Serverless concept

Serverless is a managed, native cloud architecture that shifts operational responsibility of servers to the cloud. That allows you, the developer, to focus on building applications and services without even thinking about the servers and software needed to run your code.

Cost-efficiency is intertwined with serverless because you only pay for what you use, whereas even virtual servers cost money to run 24/7.

In this lesson, we will use a serverless frontend called **API Gateway** which will send the request to a **Lambda Function**, which will serve as our backend.

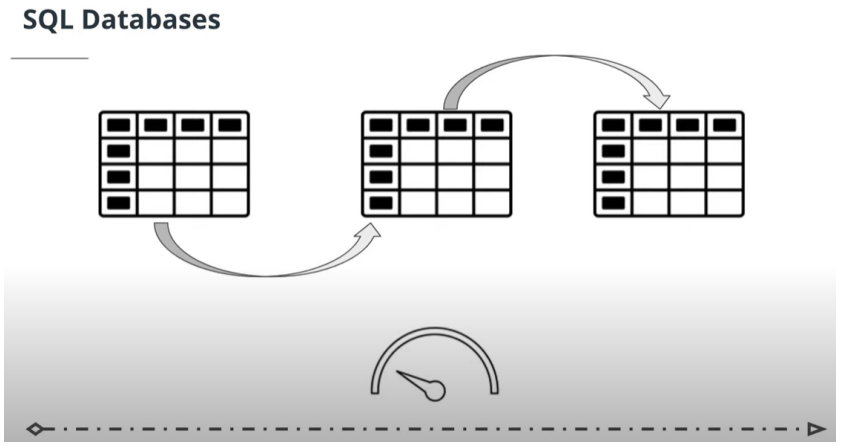
That Lambda will either set or get the user's name from a managed, in-memory database service. The Lambda will return the name back to the user via the API Gateway.

Intro to Cloud Computing https://classroom.udacity.com/courses/ud080/lessons/6860cb92-05c5-4de6-bba8-679709fb21f0/concepts/3425934a-5c11-4136-b5be-0df48c937f7a

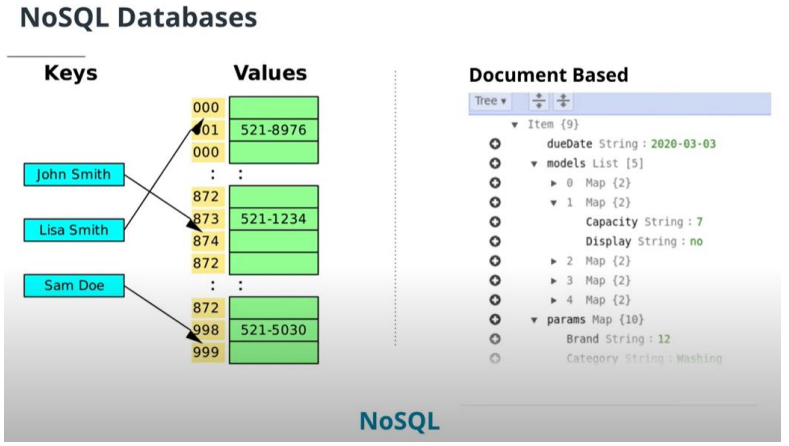
ElastiCache

ElastiCache

- Managed, in-memory NoSQL database
- For security, instances are deployed to a private subnet by default and lack a publicly reachable address
- Redis is an open-source data store, whereas ElastiCache is AWS's proprietary service the hosts Redis databases for AWS users
- Great choice for real-time applications such as gaming, chat, and video
- Easily scalable and replicable



Rational databases
structured data



Not only SQL databases
flexible data requirements

Exercise: ElastiCache For Redis

In this exercise, you will launch a Redis instance on ElastiCache. You will continue to use this instance in the next exercise.

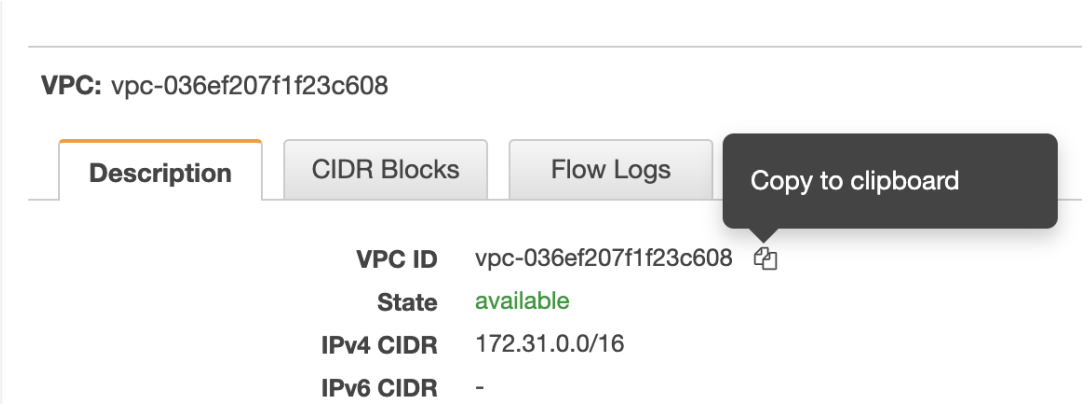
Instructions

- **Single node** with **no replication**
- **t2.micro** type
- **us-east-1 N.Virginia** region
- Setup on the **default VPC**
- No encryption either **in transit** (No Auth) or **at-rest**
- Name the node **redis**

Before you start, you need to get the Virtual Private Cloud ID of the default VPC:

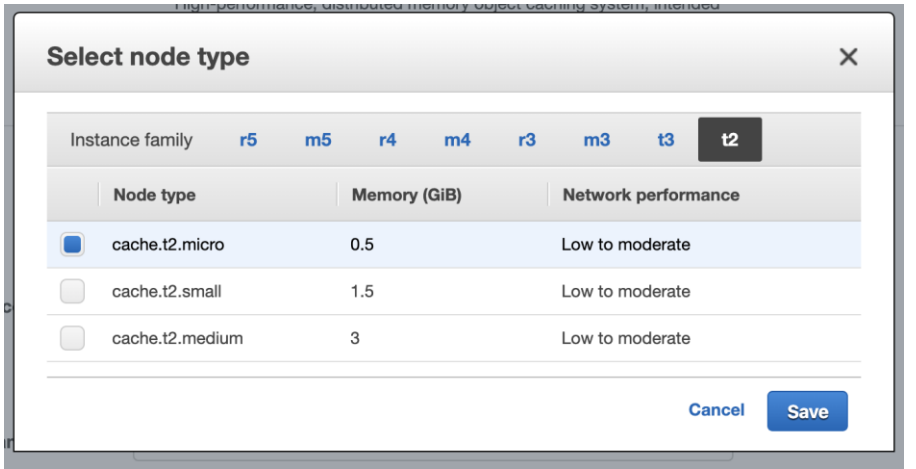
Default VPC ID

1. Open the VPC console
2. Note the default VPC ID

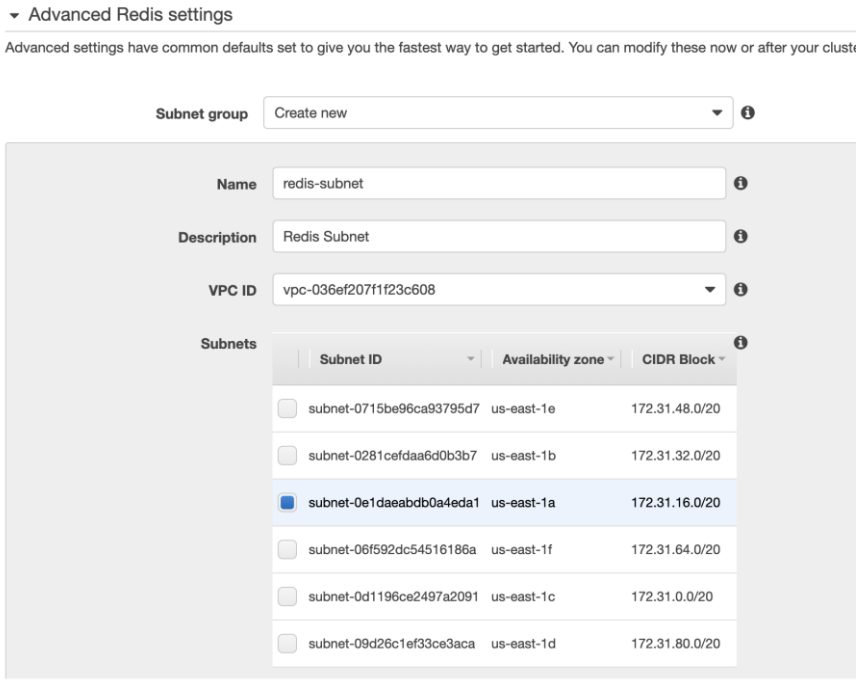


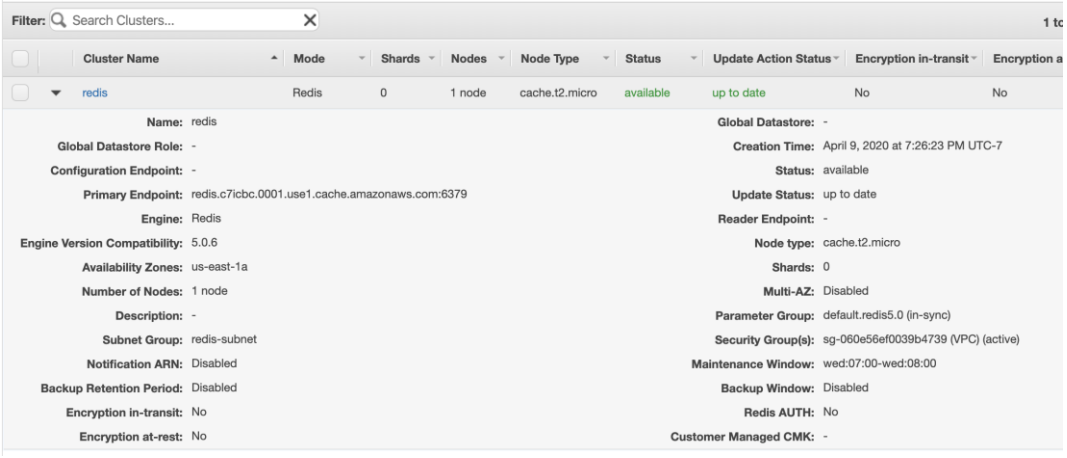
Create the ElastiCache Redis Instance

1. Open the ElastiCache console
2. Change the region on the top right to be **us-east-1 N.Virginia**
3. Click **Redis** on the sidebar menu, then click the **Create** button
4. Name the instance **redis**
5. Change the number of replicas to 0
6. Change the **Node type** to be **t2.micro**
7. Save



8. Under the Advanced Redis Settings, set the **Name** to be **redis-subnet**
9. Set the **Description** to **Redis Subnet**
10. Change the VPC ID to the default VPC ID that you noted previously
11. Select **us-east-1a** as the subnet



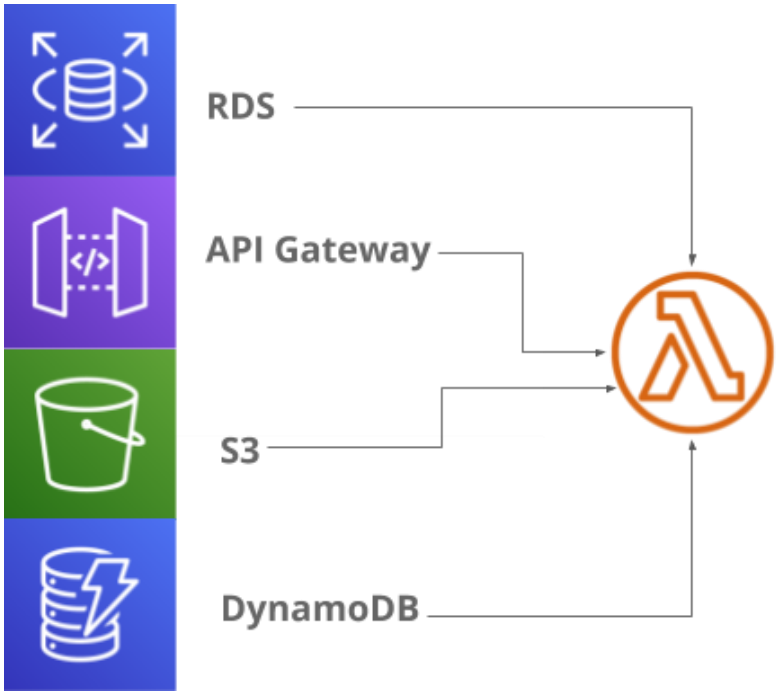


12. Scroll down to the **Backup** section and deselect **Enable Automatic backups**
13. Click on the **Create** button

Wait a few minutes for the instance to initialize. You can refresh the dashboard using the refresh arrows icon in the top-right corner of the dashboard.

