

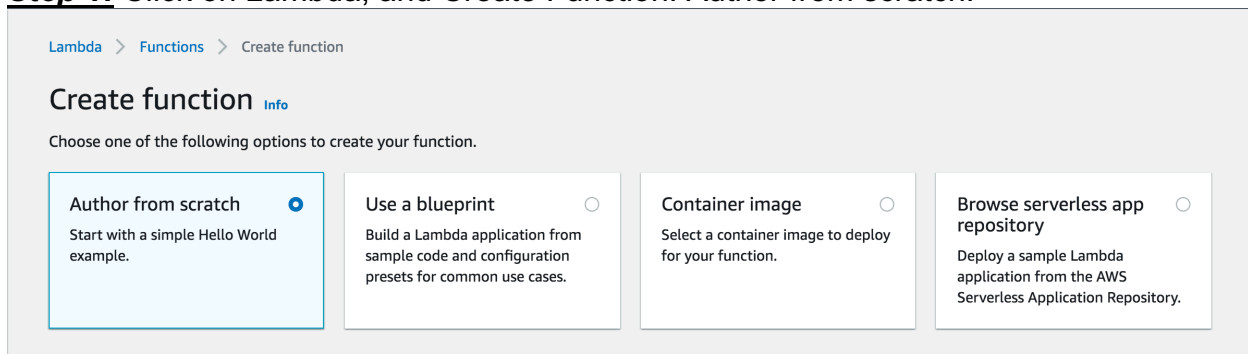
## CPS847 Group 17: Lambda Function Report

### → Part 6: Create AWS Lambda function that reads in JSON: Write a report on how the lambda function works.

AWS Lambda is an event-driven, server less computing platform. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code. It runs code without provisioning or managing servers (Amazon has its own servers that are used). Lambda only runs your code when needed and scales automatically, from a few requests per day to thousands per second. You basically only pay for the compute time that you consume and there is no charge when your code is not running.

Since AWS Lambda is event-driven, that basically implies that a Lambda function can be triggered during the occurrence of an event, so the application flow is mainly driven by events. It can be triggered by outside events such as other requests that are from other platforms such as changes to data in an Amazon Simple Storage Service (Amazon S3) bucket or an Amazon DynamoDB table.

#### **Step 1:** Click on Lambda, and Create Function: Author from scratch.



The screenshot shows the AWS Lambda 'Create function' page. At the top, there is a breadcrumb trail: 'Lambda > Functions > Create function'. Below this, the heading 'Create function' is followed by a small 'Info' link. A sub-heading reads 'Choose one of the following options to create your function.' There are four selectable options, each in a box with a radio button:

- Author from scratch** (selected): 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.

**Step 2:** We will name the function as “lambdaFunction”, choose the language as Python 3.7 and then click Create Function at the bottom of the page.

**Basic information**

**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.

**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.

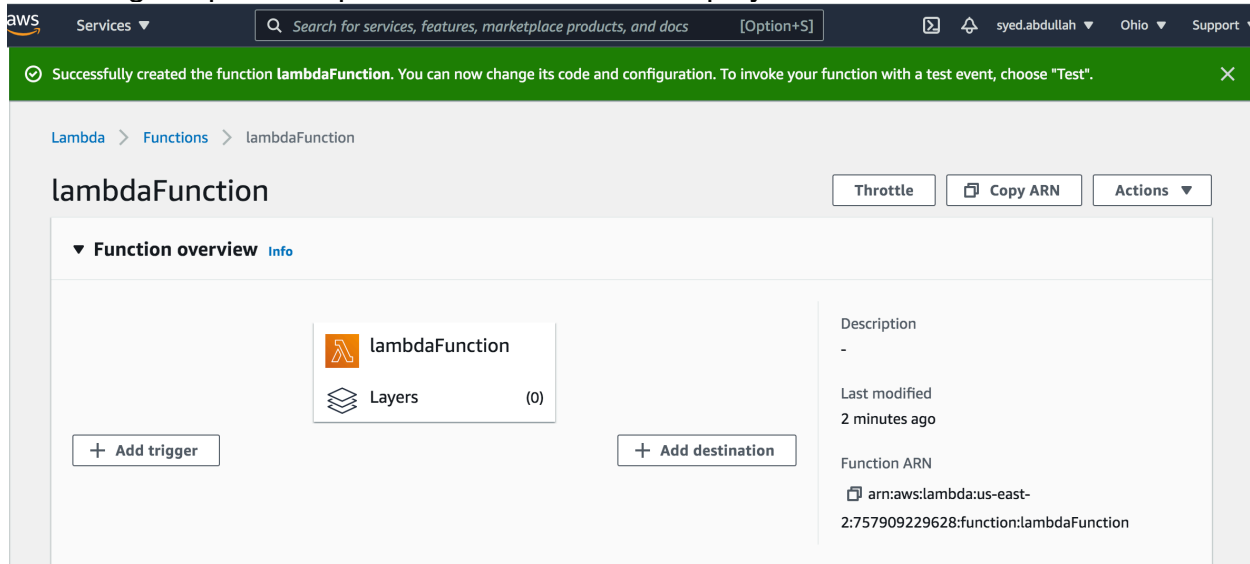
**▼ Change default execution role**

**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

 Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.

