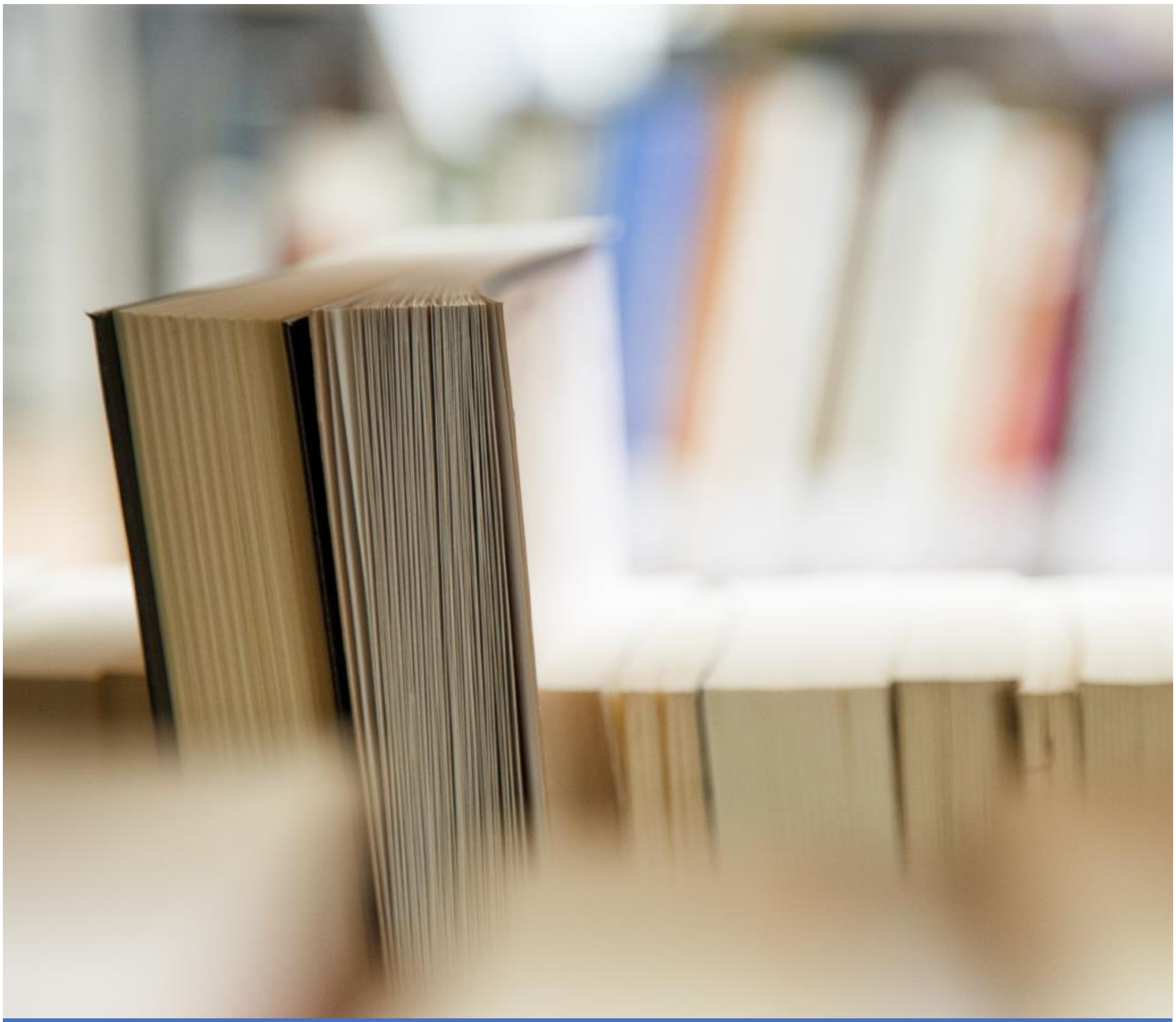


Demo: Creating a Lambda Function

Saurav Raghuvanshi

12/3/21

Serverless Computing

Author: Saurav Raghuvanshi

Aim:

- **Understand what Lambda functions are and what makes them unique**
- **Create Lambda functions using the AWS Console**
- **Test Lambda functions**

Description:

AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you, making it easy to build applications that respond quickly to new information. Lambda opens all kinds of new possibilities and can lower your costs at the same time. When running a job-processing server in EC2, you are charged for compute time as long as your instance is still running. Contrast that with Lambda where you are only charged while processing a job. This makes Lambda a great fit for spiky or infrequent workloads because it scales automatically and minimizes costs during slow periods. The event-based model Lambda provides makes it perfect for providing a backend for mobile clients, other smart devices, or adding no-stress asynchronous processing to an existing application.

In this introductory Lab, you will learn how to use AWS Lambda to easily run code to react to events. Events can come from DynamoDB changes, SNS messages, S3 objects, Kinesis streams, or a variety of other sources. Owing to its versatility, Lambda has found use in a wide variety of applications including mobile apps, Internet of Things backends, and big data systems.

Introduction:

AWS Lambda Logo AWS Lambda is a service released for Developer Preview in April 2015. Some examples of the kinds of things you can achieve with Lambda are designing advanced materialized views out of DynamoDB tables, reacting to uploaded files on S3, and processing SNS messages or Kinesis streams.

In short, you can write a stateless Lambda function that will be triggered to react to certain events (or HTTP endpoints). In a way, it's the same Platform as a Service (PaaS) vein as Heroku. The key difference is in how data (events for Lambda, web requests for Heroku) is delivered to your code and the level of granularity you can achieve. In both cases, you write code that accepts some constraints in exchange for not having to do any provisioning, updating, or management of the underlying resources.

Lambda opens up all kinds of new possibilities and can lower your costs at the same time. When running a job processing server in EC2, you are charged for compute-time as long as your instance is running. Contrast that with Lambda, where you are only charged while actually processing a job, on a 100ms basis. Basically, you never pay for idle time.

This makes Lambda a great fit for spiky or infrequent workloads because it scales automatically and minimizes costs during slow periods. The event-based model Lambda provides makes it perfect for providing a backend for mobile clients, IoT devices, or adding no-stress asynchronous processing to an existing application, without worrying too much about scaling your compute power.

The current computing landscape for AWS looks crowded at first glance with EC2, Elastic Beanstalk, EKS, Lambda, Simple Workflow Service, and more vying for your workload. Before you start on this Lab, you should understand where Lambda fits in.

EC2 is the most basic service, as it only provides the instance with a base image while you supply the automation, configuration, and code to run. It's the most flexible option, but it also requires the most work from you.

Lambda is the complete opposite in that it handles provisioning, underlying OS updates, monitoring, and failover transparently. You only need to provide the code that will run and specify what events should trigger your code. Scaling a Lambda function happens automatically; AWS provisions more instances as needed and only charges you for the time your function runs.

Elastic Beanstalk is a PaaS that lets you deploy code without worrying about the underlying infrastructure. However, compared to Lambda, it does provide more choices and controls. You can deploy complete applications to Elastic Beanstalk using a more traditional application model compared to deploying individual functions in Lambda.

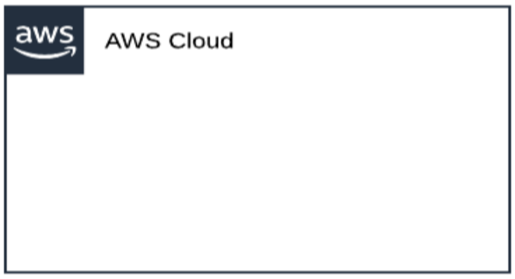
Amazon Elastic Container Service (Amazon ECS) and Amazon Elastic Container Service for Kubernetes (Amazon EKS) are centered around containers compared to the individual functions of Lambda. ECS and EKS require less managerial overhead compared to running containers on EC2 instances, but generally require some operational expertise. Lambda is ideal for developers who just want to focus on their code.