Lambda Function

- Serverless fast solution to run code for a task
- Write code directly in the AWS console or upload a zip package with dependencies
- Supports multiple code languages
- Can be tested from the console



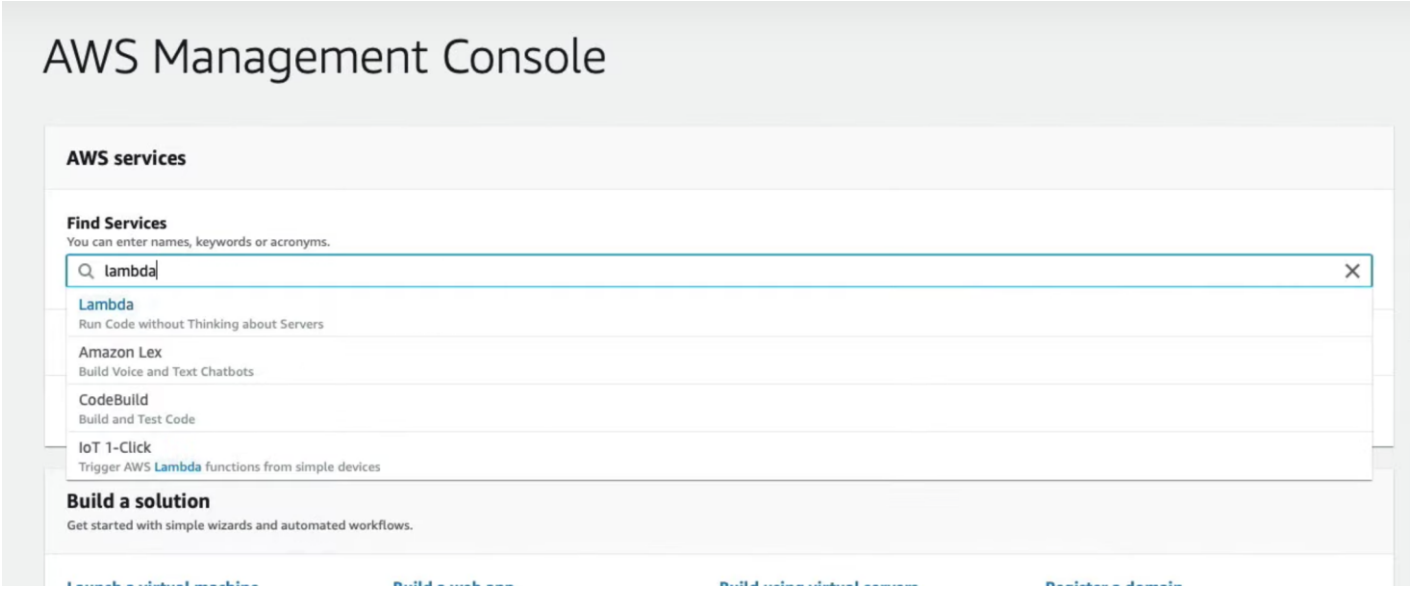
Exercise: Hello from Lambda

In this exercise, you will create a basic Lambda Function.

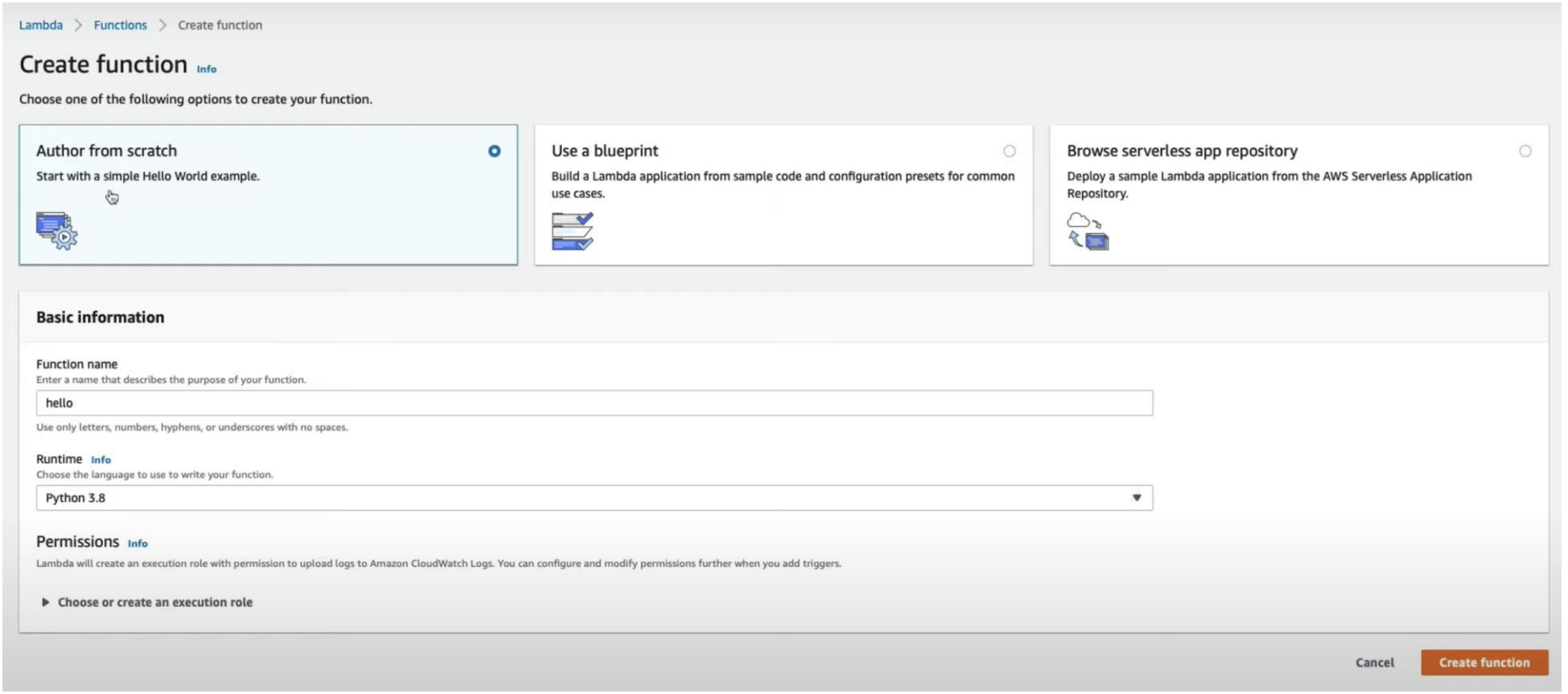
Instructions

- Create a Lambda Function from scratch
- The Lambda Handler should be set to the default **lambda_function.lambda_handler**
- Manually trigger the Lambda via a test event and verify the message "Hello from Lambda“

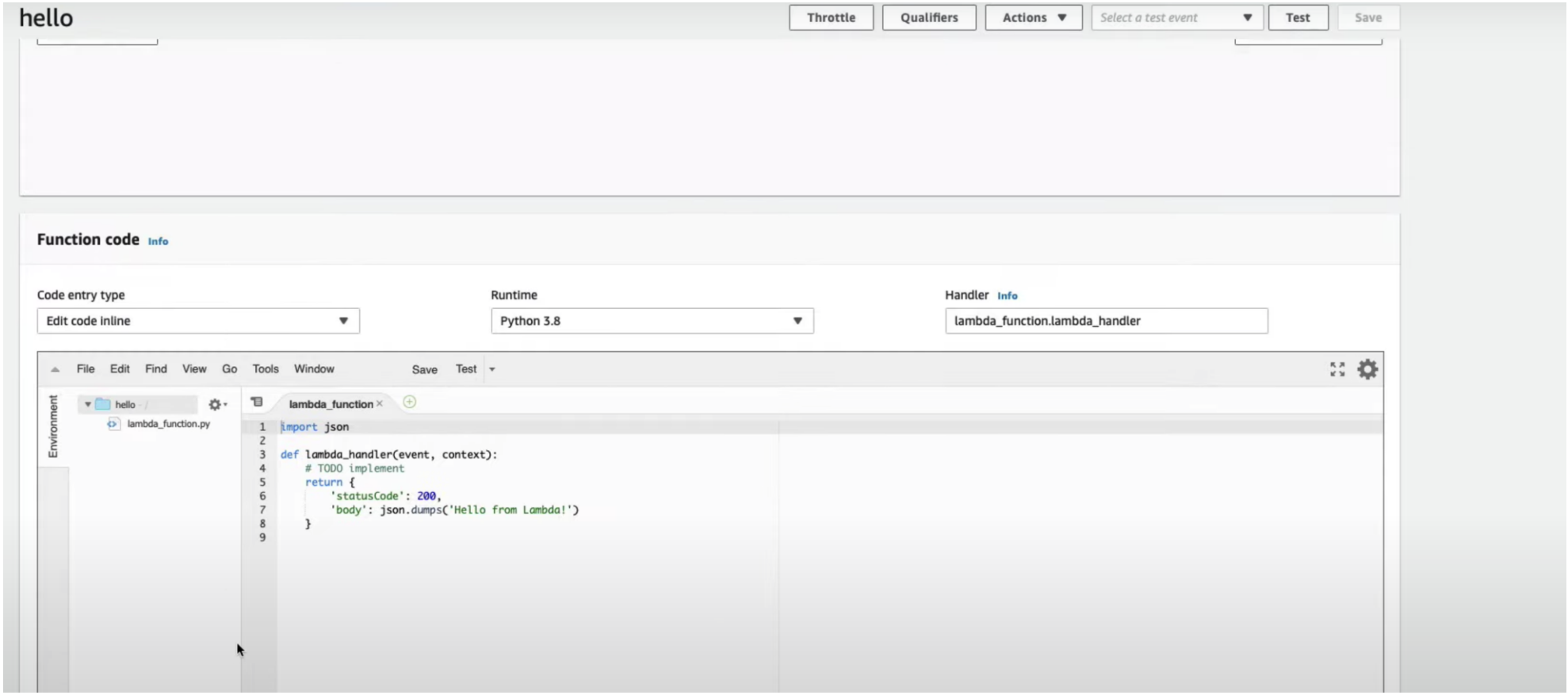
1. Navigate to the Lambda Console by searching for **Lambda** under **Find Services**



