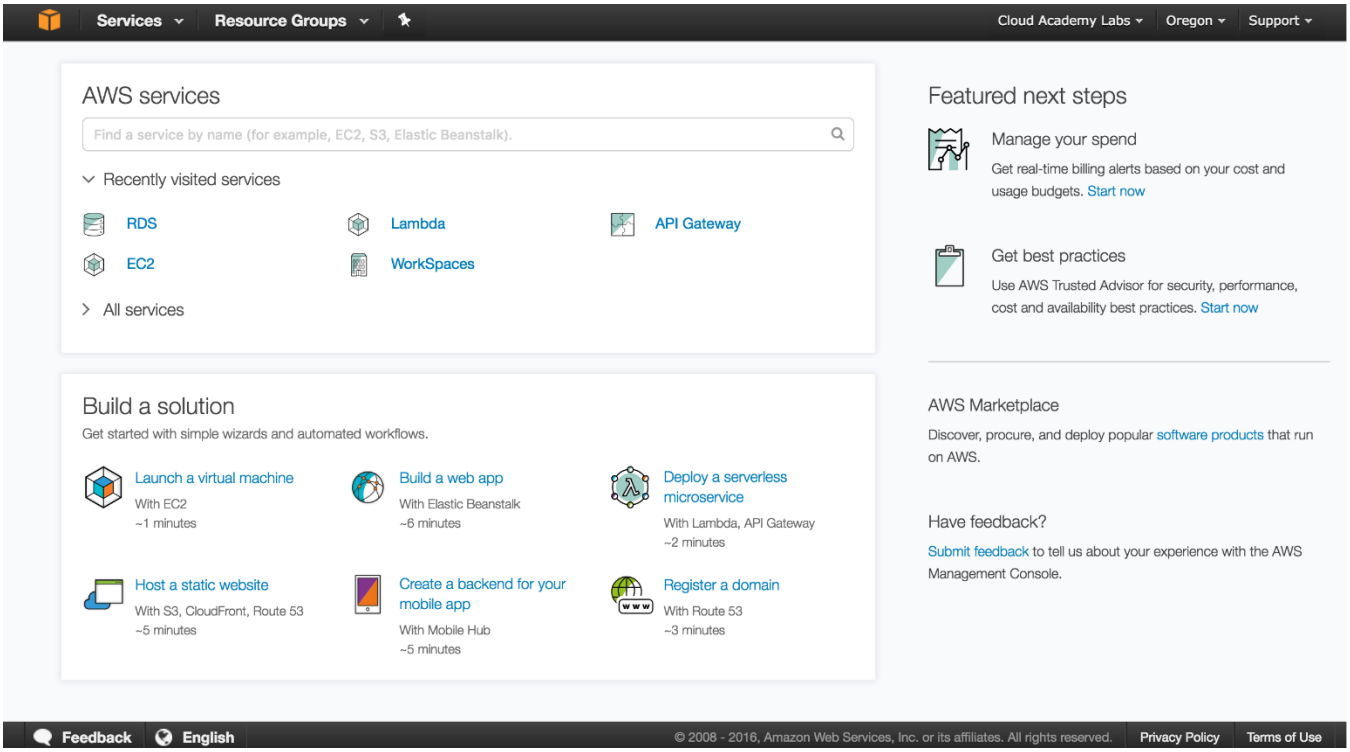
AWS Lambda

Step 1 Logging in to the Amazon Web Services Console

Introduction

This Lab experience involves Amazon Web Services (AWS), and you will use the AWS Management Console to complete all the Lab Steps.



The AWS Management Console is a web control panel for managing all your AWS resources, from EC2 instances to SNS topics. The console enables cloud management for all aspects of the AWS account, including managing security credentials, and even setting up new IAM Users.