Simple Workflow Service is a coordination service, and you must provision workers to complete your tasks.

In this Lab, you will set up a Lambda function, learn how to test code in the AWS Console, and discuss different event sources for bringing data into Lambda. Functions like the ones you will write in this Lab can be used to help keep data in sync, fan out writes to users' news feeds, or update indexes in DynamoDB and other databases.

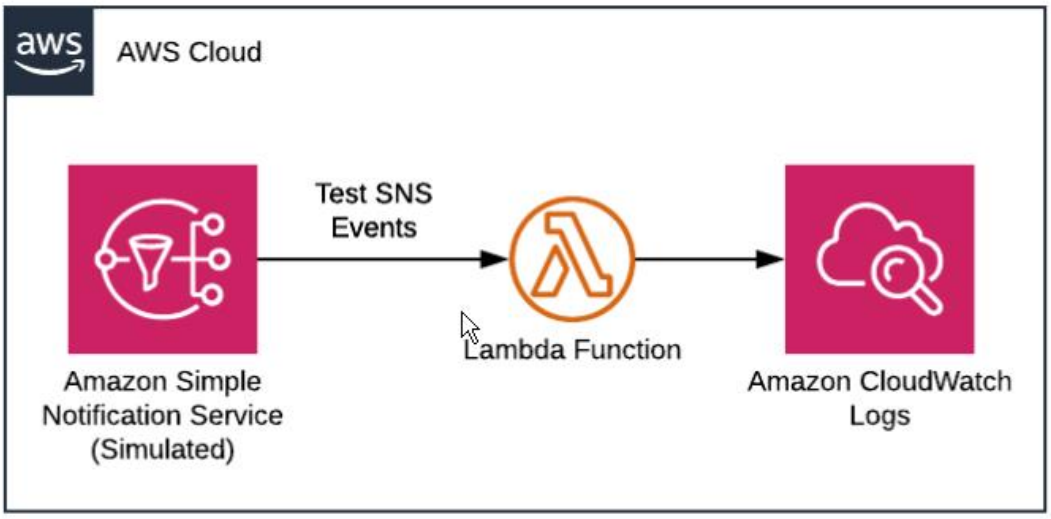
Architecture we are going to design:

Initial:



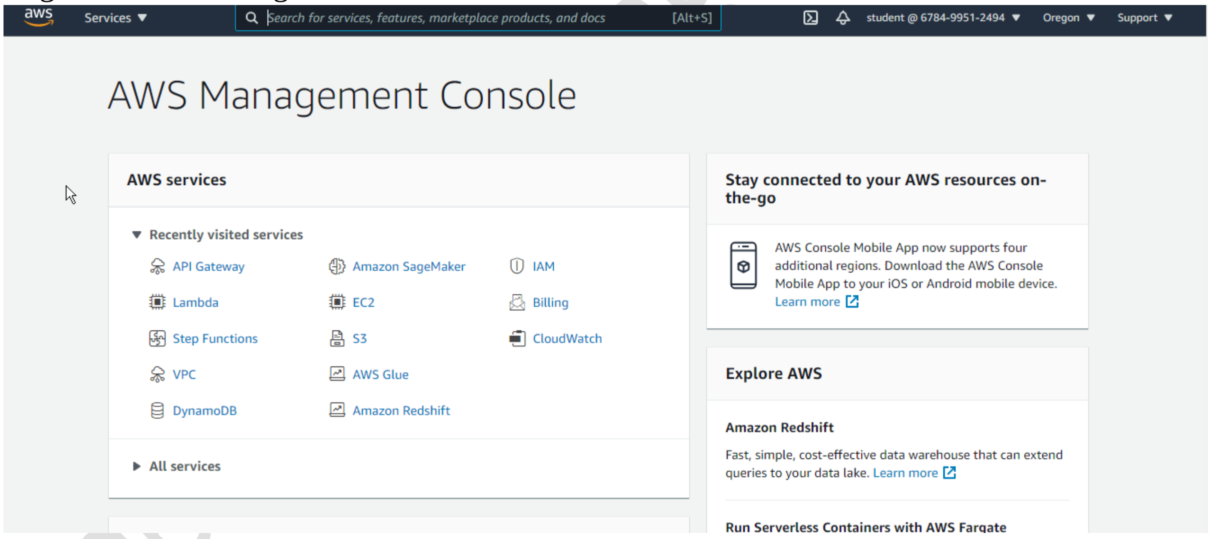
Author: Saurav Raghuvanshi

Final:



Instructions:

1. Login to AWS Management Console.



Stay connected to your AWS resources on-the-go

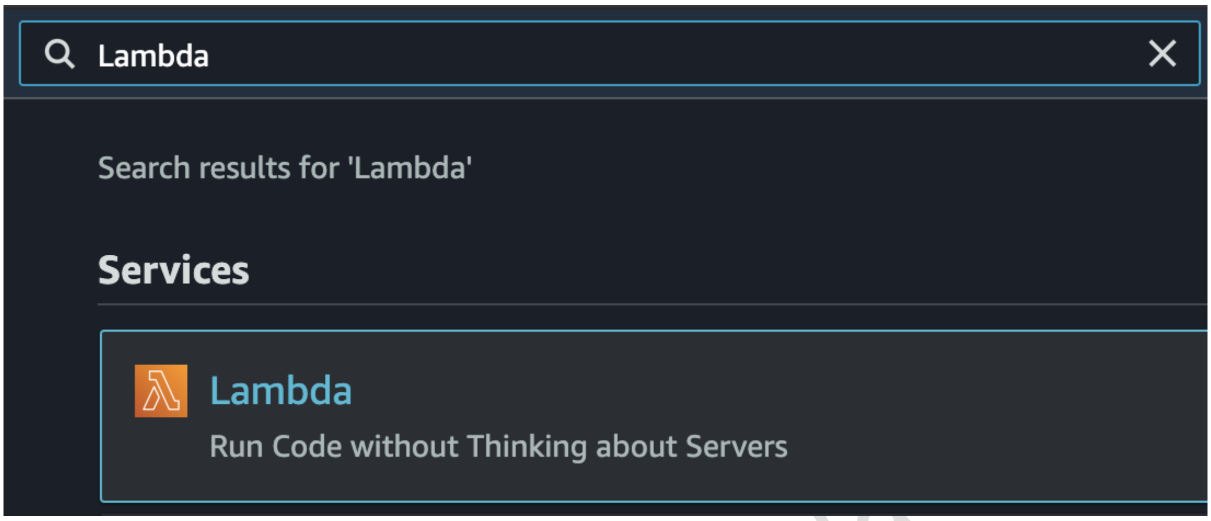
AWS Console Mobile App now supports four additional regions. Download the AWS Console Mobile App to your iOS or Android mobile device. Learn more

Amazon Redshift

Fast, simple, cost-effective data warehouse that can extend queries to your data lake. Learn more

Run Serverless Containers with AWS Fargate

2. In the AWS Management Console search bar, enter Lambda, and click the Lambda result under Services:



Lambda

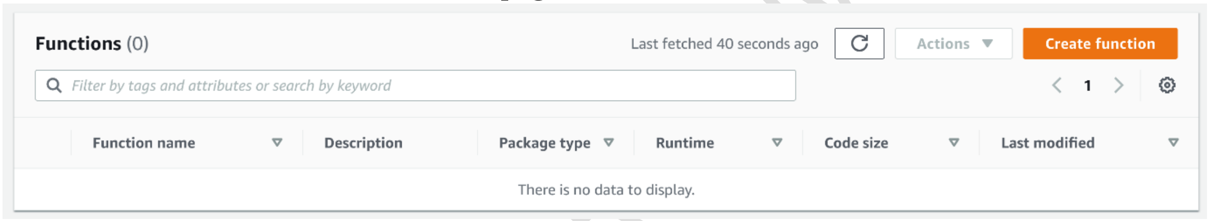
Search results for 'Lambda'