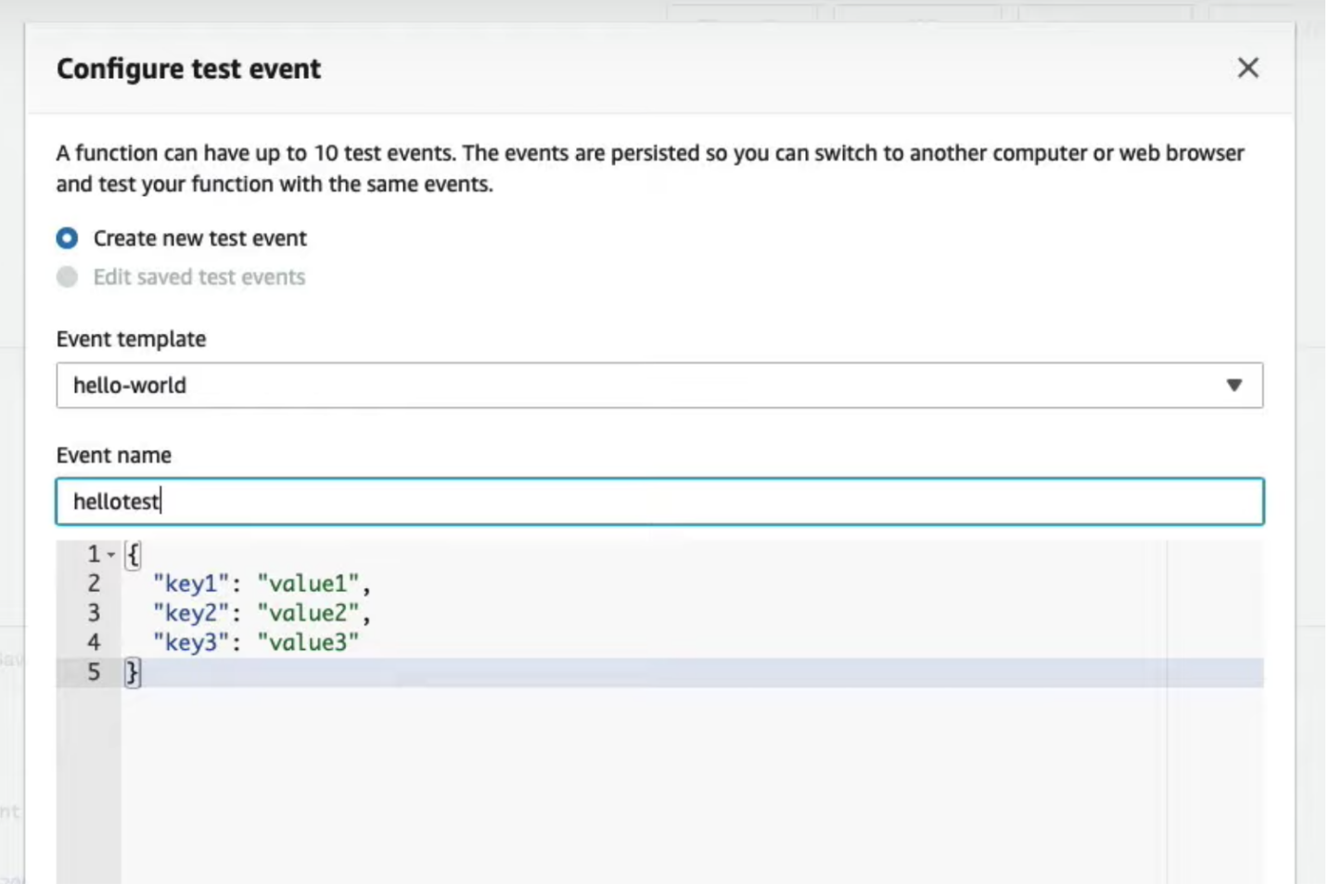
2. Click the **Create function** button and select **Author from scratch**
3. Name the function **hello**
4. Set the runtime to **Python3.8**
5. Click the **Create Function** button



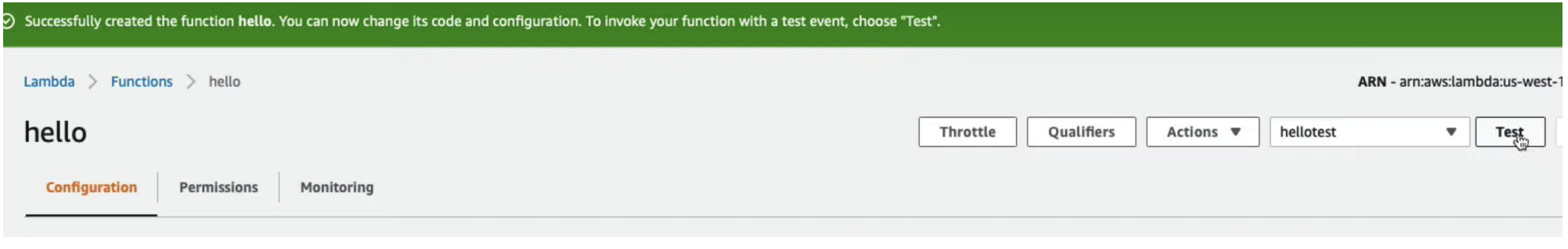
6. Notice the **handler** field on the embedded code page. It's format is **file-name.function-name**. In our case, the file name is **lambda_function.py** and the code function name is **lambda_handler**, which is why the handler is pre-set to **lambda_function.lambda_handler**.



7. Click on the **Test** button and configure a test event.
8. Name the event **hellotest** and save the event.

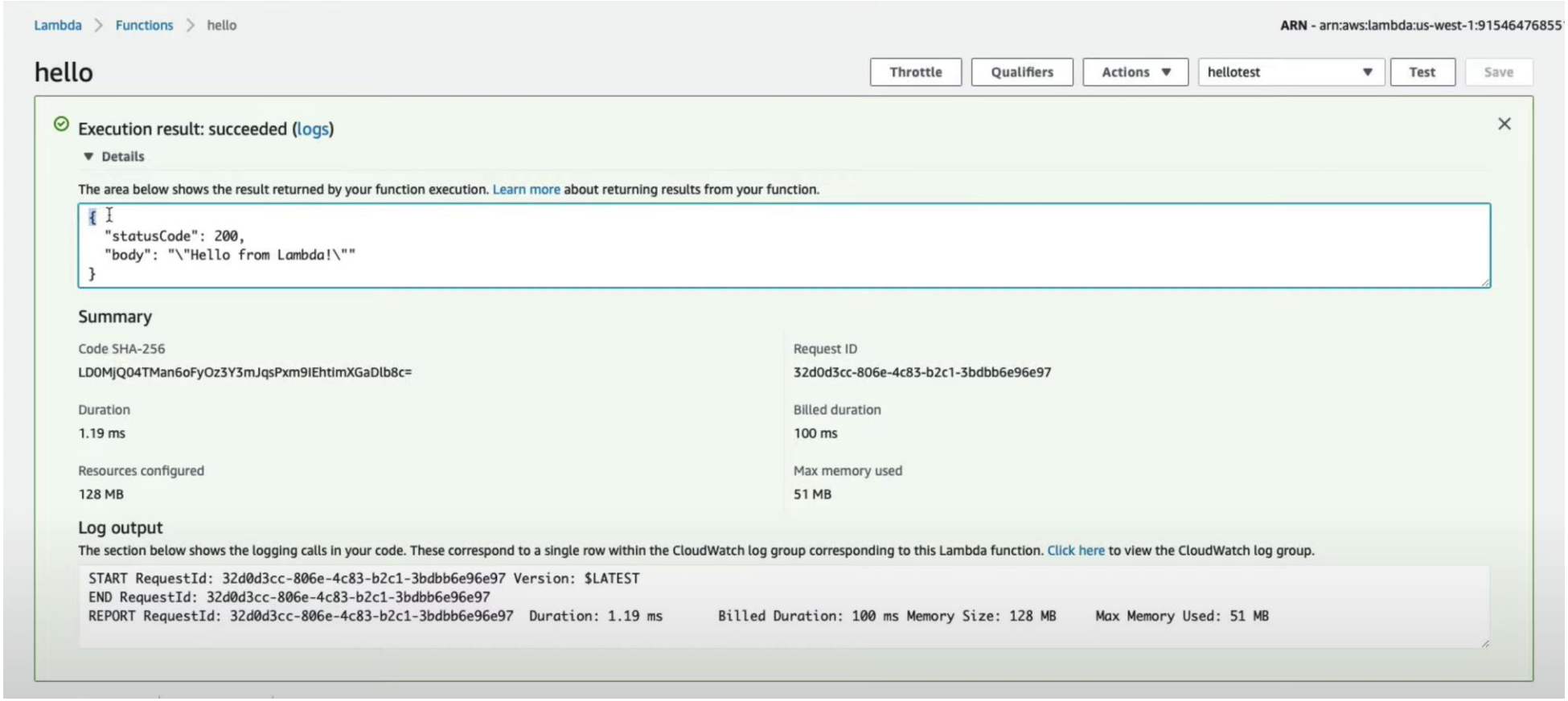


9. Click the **Test** button again while the **hellotest** event is pre-selected



10. Expand the **Execution result**

11. Verity the message: **"Hello From Lambda!"**



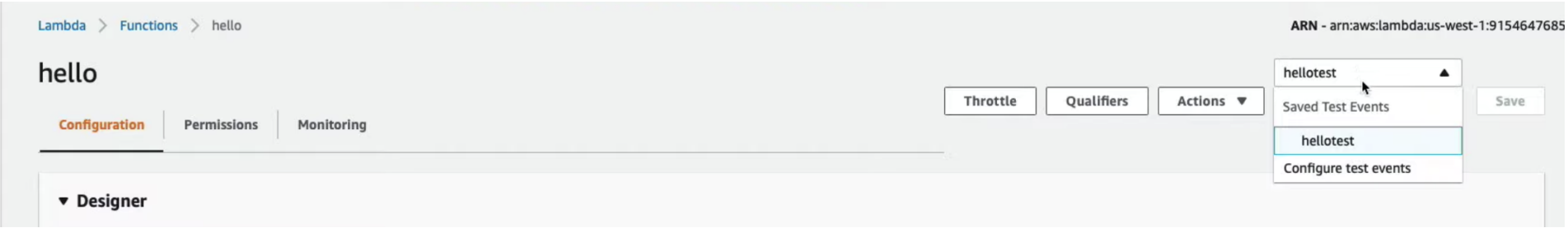
Exercise: Lambda Events

Modify the **testhello** event for the Lambda with the existing field:

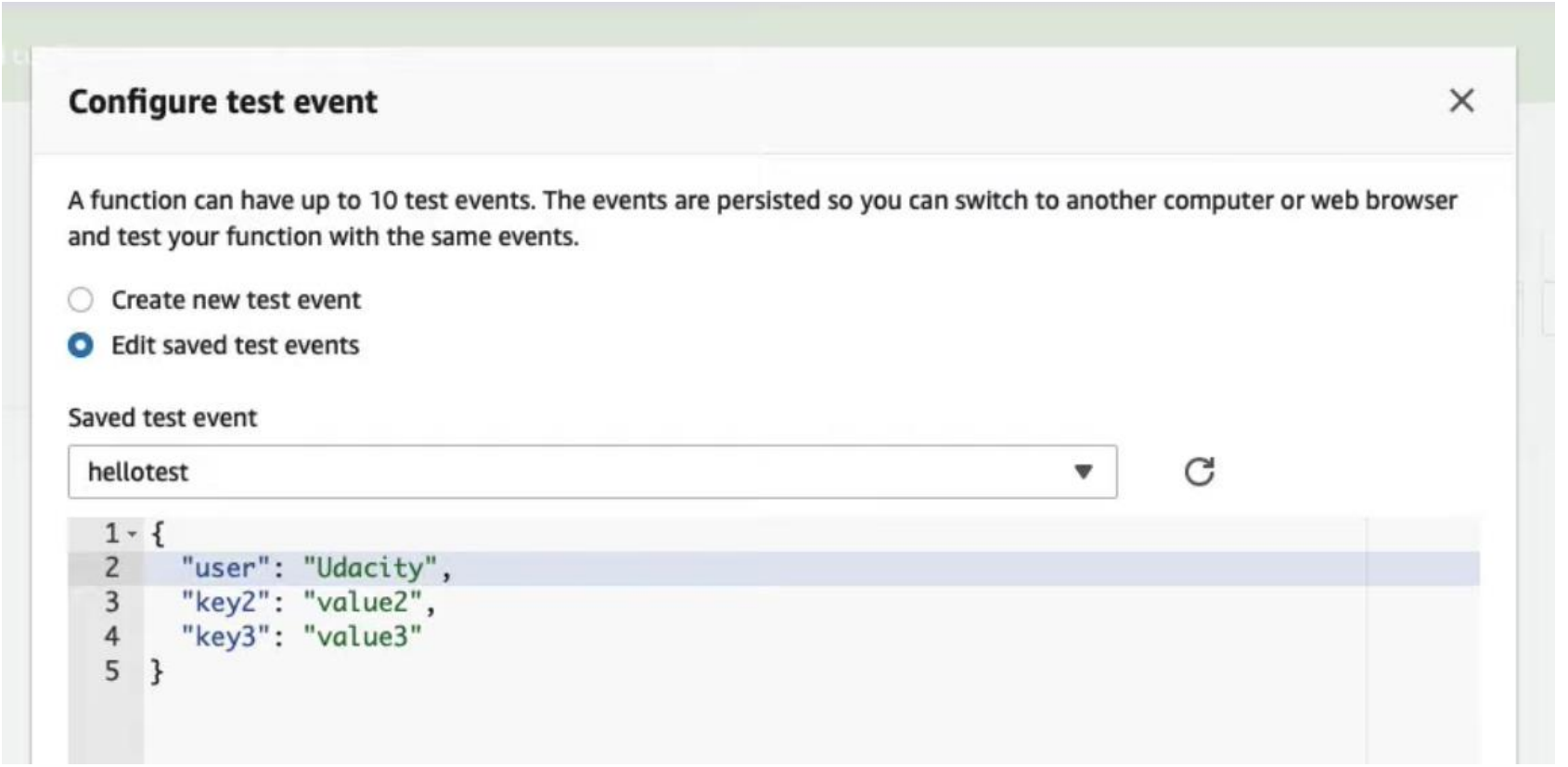
```
{"user": "Udacity"}
```

- Modify the Lambda Function code to read the user out of the event field and display "Udacity".

1. From the `hello` Lambda Function, click on the **hellotest** dropdown menu and select **Configure test events**



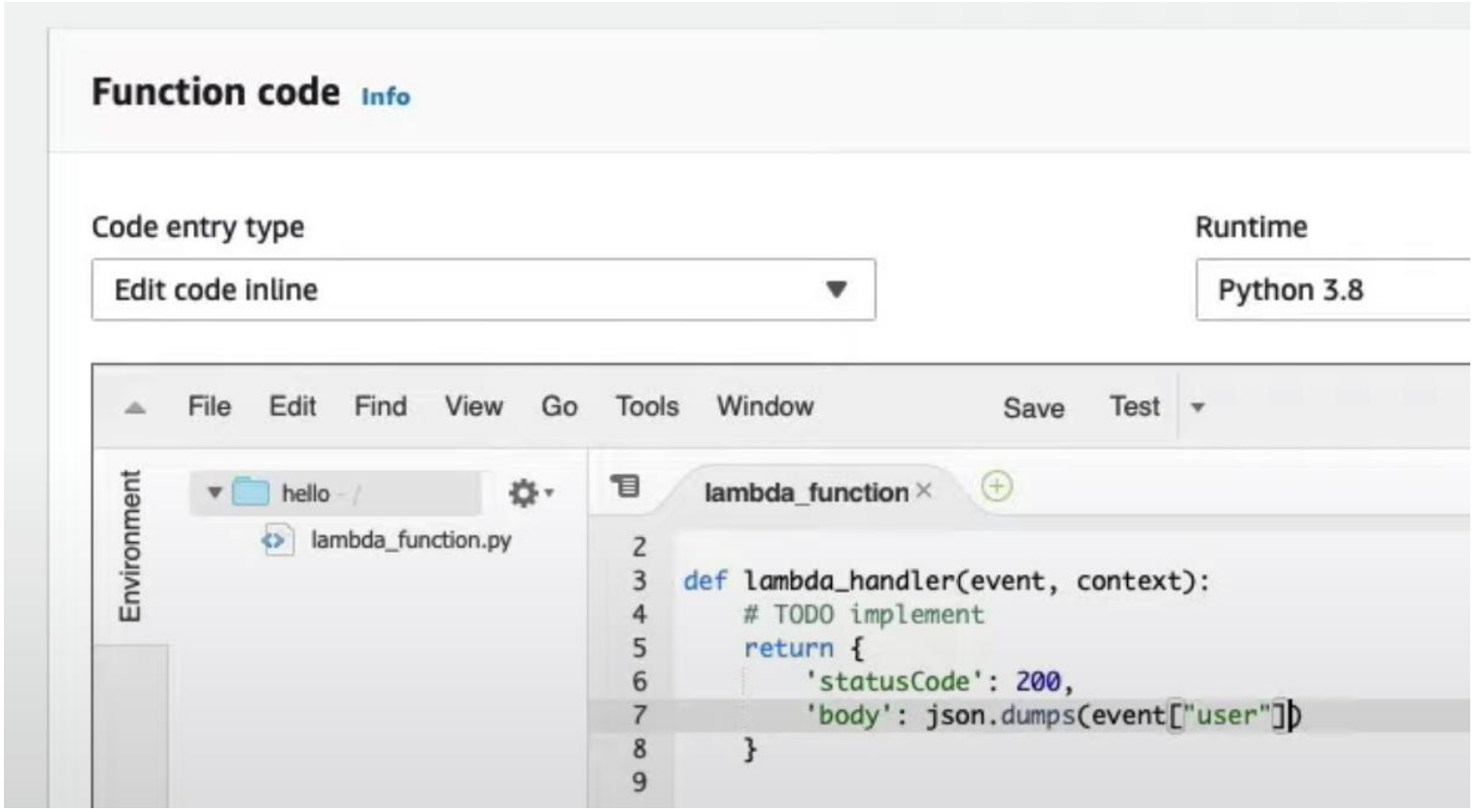
2. Select **Edit saved test events** option (it should be pre-selected already)
3. Change **key1** to be "**user**"
4. Change the value for "**user**" to be "**Udacity**"
5. Click Save Event



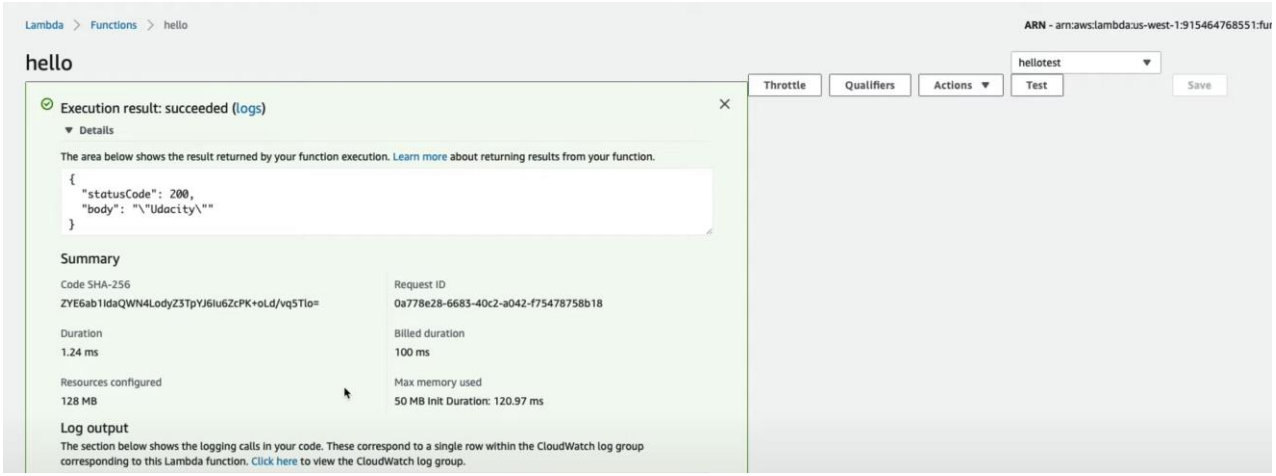
6. Edit the code in the editor and set the **body** on **line 7** to be:

```
json.dumps(event["user"])
```

7. Click **Save**.



8. Click the **Test** button.
9. Expand the results.
10. Verify message: **Udacity**



Exercise: Packaging Code For Lambda Deployment

In this exercise, you will prepare a package for a Lambda deployment. Your code will include both the Lambda Function code *and* the **Redis** dependency.

Prerequisites

To complete this exercise, you will need:

- Python3
- pip

Instructions

- Install the **Redis** dependency into the **package** folder
    - Hint: use `pip install --target`
- Zip the folder

Code to include as `main.py`

```
import redis
import os
import json

def handler(event, context):
    print("Received event: " + json.dumps(event, indent=2))
    redis_host = os.environ.get("REDIS_HOST")
    redis_port = 6379
    redis_password = ""

    r = redis.StrictRedis(
        host=redis_host,
        port=redis_port,
        password=redis_password,
        decode_responses=True
    )

    name = event.get("name")

    if event.get("body"):
        name = json.loads(event["body"]).get("name")

    if name:
        redis_successful_set = r.set("name", name)
        if redis_successful_set:
            return {
                "statusCode": 200,
                "body": "Success! {name} was written to Redis".format(name=name.capitalize())
            }
        else:
            return {
                "statusCode": 500,
                "body": "Oops! Could not write {name} to Redis".format(name=name.capitalize())
            }

    return {
        "statusCode": 200,
        "body": "Hello {name} nice to meet you".format(name=r.get("name").capitalize())
    }
```

Prerequisites
To complete this exercise solution, you will need:
- Python3 installed

Create the Lambda Package
For this section, you will need to have Python3 installed.