Instructions

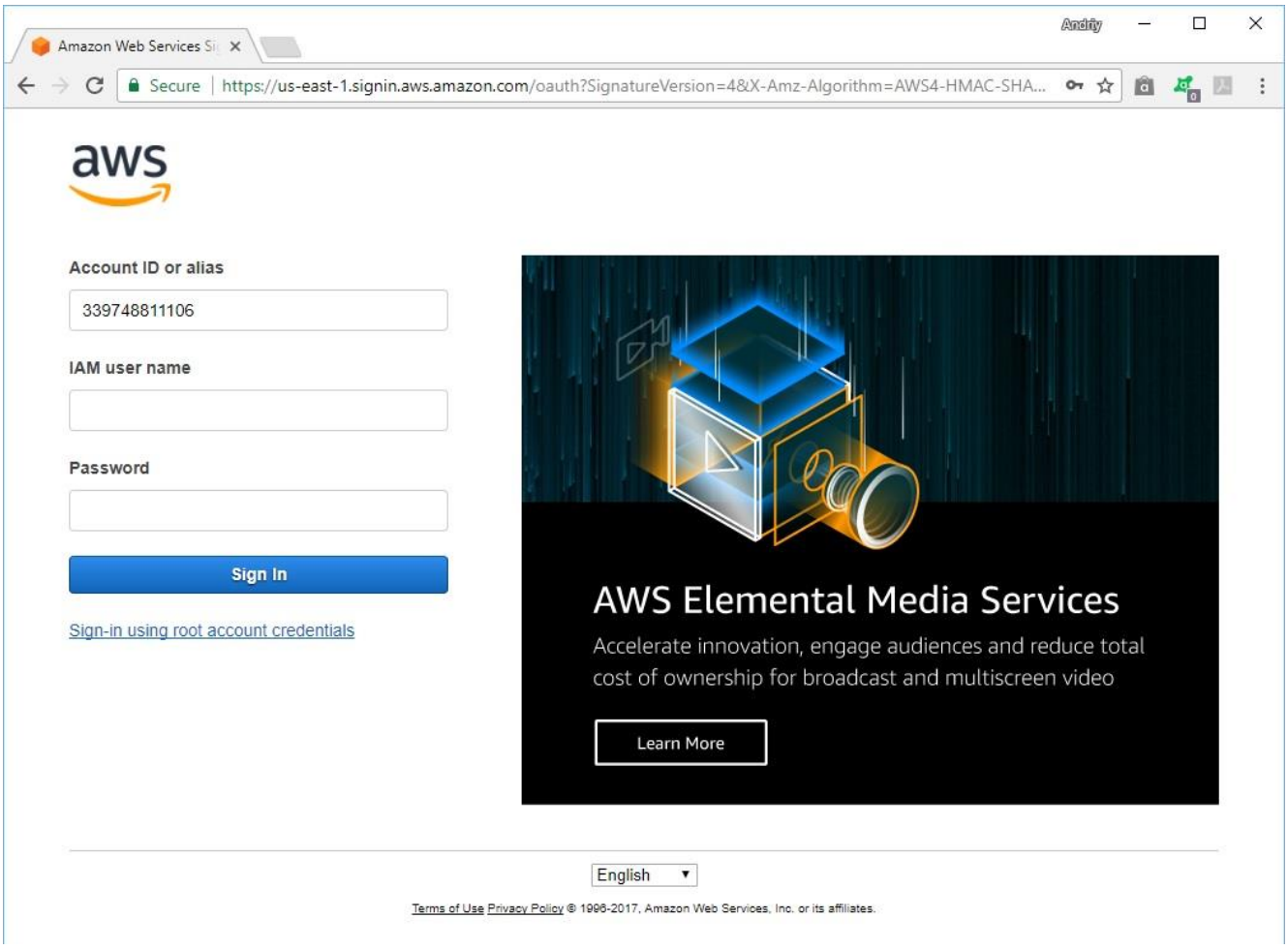
1. To start the Lab experience, open the Amazon Console by clicking this button:

Open AWS Console

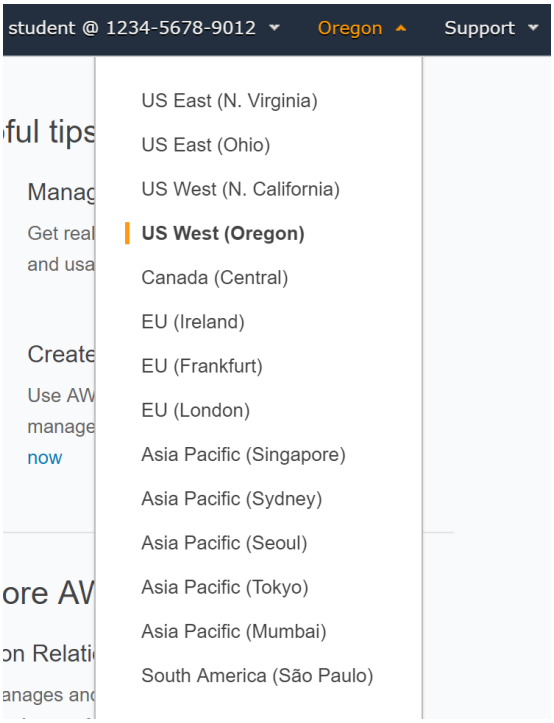
2. Enter the following credentials created just for your Lab session, and click **Sign In**:

- **Account ID or alias**: Keep the pre-populated value
- **IAM user name**: *student*
- **Password**: *$training0*

After that you will be redirected to AWS Login page:



3. Select the **US West (Oregon)** region using the upper right drop-down menu on the AWS Management Console:



Amazon Web Services are available in different regions all over the world, and the console lets you provision resources across multiple regions. You usually choose a region that best suits your business needs to optimize your customer's experience, but you must use the **US West (Oregon)** for this Lab.

Step 2 Welcome to AWS Lambda

AWS Lambda is a relatively new service, it was released from Developer Preview in April 2015. You can use it to design advanced materialized views out of DynamoDB tables, react to uploaded files on S3, process 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 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.

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, Lambda, Simple Workflow Service, and more vying for your workload. Before we start on this lab, let's see 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 Javascript code that will run and specify what events should trigger your code. Scaling Lambda functions happens automatically; AWS provisions more instances as needed and only charges you for the time your function runs.

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

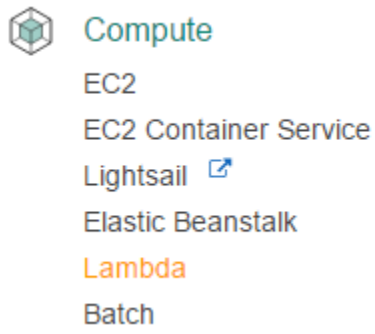
In this lab, we will set up a Lambda function, learn how to test code in the AWS Console, and discuss different event sources for bringing data into Lambda. Functions like the ones we'll write in this lab can be used to help keep data in sync, fan out writes to users' news feeds, or update indexes in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

Step 3 Create a new Lambda function

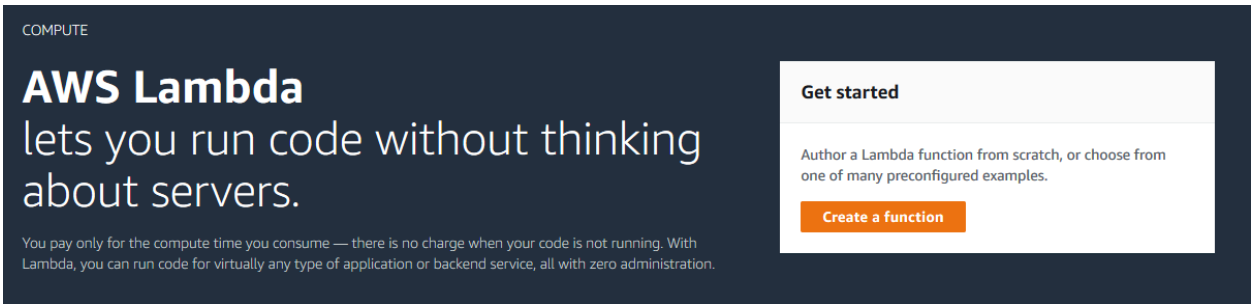
A Lambda function is a small unit of computation that can execute in parallel, in a stateless fashion. This is only partially true though, since you can potentially have some shared initialization code, at the container level.

Let's now create a new function and play with it for the rest of this lab.

To get started, click on the "Lambda" icon in the Compute category (it's on top left in your AWS Dashboard):

