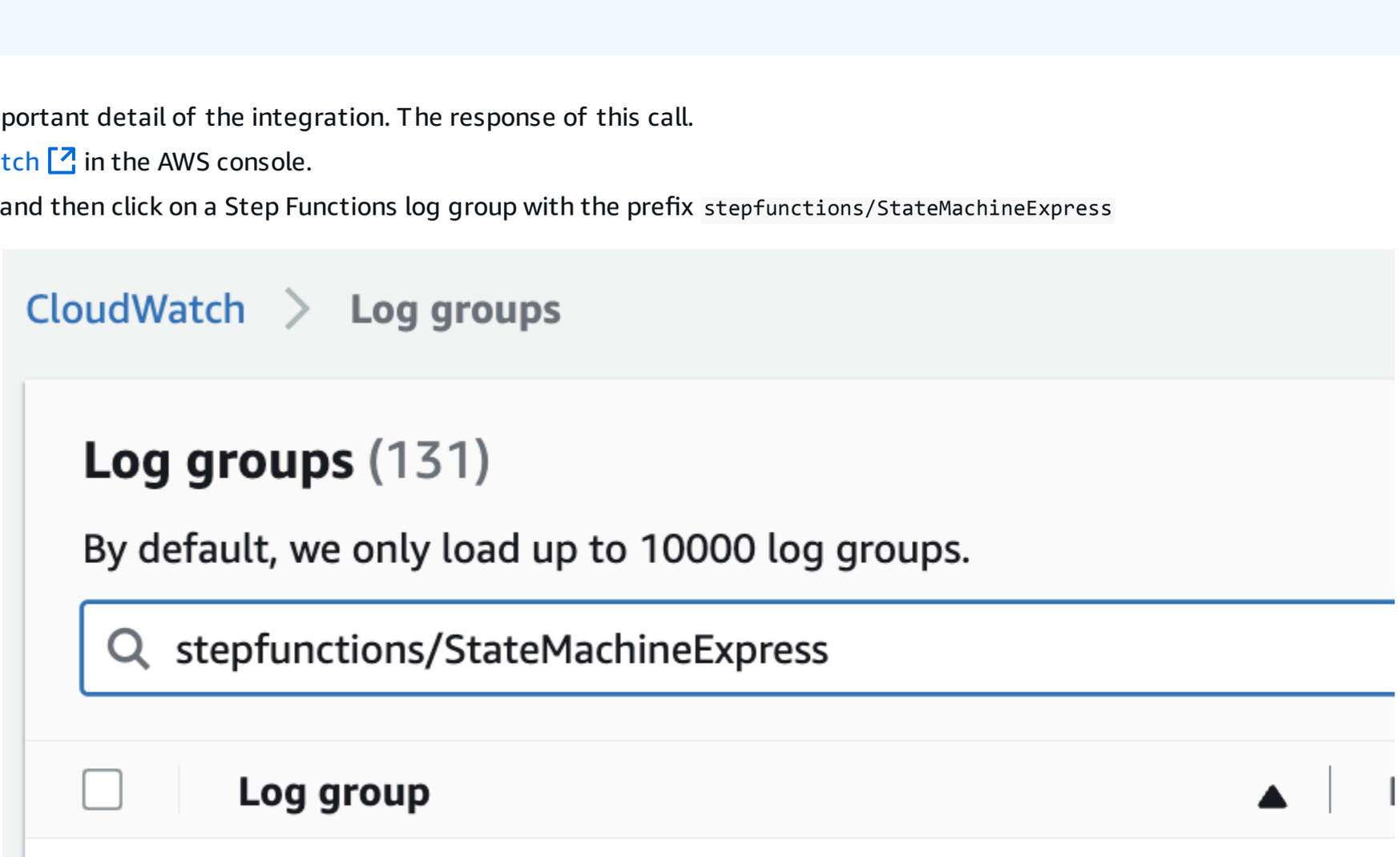
7. For Stage Variable **SFARNEXPRESS**, **copy/paste** the output ARN **ExpressStepFunctionsArn** from Cloud9 Console, or go to Step Functions console, click on **StateMachineExpress** and copy the arn.



8. The test result are showed in the log. As this is a Synchronous call, the Step Functions do return the response, and you can see the message `Hello from Step Functions!`. You can see the function called and the response:

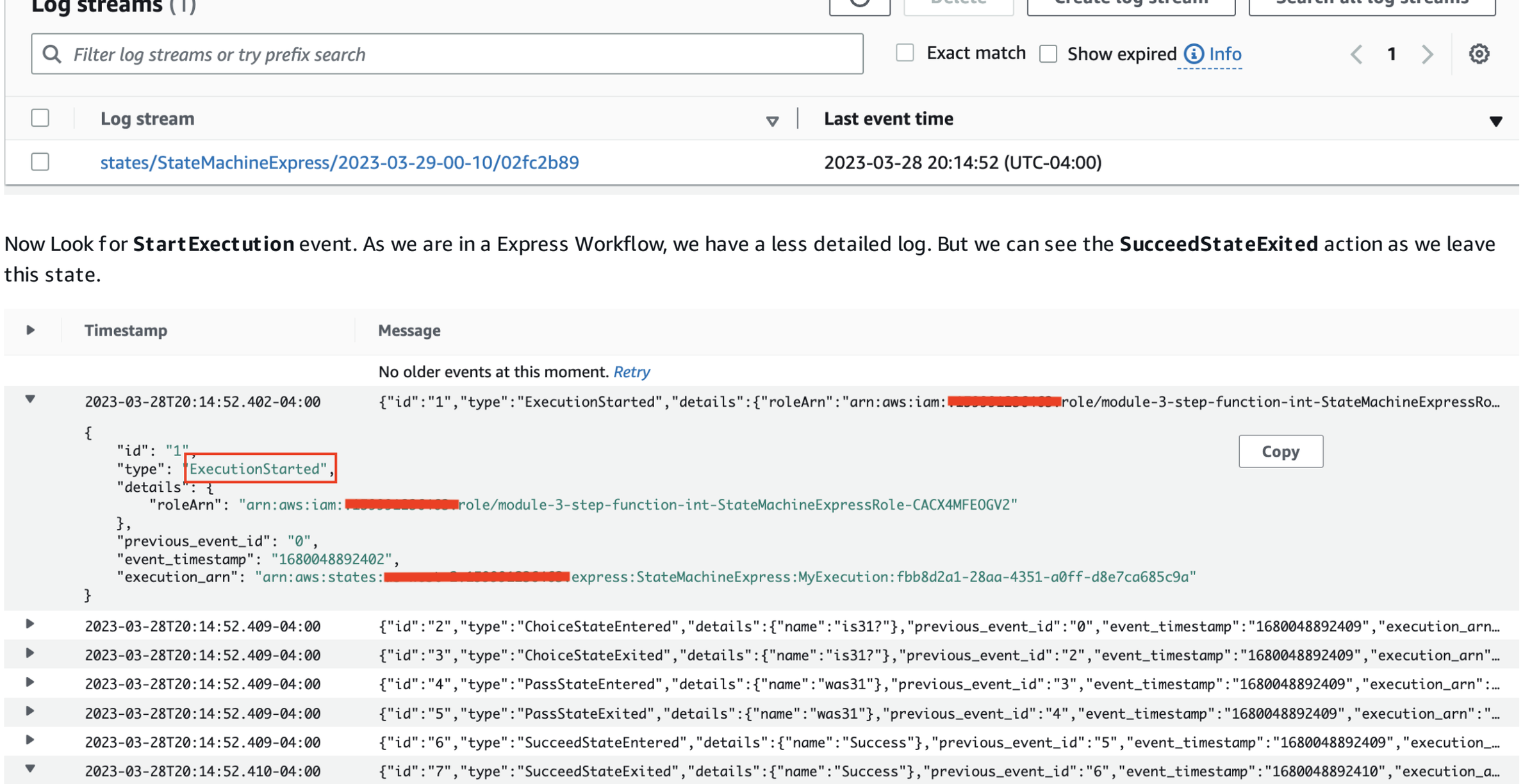


9. Now let's check an important detail of the integration. The response of this call.
10. Navigate to CloudWatch in the AWS console.
11. Click on **Log Groups**, and then click on a Step Functions log group with the prefix `stepfunctions/StateMachineExpress`



12. Click on the most recent logstream

13. Now Look for **StartExecution** event. As we are in a Express Workflow, we have a less detailed log. But we can see the **SucceedStateExited** action as we leave this state.



Test the deployed API using cURL

1. Open a new terminal window in your AWS Cloud9 environment.
2. Copy the following cURL command and paste it into the terminal window, replacing `<api-id>` with your API's API ID and `<region>` with the region where your API is deployed. You may have copied this URL in from the CloudFormation output in the last step. You can also find the full invoke URL in the API Gateway console by navigating to **Stages > dev**. This will call an Step Functions Asynchronous.

```
curl -X POST https://<api-id>.execute-api.<region>.amazonaws.com/dev/async \
-H 'Content-Type: application/json' \
-d '{"number" : 31}'
```

3. The output should be:

```
{"executionArn":"arn:aws:states:us-east-1:XYZ:execution:StateMachineStandard:XYZ,"startDate":1.680118019525E9}%
```

4. Do the same to test the Step Functions Synchronous call.

```
curl -X POST https://<api-id>.execute-api.<region>.amazonaws.com/dev/sync \
-H 'Content-Type: application/json' \
-d '{"number" : 31}'
```

5. The output should be:

```
{
 "billingDetails":{
   "billedDurationInMilliseconds":100,
   "billedMemoryUsedInMB":64
 },
   "executionArn":"arn:aws:states:us-east-1:233167937349:express:StateMachineExpress:cfd77ee3-6fdb-43d4-a7fb-8e96b0a32c15:4eea1880-b652-421f-a8c3-7b5a6ff251f0",
   "input":"{\"number\":31}",
   "inputDetails":{
       "__type":"com.amazonaws.swf.base.model#CloudWatchEventsExecutionDataDetails",
       "included":true
   },
   "name":"cfd77ee3-6fdb-43d4-a7fb-8e96b0a32c15",
   "output":"{\"message\":\"Hello from Step Functions!\"}",
   "outputDetails":{
     "__type":"com.amazonaws.swf.base.model#CloudWatchEventsExecutionDataDetails",
     "included":true
   },
   "startDate":1.680118226325E9,
   "stateMachineArn":"arn:aws:states:us-east-1:233167937349:stateMachine:StateMachineExpress",
   "status":"SUCCEEDED",
   "stopDate":1.680118226335E9,
   "traceHeader":"Root=1-642491d2-4380e8cb5764286d73bf0198;Sampled=1"
}
```

Tip
When testing a method using the API Gateway console, logs are not sent to CloudWatch Logs. When calling externally, for example via cURL, the logs are sent normally.

Note
Note that via cURL it was not necessary to pass the stage variable, as it is set in the dev stage, that is, when calling through the external endpoint, the variable is filled in by API Gateway.

Challenge
Find the payload logs before and after the **request** and **response** transformations in CloudWatch.