Services

Lambda

Run Code without Thinking about Servers

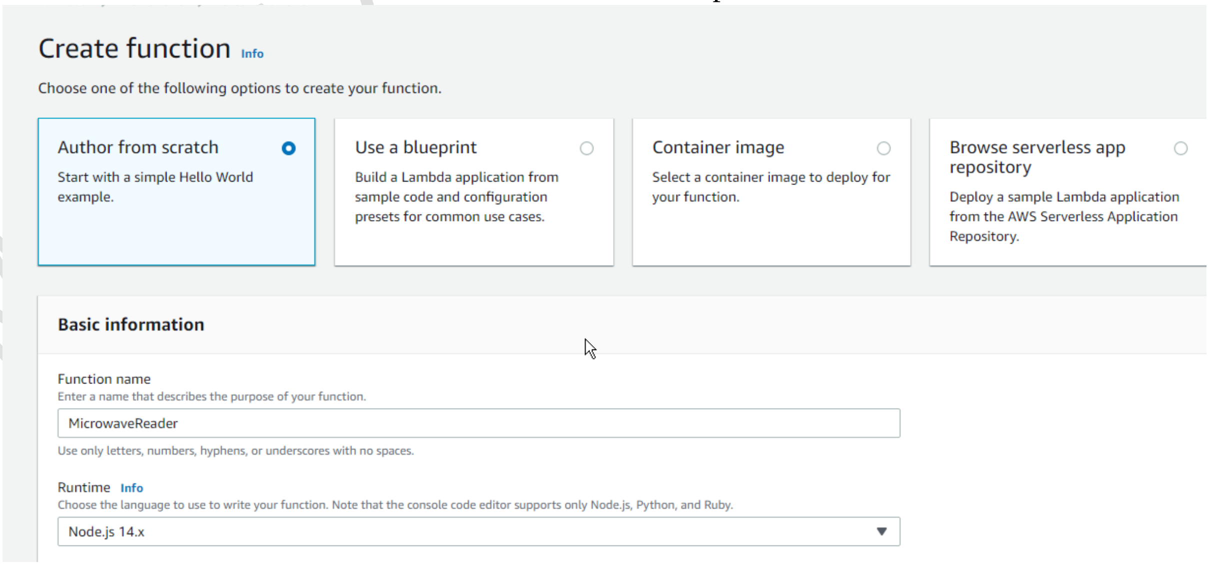
3. You will be taken to the Functions list page:



Functions (0) Last fetched 40 seconds ago

Function name	Description	Package type	Runtime	Code size	Last modified
There is no data to display.					

4. Click Create a function to start creating your first AWS Lambda function.
5. In the Create function wizard, ensure Author from scratch is selected and enter the following values in the bottom form:
   Name: MicrowaveReader
   Runtime: Node.js 12.X
   Permissions: Click Change default execution role
   Execution Role: Create a new role with basic Lambda permissions



Create function Info

Choose one of the following options to create your function.

[✓ Author from scratch] Start with a simple Hello World example.

Use a blueprint: Build a Lambda application from sample code and configuration presets for common use cases.

Container image: Select a container image to deploy for your function.

Browse serverless app repository: Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