Lambda Functions are triggered by **events**. When our Lambda Function is triggered, our function will check if the event has a "name" field. if the even has a "name" field, our function will store the name in Redis. If the event does not have a name field, our function will fetch the name set in Redis.

1. Create a new folder on your computer called **hello**
2. Create a new file called **main.py** inside the **hello** folder
3. Copy the code into the **main.py** file (see code on previous slide).

4. Use the following command to install the code dependencies into a folder called **package**
```
pip3 install --target ./package Redis
```

5. Create a ZIP archive of the dependencies using the following command
```
~hello$ cd package
~hello/package$ zip -r9 ../function.zip .
```

6. Add your function code to the archive
```
~/hello/package$ cd ..
~/hello$ zip -g function.zip main.py
```

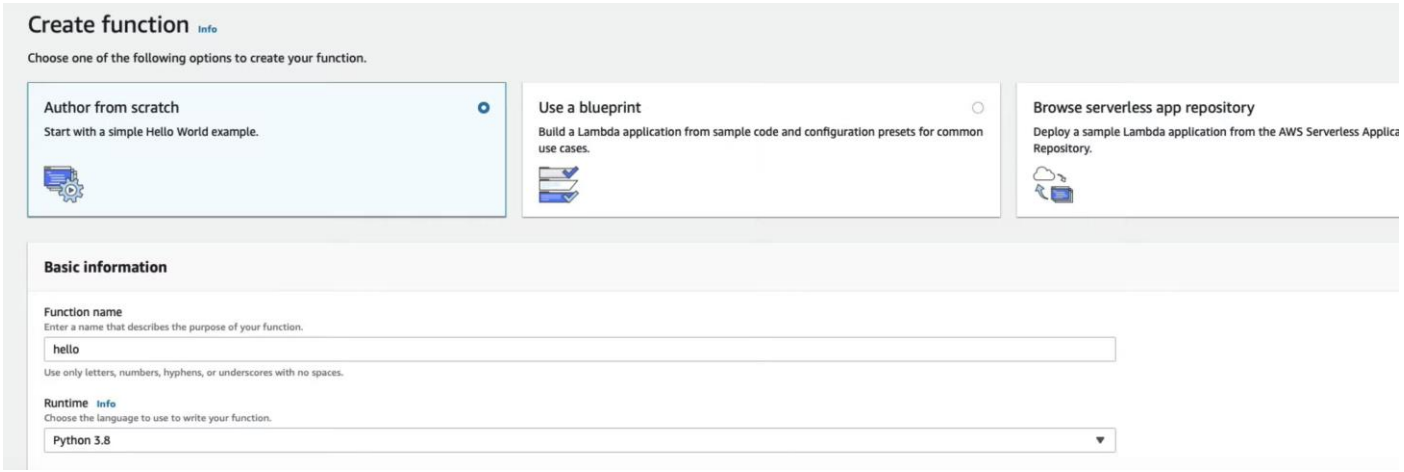
Exercise: Uploading a Lambda Function

In this exercise, you will upload a code package to Lambda.

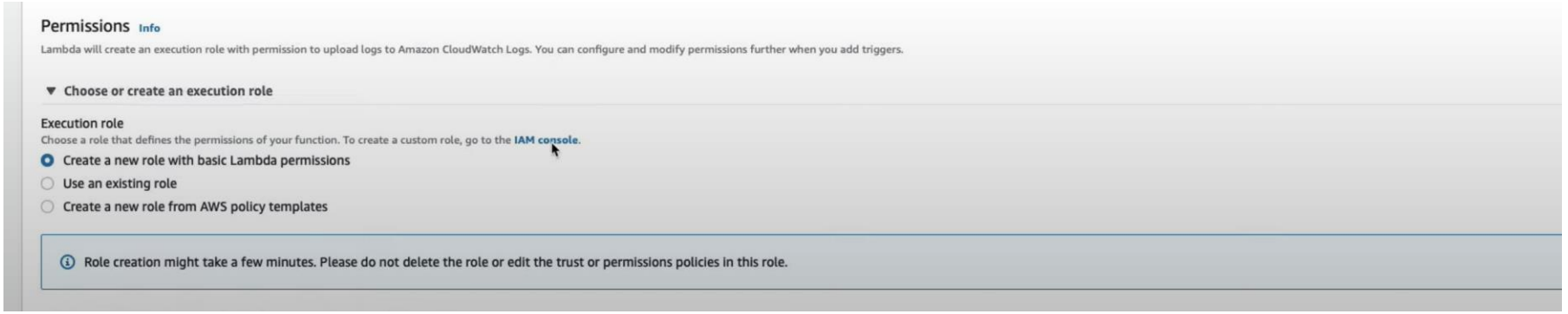
Prerequisites

If you have not already done so, complete the previous exercises to create a code package (zip file).

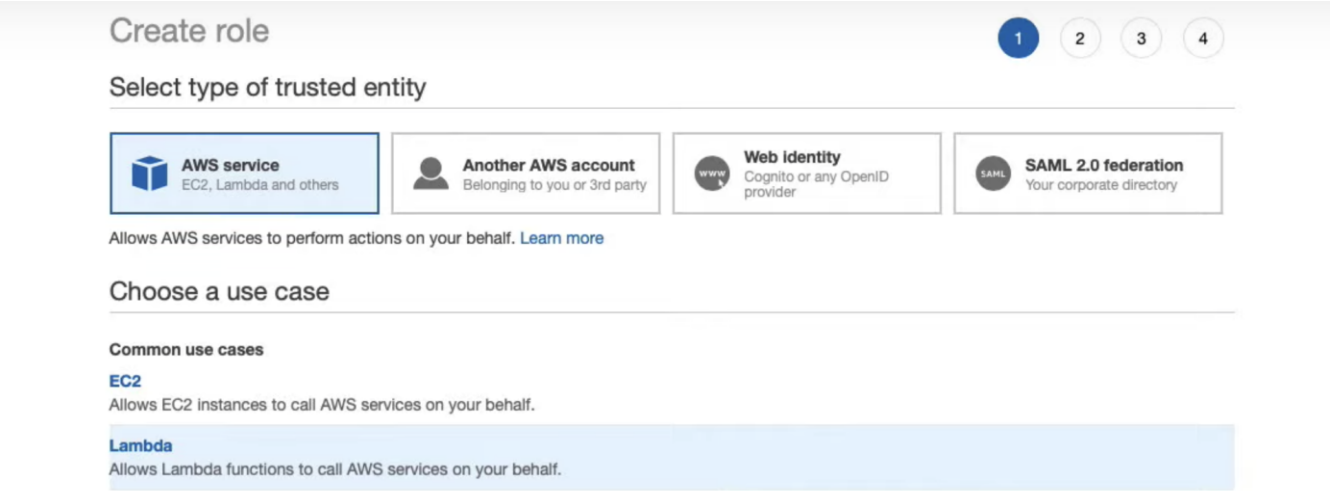
1. Navigate to the Lambda Console
2. Create a new Lambda Function
3. Use **Author from scratch**, **Python3.8**, and name the function **hello**



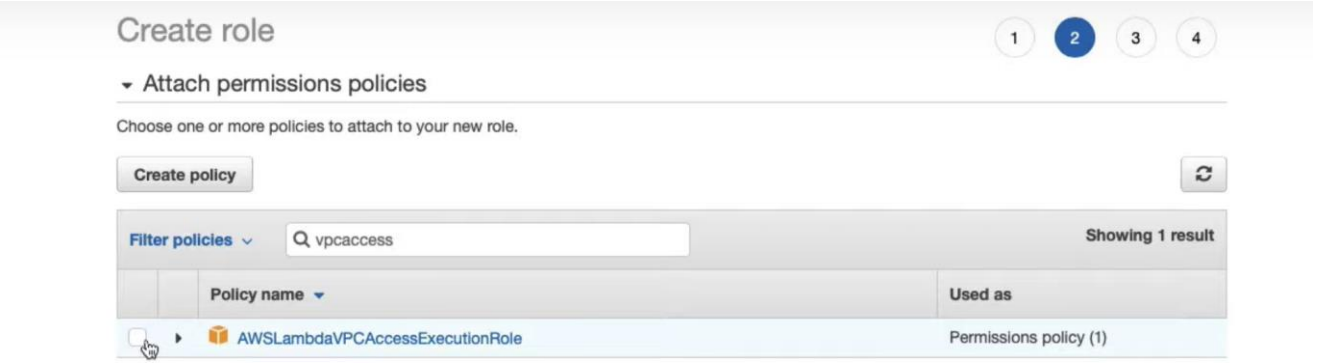
4. Expand the permissions section
5. Click the **IAM Console** link



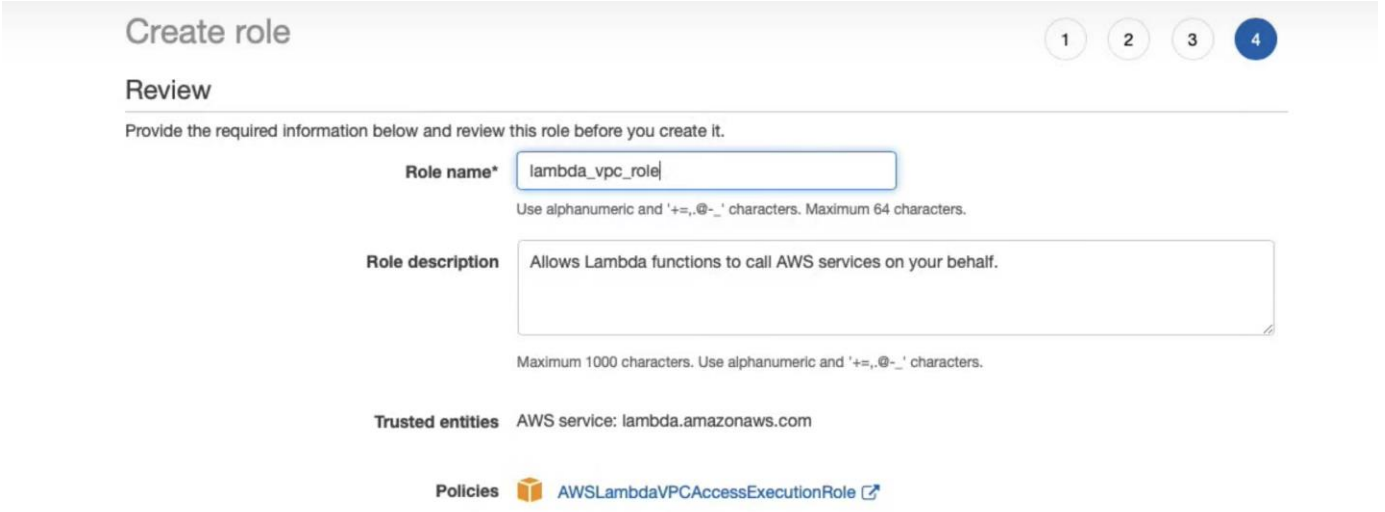
6. Click on **Roles** from the left sidebar
7. Create a new role
8. Set the type of trust identity to Lambda