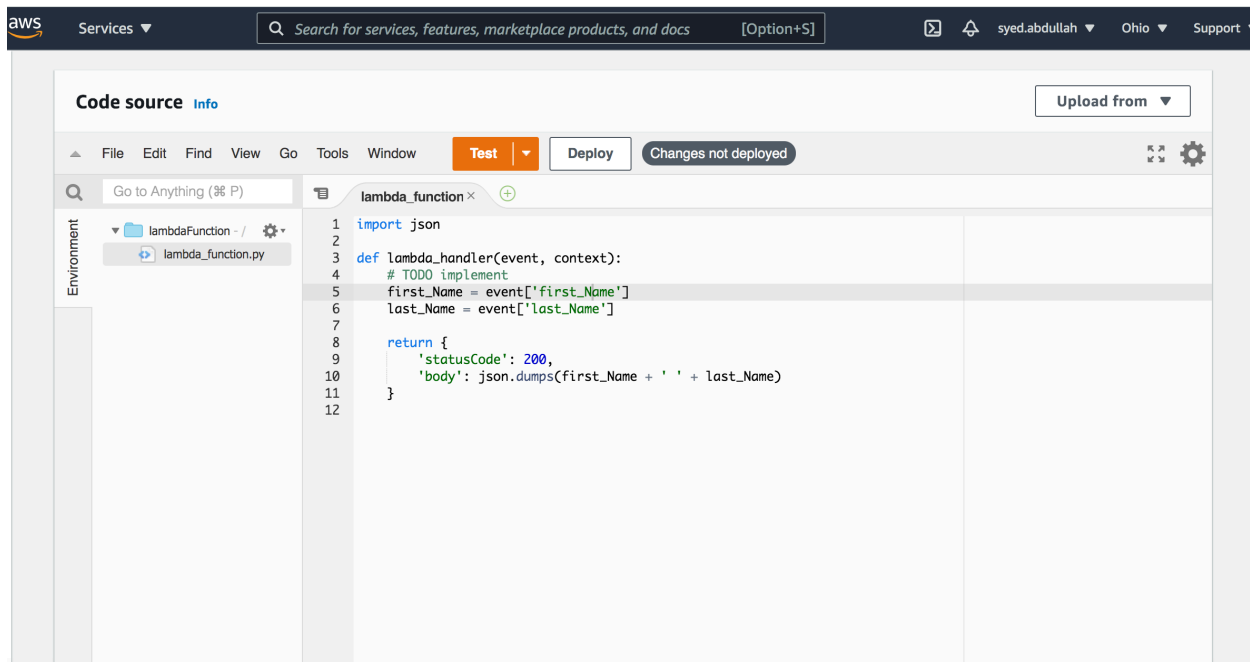
**Step 3:** Clicking Create Function will lead us to this, after which we scroll down, we can see the source code. We edit the template source code and put in our own code, according the part 6 requirements and then hit deploy.



Note: Trigger and Destination: event-driven parts of the lambda function. (Continued onto next page).

A Trigger is a Lambda resource that you configure to invoke your function in response to lifecycle events, external requests, or on a schedule. A Lambda Function can have multiple triggers, and each trigger acts as a client invoking your function independently.

A Destination is a Lambda resource which you can use to route asynchronous function results as an execution record to a destination resource without writing additional code. An execution record contains details about the request and response in JSON format. Lambda can also be configured to route different execution results to different destinations.



In the following source code, the Lambda function takes in the parameters: event and context. Event is the JSON which is inputted, and Context is a custom context class which can be used to find out logistics such as the function name, memory limit, the context from the client while the Lambda Function is running.

In order to test the code, the code must be typed without syntax errors, and we must hit “Deploy”, so it can be saved and then be tested to run.

**Step 4:** After we hit deploy, we hit Test, which will open a window “Configure test event”. We write the event name as “testlambdaFunction”, edit the source code and code accordingly for the test and input is then shown below, after which we click “Create” (It is not visible in this screenshot).

Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

Create new test event

Edit saved test events

Event template

hello-world

Event name

testlambdaFunction

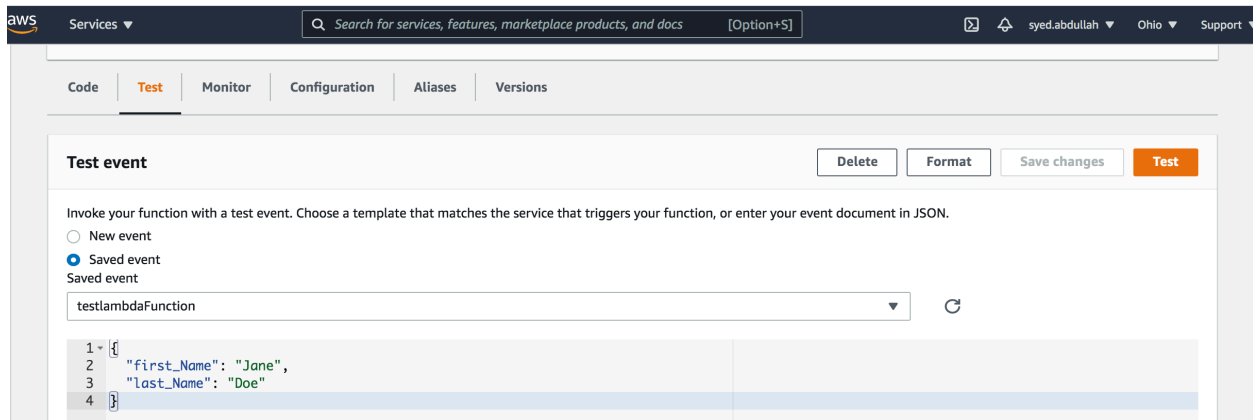
1 {

2   "first\_Name": "Jane",

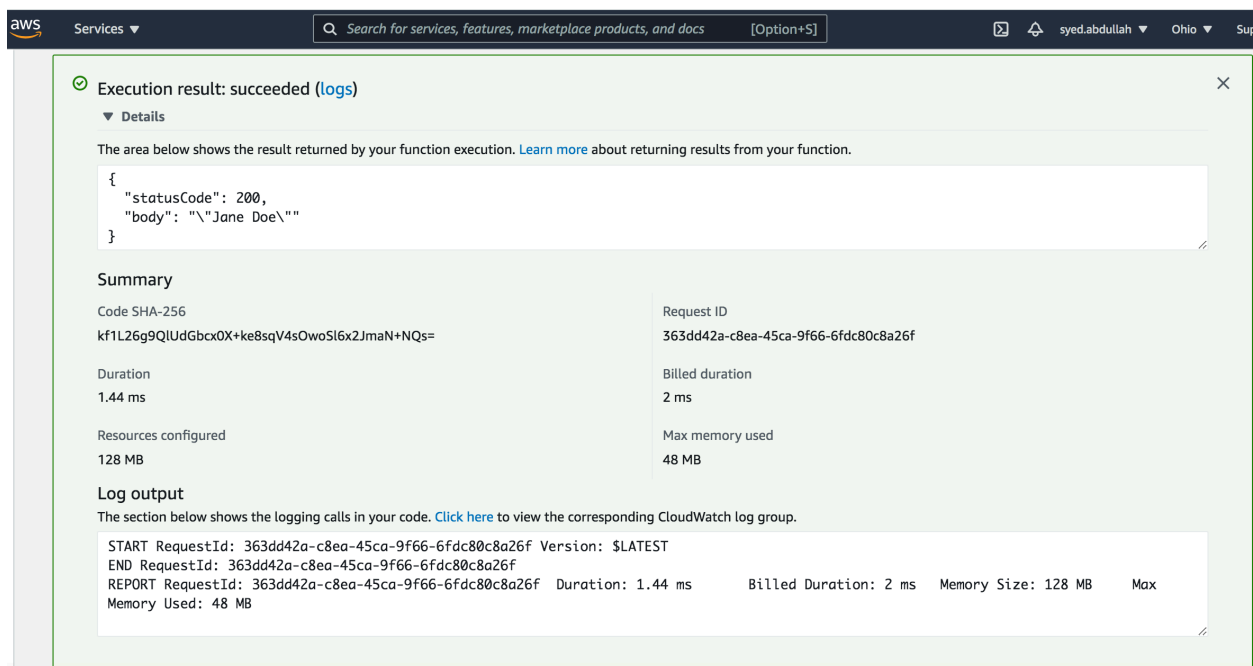
3   "last\_Name": "Doe"

4 }

**Step 5:** Then we can go to the 2<sup>nd</sup> option: Test tab and hit “Test” in the right side of the screen as shown in 1<sup>st</sup> screenshot below, which will give us its Execution result. Execution Result is shown in the 2<sup>nd</sup> screenshot below, it succeeded.



When you click Test for this test event with JSON, which contains first\_Name and last\_Name and their values “Jane” and “Doe”, it basically runs the test case against the Lambda Function that we created “lambdaFunction”.



Since AWS Lambda is server less, the execution is basically run on the AWS platform, so the resources such as memory, the execution time is all automatically done, and is not needed to be set by you. Now after deploying the code and then hitting test on the test event, the execution results are shown, as highlighted in this screenshot above. After

observing this screenshot, you can see that the response given by the Lambda function:

```
{  
  "statusCode": 200,  
  "body": "\ "Jane Doe\ ""  
}
```

Now the "statusCode": 200 is basically the HTTP OK response code, which basically implies that there were no errors in the test. While "body": "\ "Jane Doe\ "" is the result that is returned by the function after running through the code and the input test cases. You can also see other information on this regards to the execution result of this function which includes the Duration, Billed duration, Memory Size, Memory used along with any output which can be passed onto other functions using destination resource in the Lambda function.