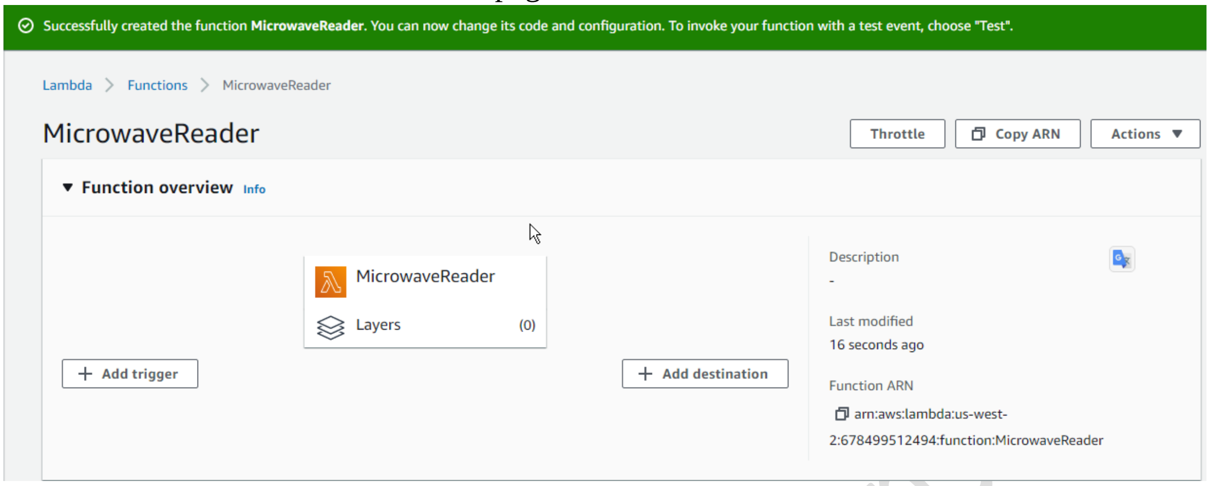
Function name
Enter a name that describes the purpose of your function.
MicrowaveReader
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 14.x

6. Click Create function.

Author: Saurav Raghuvanshi

7. You are taken to the function's details page:



Successfully created the function **MicrowaveReader**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > MicrowaveReader

MicrowaveReader

Throttle | Copy ARN | Actions ▼

Function overview Info

MicrowaveReader

Layers (0)

+ Add trigger

+ Add destination

Description

-

Last modified

16 seconds ago

Function ARN

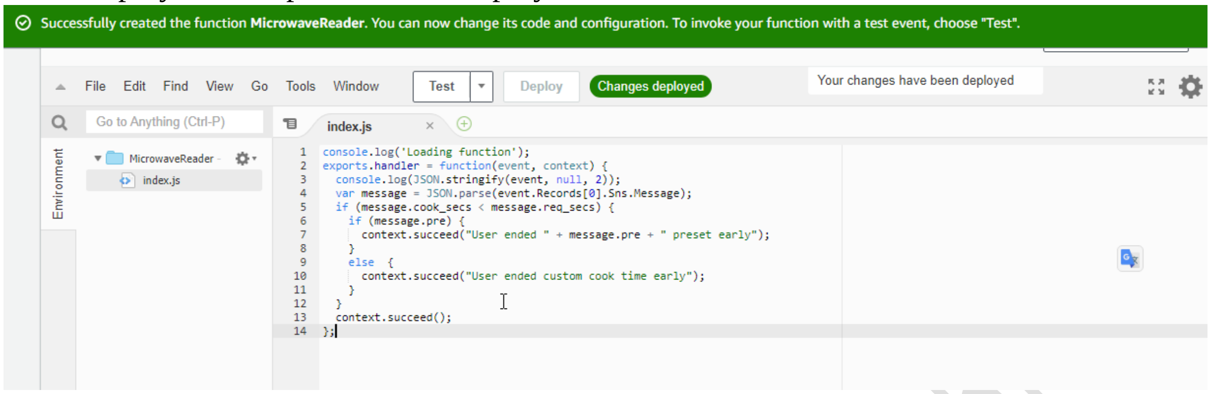
arn:aws:lambda:us-west-2:678499512494:function:MicrowaveReader

8. Scroll down to the **Code source** section, double-click the **index.js** file on the left, and overwrite the contents of the file with the following code:

```
console.log('Loading function');

exports.handler = function(event, context) {

  console.log(JSON.stringify(event, null, 2));

  var message = JSON.parse(event.Records[0].Sns.Message);

  if (message.cook_secs < message.req_secs) {

    if (message.pre) {

      context.succeed("User ended " + message.pre + " preset early");

    }

    else  {

      context.succeed("User ended custom cook time early");

    }

  }

  context.succeed();

};
```