9. Attach the built-in policy: **"AWSLambdaVPCAccessExecutionRole"**

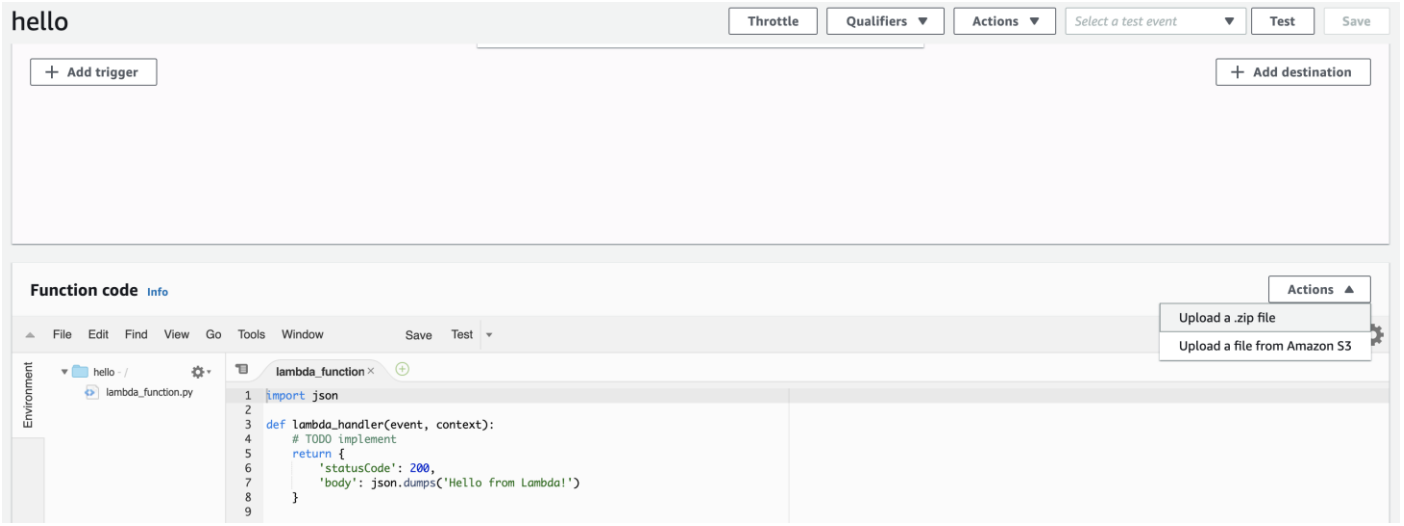


10. Name the role **lambda_vpc_role**

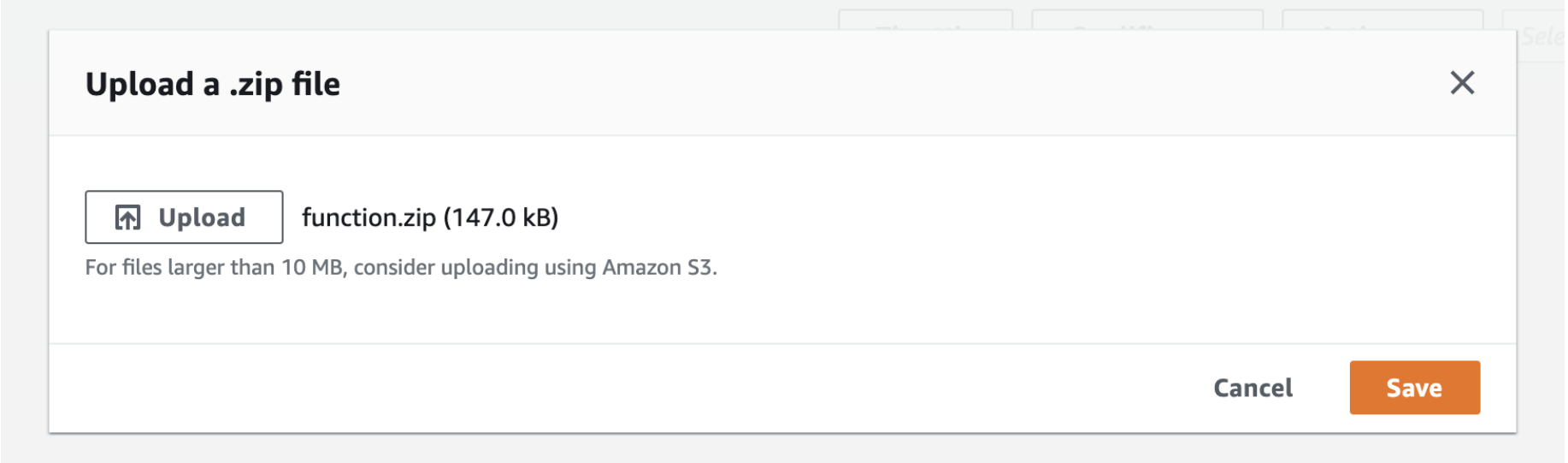


11. Click the **Actions** drop-down menu from the **Function code** section of the Lamba Function Console

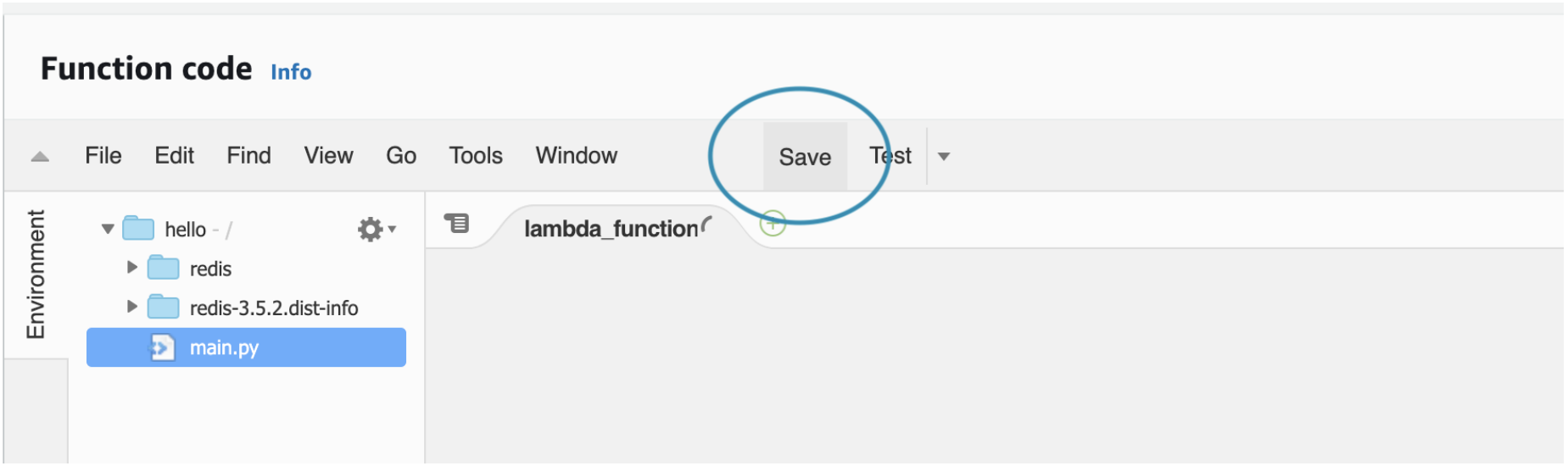
12. Select **Upload a .zip file**



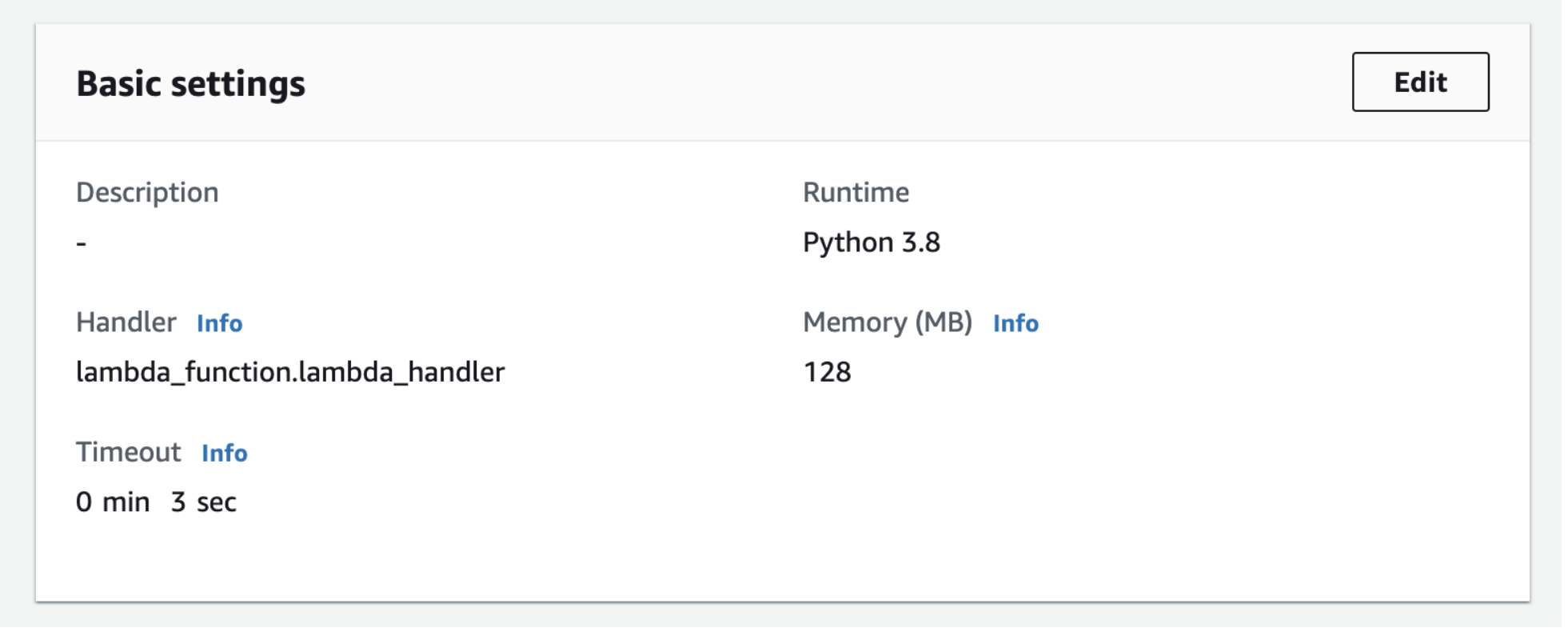
13. Select the **function.zip** package created in the previous exercise