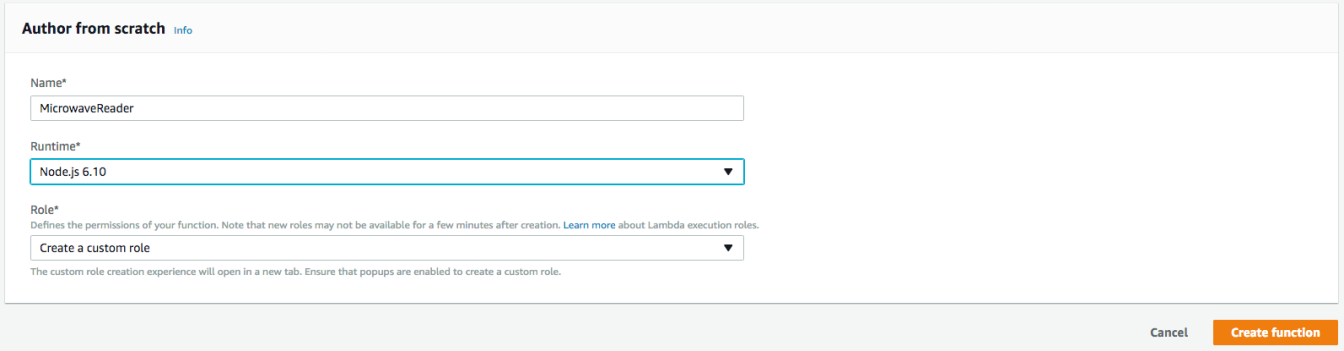


If no function exists already in the selected region, you will see the following landing page:



Click **Create a Lambda function** and start the creation wizard.

Name your function `MicrowaveReader` and select the `Create a custom role` from the **Role** dropdown menu. This is an IAM role that grants the minimum permissions for the Lambda function to run; just the ability to save logs.



By choosing `Create a custom role`, you will be asked to name the new role in a new tab.

You can leave the default role name and click **Allow** to be redirected back to the Lambda Wizard.

Select the role you just created and click the **Create Function** button.

In this page we will finish configuring our Lambda Function. Fill the followings as such:

Function code:

- **Runtime**: Node.js
- **Code:** *Paste the following*

```
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();
};
```

Step 4 IoT Lambda function example

The Internet of Things 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.

We will use a microwave as our example. It will send us the cooking time, the button presses used to start the session and information about presets usage. Lambda can receive these messages over SNS, and you can hook as many functions as you like to the same SNS topic.

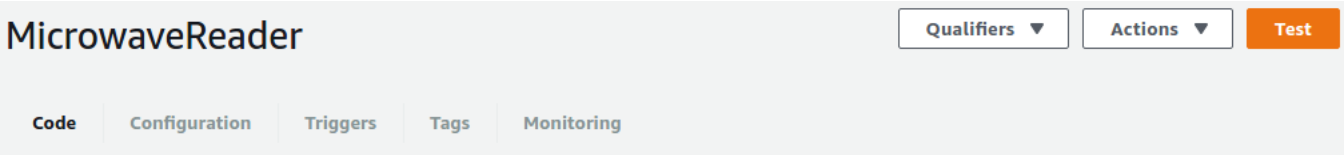
Lambda functions are intended to be very simple. This makes testing, debugging, updating, and maintaining Lambda backends super easy. In our example, we are just going to collect information about whether a preset was used, and if they stopped the microwave before time ran out. This sort of information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