Author: Saurav Raghuvanshi

9. Click Deploy at the top to save and deploy the Lambda function.



Successfully created the function **MicrowaveReader**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

```
1  console.log('Loading function');
2  exports.handler = function(event, context) {
3    console.log(JSON.stringify(event, null, 2));
4    var message = JSON.parse(event.Records[0].Sns.Message);
5    if (message.cook_secs < message.req_secs) {
6      if (message.pre) {
7        context.succeed("User ended " + message.pre + " preset early");
8      }
9      else  {
10       context.succeed("User ended custom cook time early");
11     }
12   }
13   context.succeed();
14 };
```

Introduction:

The Internet of Things (IoT) has a wealth of applications for AWS Lambda because most devices are low-power by design. They are also hard to update and need to be cheap enough to be embedded in fridges, toasters, televisions, robots, cars, and more. All this, of course, without driving the cost of each device too high.

You will use a microwave as an example. It will send you the cooking time, the button presses used to start the session and information about presets usage. Lambda can receive these messages over Amazon Simple Notification Service (SNS), and you can hook as many functions as you like to the same SNS topic. (For large-scale IoT installations, you should consider AWS IoT)

Lambda functions are intended to be very simple. This makes testing, debugging, updating, and maintaining Lambda backends super easy. In the example function, you are going to collect information about if the user stopped the microwave before time ran out and if a preset was used. This sort of information can help you improve the presets by collecting real-world data.

The code you used in the previous Lab Step reads an incoming SNS message and parses the JSON body to be analyzed (line 4), then checks if the microwave was stopped before the requested number of seconds (line 5), and finally if a preset was used or not (lines 6-11):

```
1   console.log('Loading function');
2   exports.handler = function(event, context) {
3     console.log(JSON.stringify(event, null, 2));
4     var message = JSON.parse(event.Records[0].Sns.Message);
5     if (message.cook_secs < message.req_secs) {
6       if (message.pre) {
7         context.succeed("User ended " + message.pre + " preset early");
8       }
9       else  {
10        context.succeed("User ended custom cook time early");
11      }
12    }
13    context.succeed();
14  };
```

Instructions:

10. Click Test at the top

Author: Saurav Raghuvanshi

11. In the Configure test event form, enter the following values into the form:

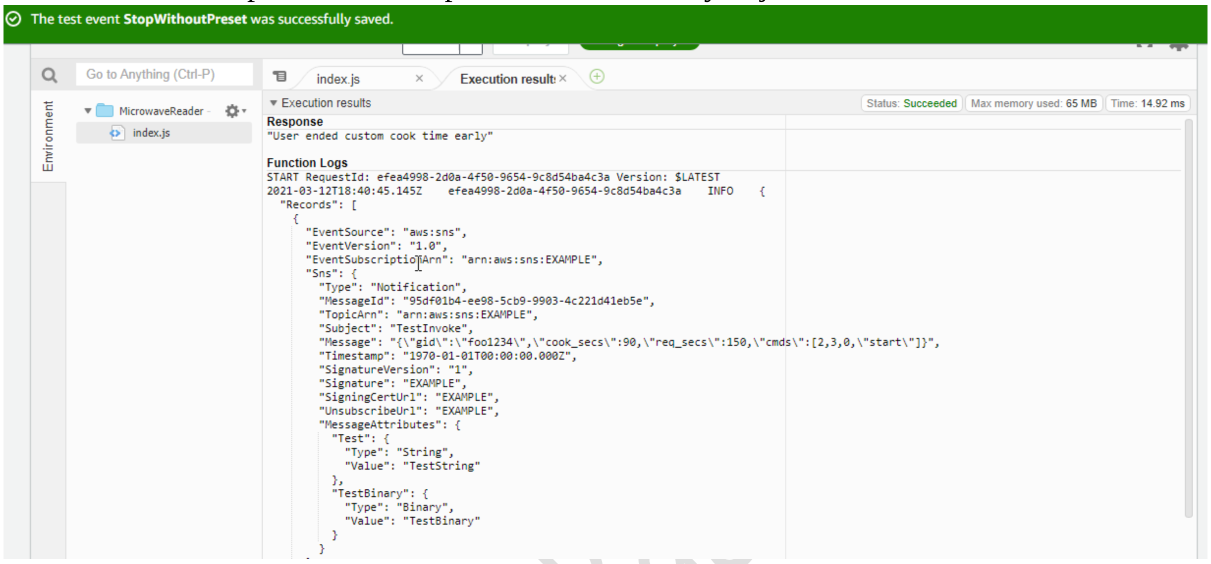
- **Event name:** StopWithoutPreset
- **Event body:** Enter the following JSON into the event editor at the bottom of the form

```
{
  "Records": [
    {
      "EventSource": "aws:sns",
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:EXAMPLE",
      "Sns": {
        "Type": "Notification",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "TopicArn": "arn:aws:sns:EXAMPLE",
        "Subject": "TestInvoke",
        "Message":
"{\"gid\":\"foo1234\",\"cook_secs\":90,\"req_secs\":150,\"cmds\":[2,3,0,\"start\"]}",
        "Timestamp": "1970-01-01T00:00:00.000Z",
        "SignatureVersion": "1",
        "Signature": "EXAMPLE",
        "SigningCertUrl": "EXAMPLE",
        "UnsubscribeUrl": "EXAMPLE",
        "MessageAttributes": {
          "Test": {
            "Type": "String",
            "Value": "TestString"
          },
          "TestBinary": {
            "Type": "Binary",
            "Value": "TestBinary"
          }
        }
      }
    }
  ]
}
```