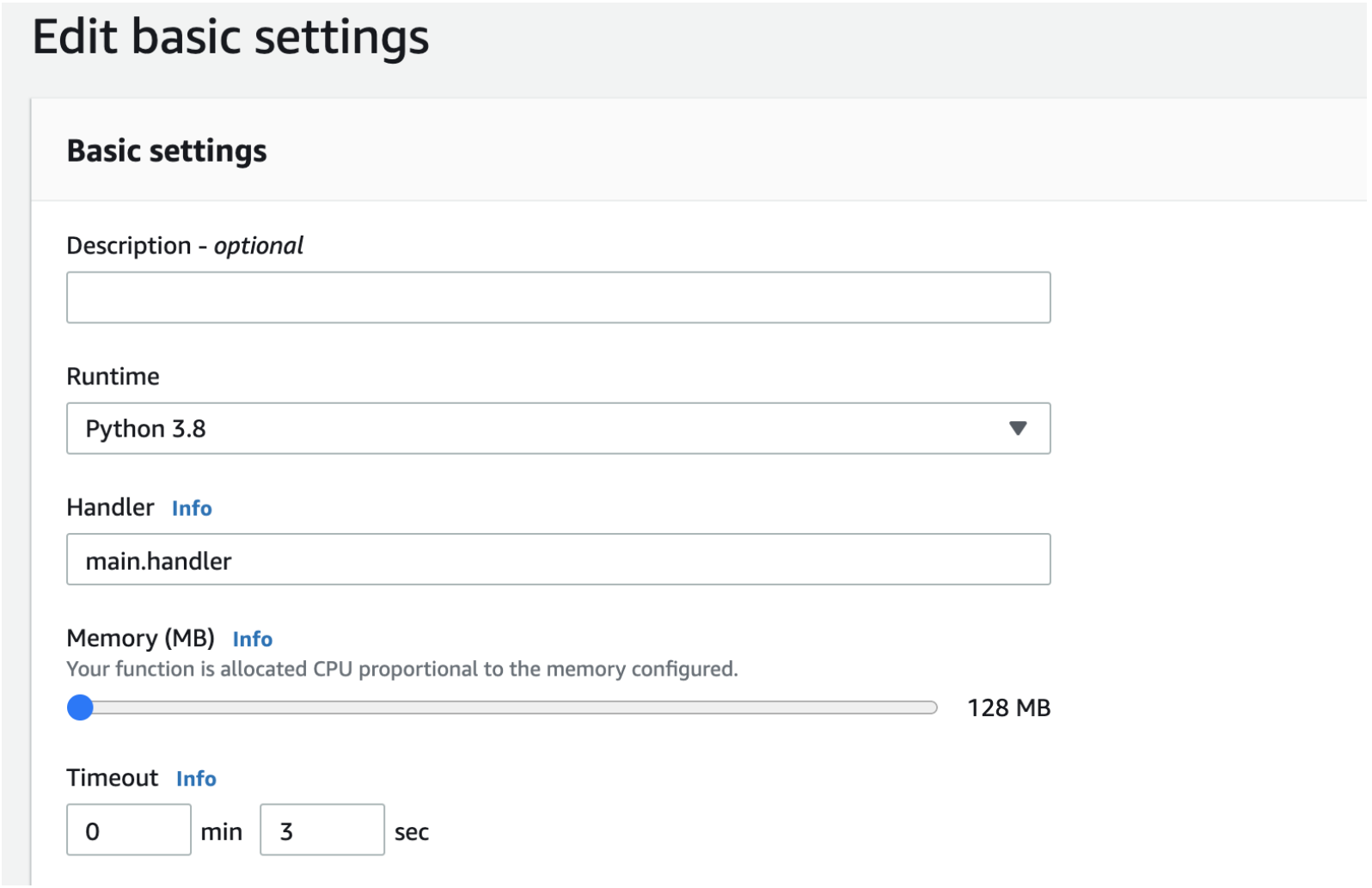
Once selected, you **MUST** click **Save**



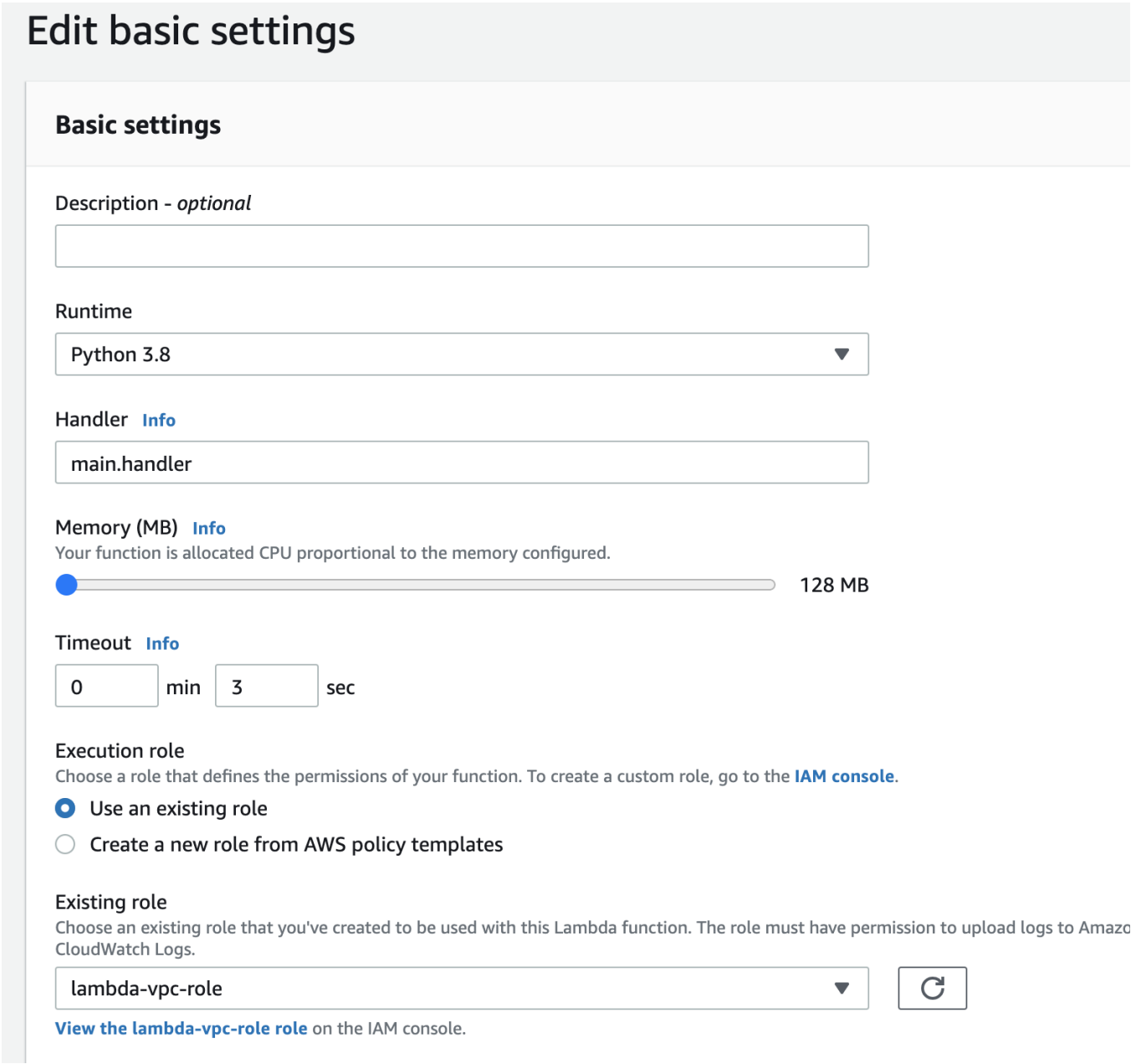
14. Scroll down to the **Basic settings** section
15. Click the **Edit** button



16. Set the **Handler** field to **main.handler**

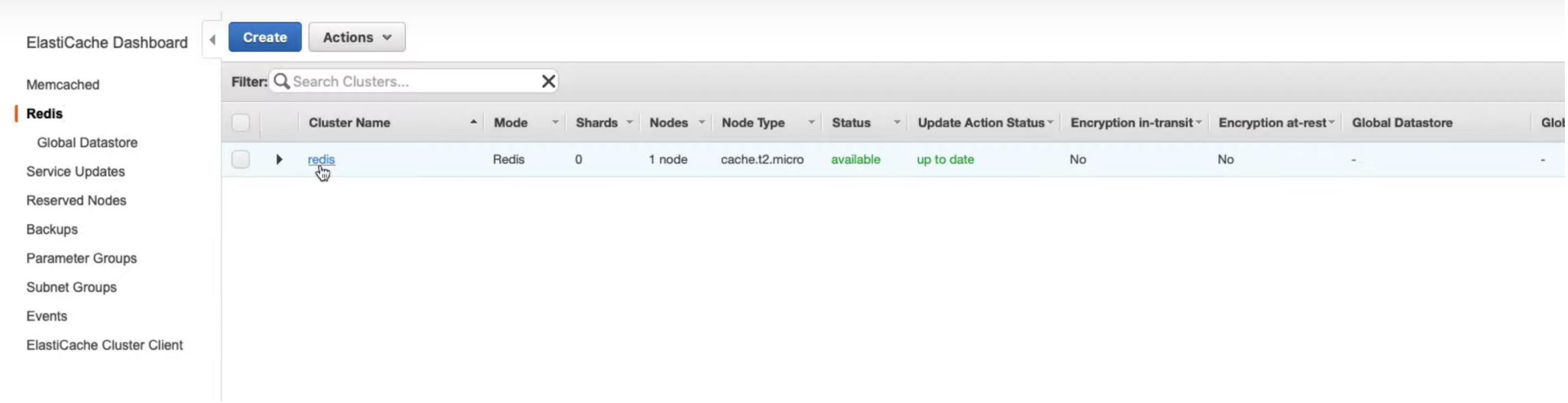


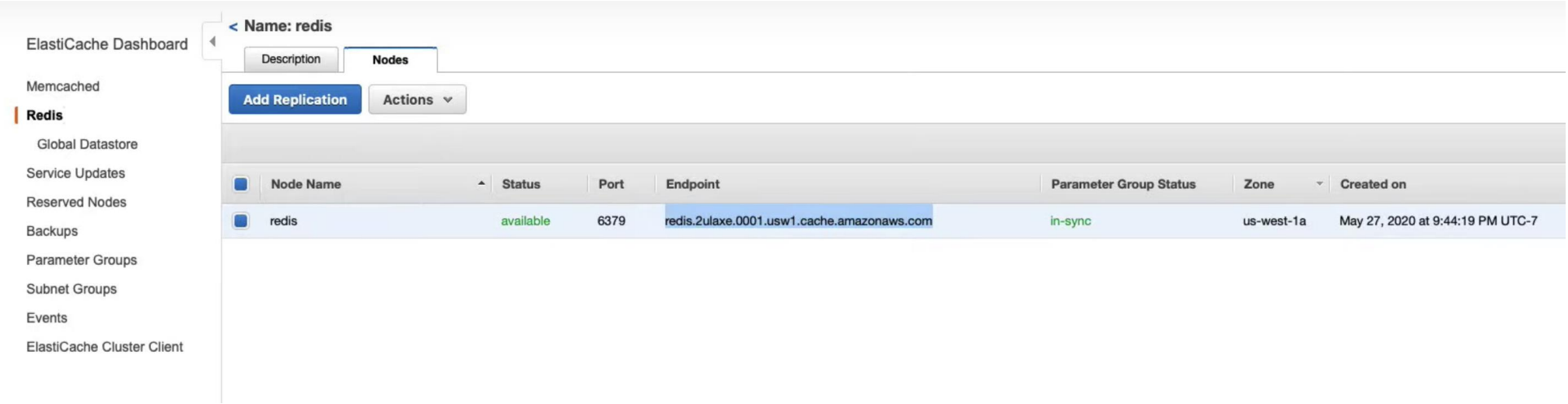
17. Select **Use an existing role**
18. Select **lambda-vpc-role** from the drop-down menu of existing roles