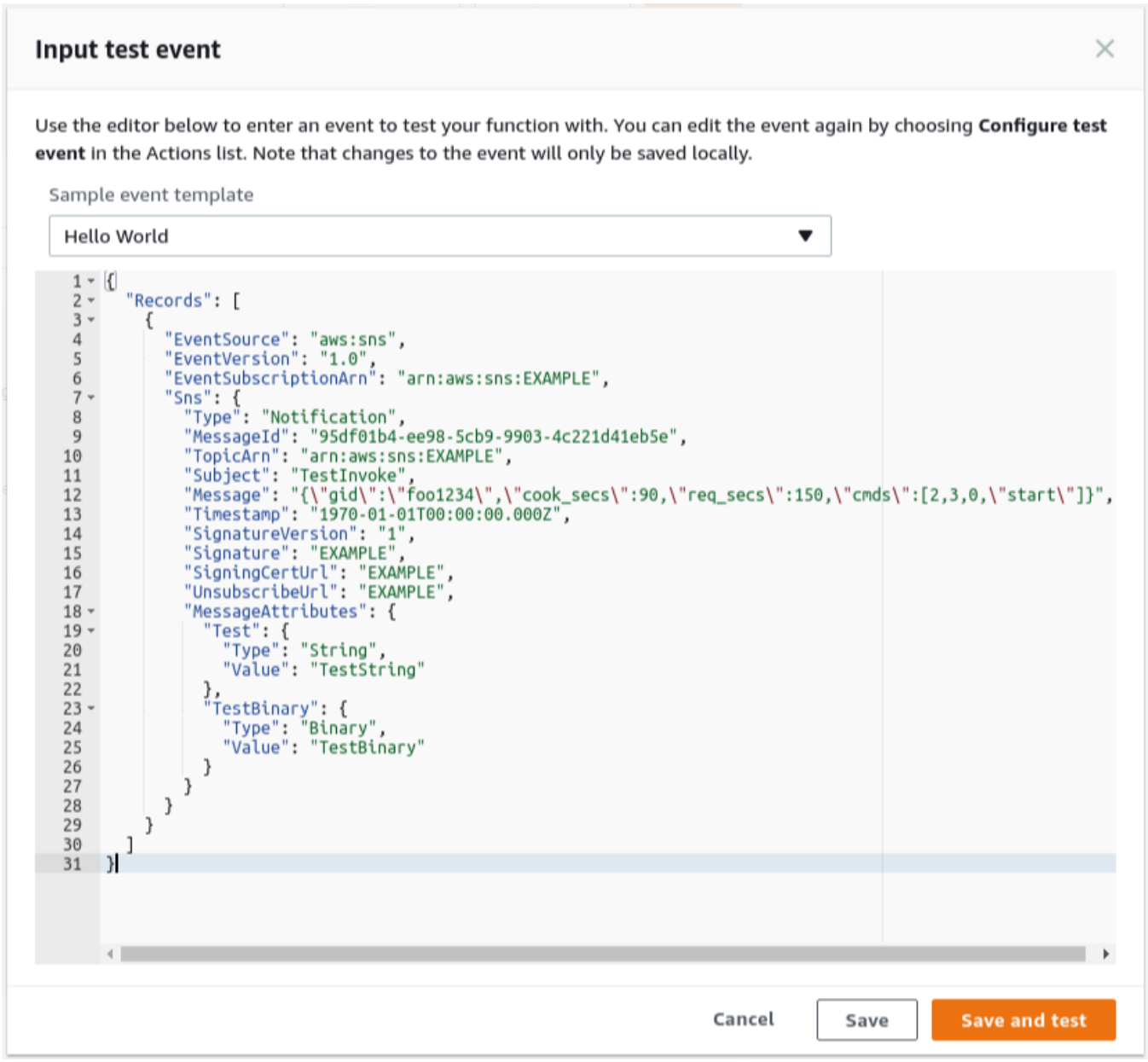
```
console.log('Loading function');

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();
};
```

The code above just reads an incoming SNS message and parses the JSON body to be analyzed.

You can check if everything works as expected using the Test feature. Click the **Test** blue button and you'll see the "Input sample event" popup.