Author: Saurav Raghuvanshi

In practice, the event body comes from SNS or whatever event source you configure. For now, you will just use Lambda's testing functionality to send in sample event data. The Message property sets the cook time (cook_sec) to be less than the request seconds (req_sec). It also sets the commands (cmds) to the digits entered for the number of seconds followed by start, indicating a preset was not used)

12. Click Test at the top to run the StopWithoutPreset test you just created:



The test event **StopWithoutPreset** was successfully saved.

Response

```
"User ended custom cook time early"
```

Function Logs

```
START RequestId: efea4998-2d0a-4f50-9654-9c8d54ba4c3a Version: $LATEST
2021-03-12T18:40:45.145Z    efea4998-2d0a-4f50-9654-9c8d54ba4c3a    INFO    {
  "Records": [
    {
      "EventSource": "aws:sns",
      "EventVersion": "1.0",
      "EventSubscriptionArn": "arn:aws:sns:EXAMPLE",
      "Sns": {
        "Type": "Notification",
        "MessageId": "95df01b4-ee98-5cb9-9903-4c221d41eb5e",
        "TopicArn": "arn:aws:sns:EXAMPLE",
        "Subject": "TestInvoke",
        "Message": "{\"gid\":\"foo1234\",\"cook_secs\":90,\"req_secs\":150,\"cmds\":[2,3,0,\"start\"]}",
        "Timestamp": "1970-01-01T00:00:00.000Z",
        "SignatureVersion": "1",
        "Signature": "EXAMPLE",
        "SigningCertUrl": "EXAMPLE",
        "UnsubscribeUrl": "EXAMPLE",
        "MessageAttributes": {
          "Test": {
            "Type": "String",
            "Value": "TestString"
          },
          "TestBinary": {
            "Type": "Binary",
            "Value": "TestBinary"
          }
        }
```

13. You can see the User ended custom cook time early near the top of the Details. This matches the expected output given the test event configuration.

Summary:

In this Lab Step, you tested the Lambda function you created by providing a test event. The event was configured to test a user stopping a microwave early without using a preset. The function only returns a string based on the conditions, but you could be counting incidents in DynamoDB, sending notifications to users, or any number of other things. Of course, the cooking time on a microwave isn't incredibly interesting data. Cheap sensors make it feasible to collect all sorts of more interesting data and use Lambda to act on it in near-real-time.

If you have extra time in your Lab session, you can try modifying the test event to exercise different code paths and check the logs after running the test to confirm everything works as expected. Be sure to leave a few minutes for the final Lab Step that validates the work you did in this Lab.