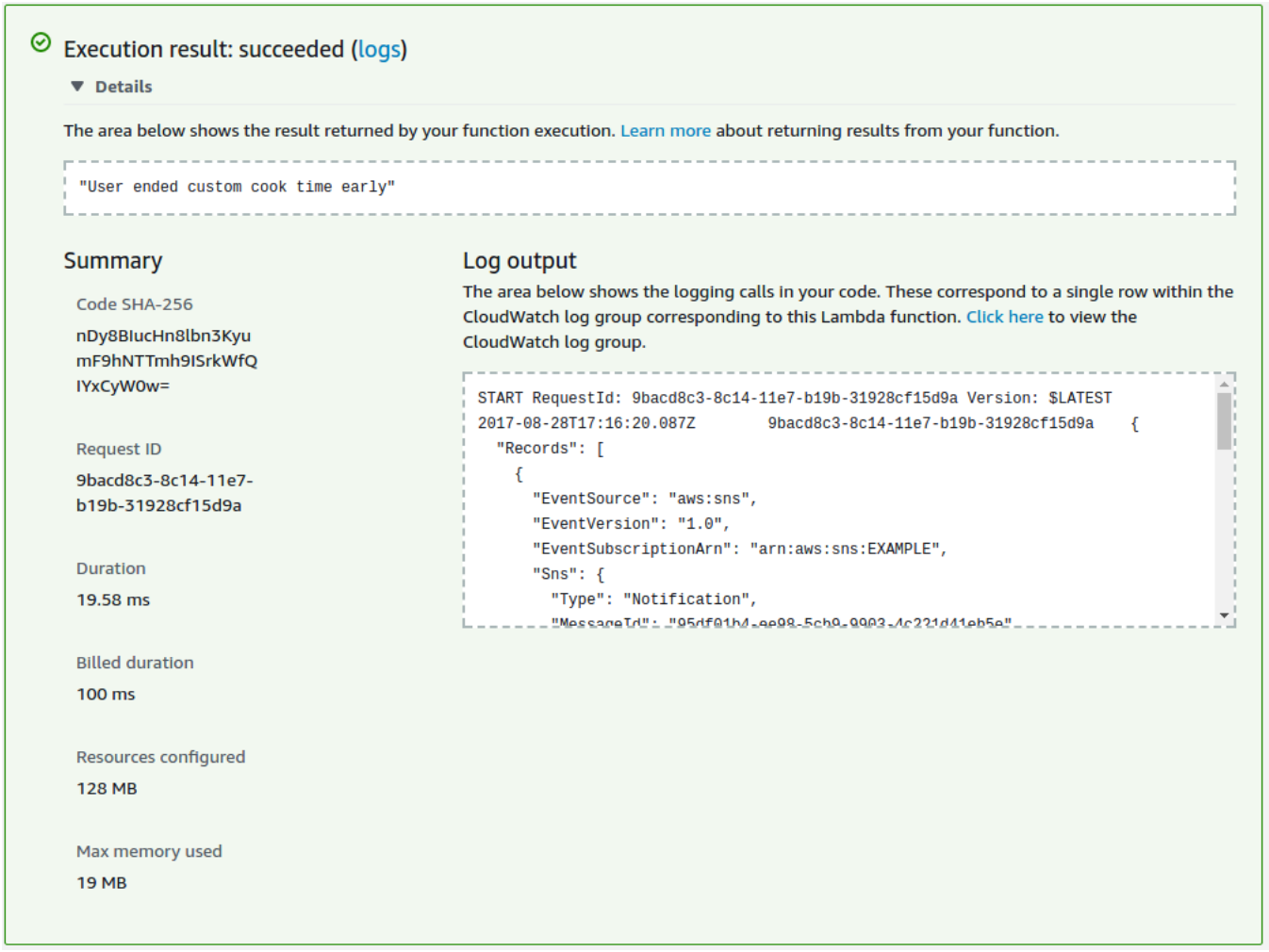


To test the new function, we need to provide some sample data. Normally, it would be provided by SNS or whatever event source you configure. For now, we will just use Lambda's testing functionality to send in data. Paste the JSON below into the big textarea:

```
{
  "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,\"sta
rt\"]}",
        "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"
          }
        }
      }
    }
  ]
}
```

Press "**Create**" and once the test is created press "**Save**" and the "**Test**" to run the function with the input data and you will see the execution result "*User ended custom cook time early*" in the box below.

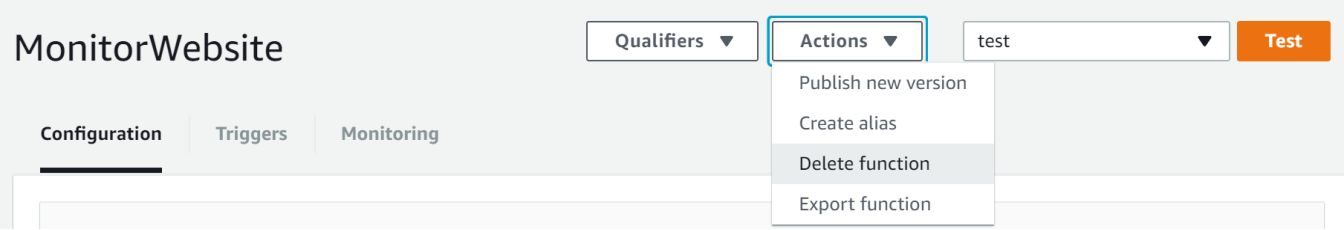


In this function, all we're doing is returning a string based on the conditional, but we could be counting incidents in DynamoDB, sending notifications to users, or any number of other things. Of course, the cooking time on a microwave isn't incredibly interesting data. Cheap sensors make it feasible to collect all sorts of more interesting data and use Lambda to act on it in near-real-time.

Step 5 Clean up the Lambda Function

Now that we have run the function, let's clean up what we created.

From the Lambda control panel, look for the **Actions** drop-down menu and select **Delete function**:



Click **Delete** in the confirmation modal.

Keep in mind that when Lambda is not running the only thing that is charged for is the code storage itself.