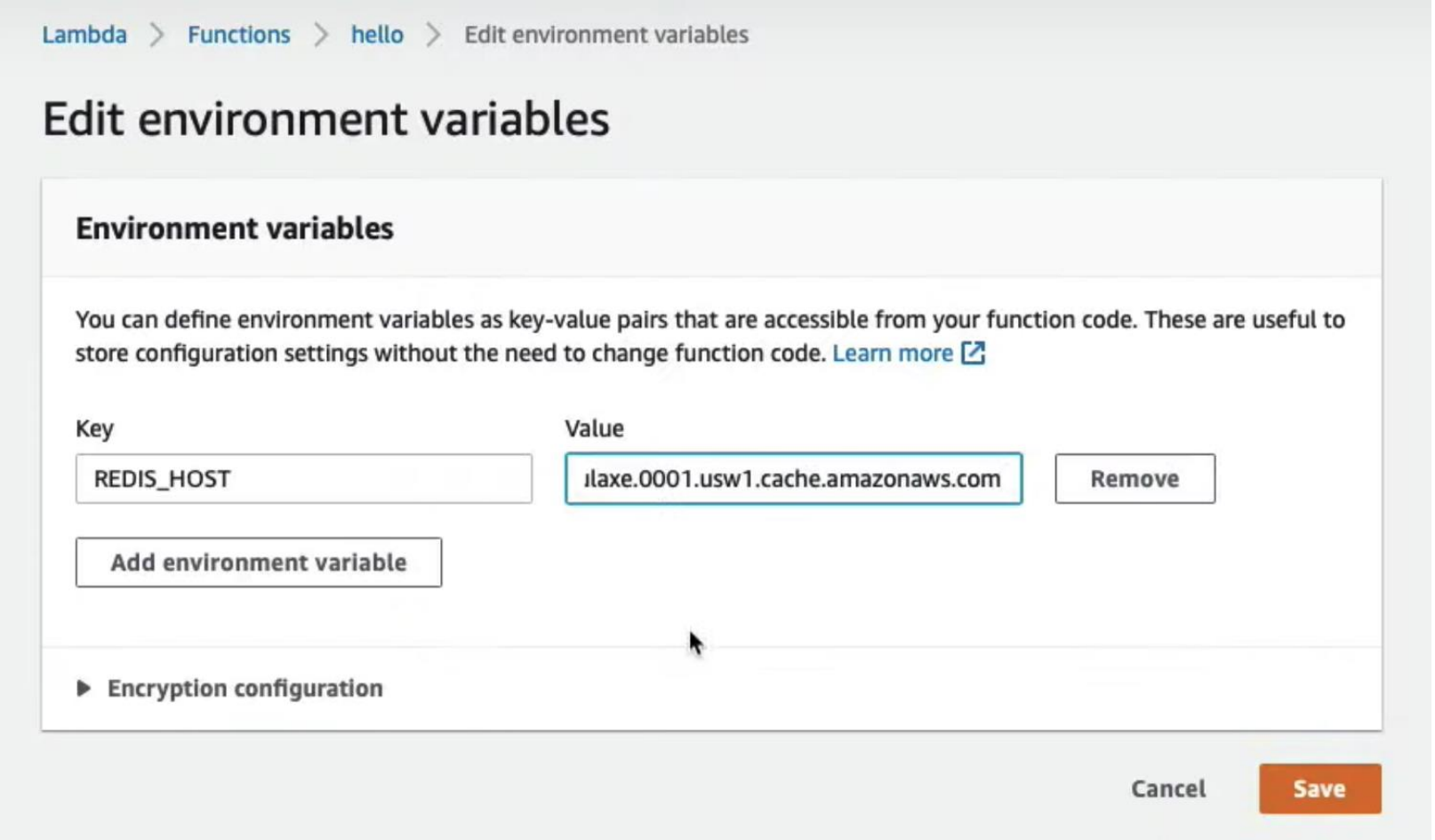
Setting the REDIS_HOST Environment Variable

1. In another tab, open the ElastiCache Console
2. Expand the "redis" node
3. Copy the node's endpoint



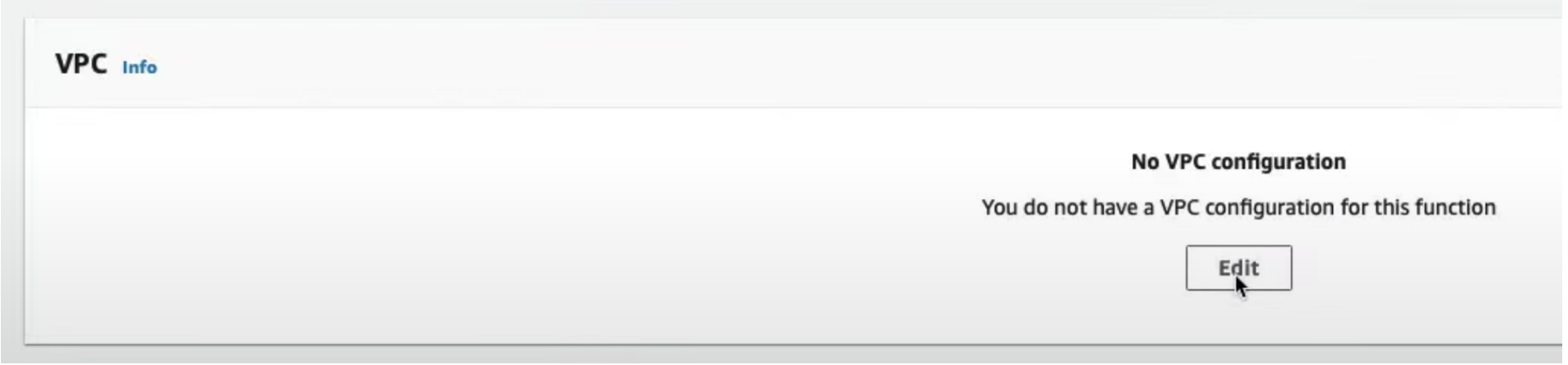


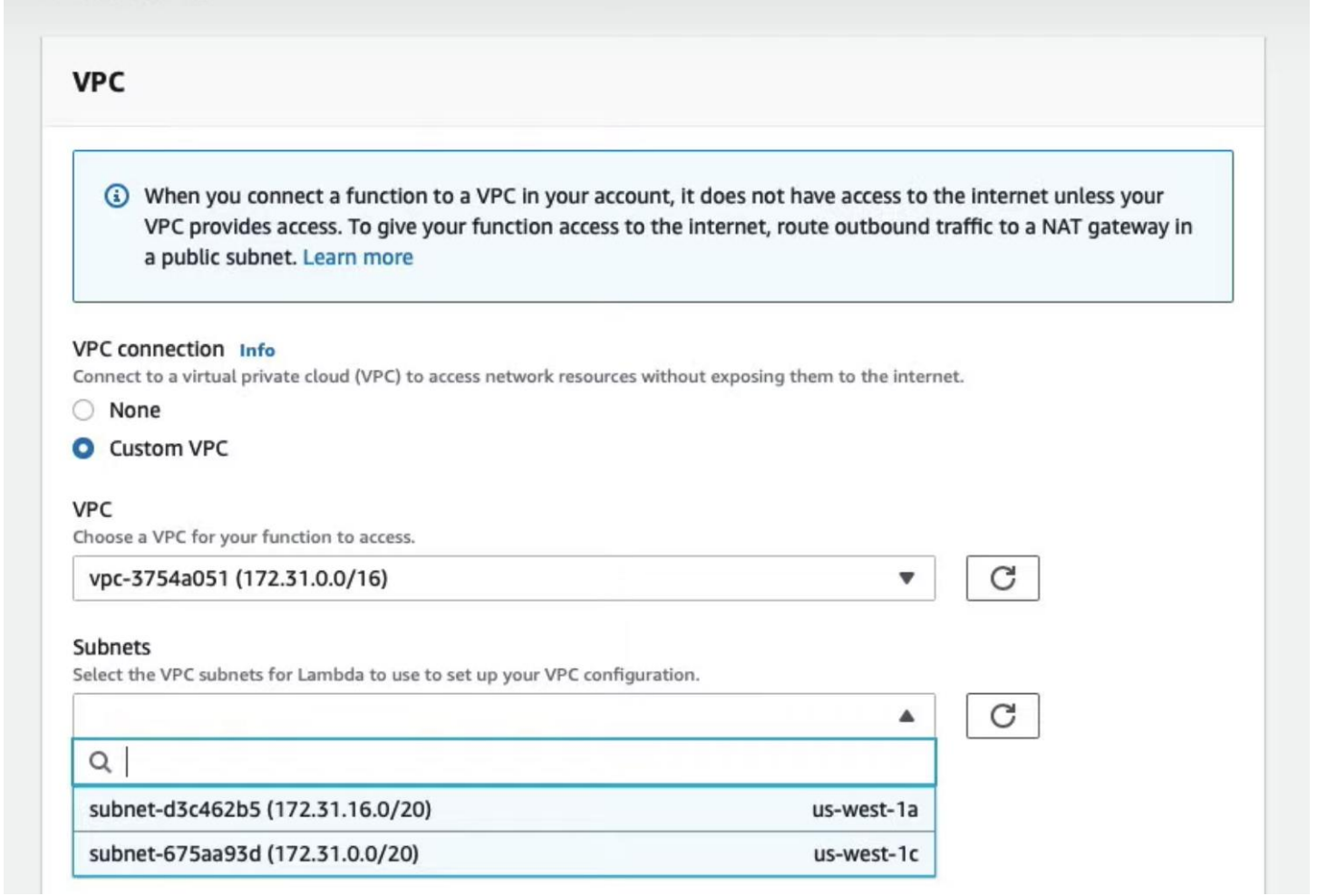
4. Click the **Edit** button in the **Environment Variables** section
5. Set the **key** to **REDIS_HOST** and paste the node endpoint as the **value**

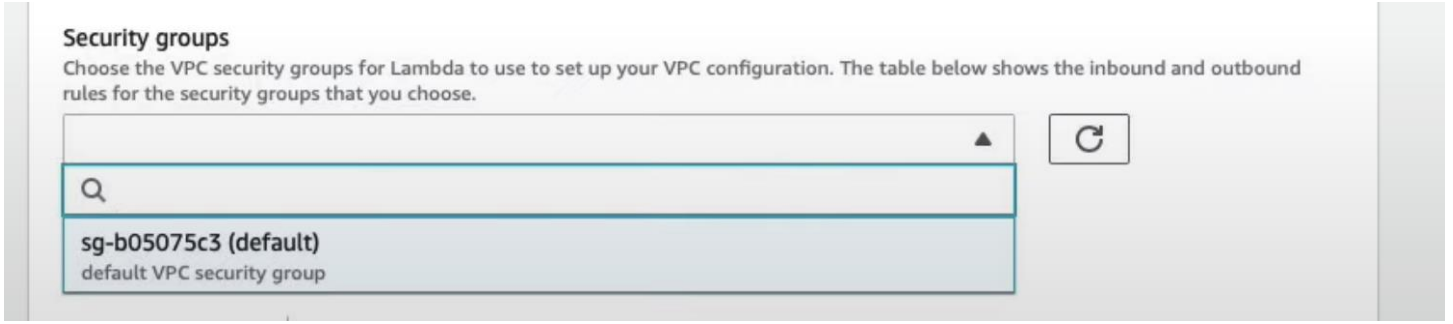


Connect The Lambda to the ElastiCache Redis Network

1. Click on the Edit button on the **VPC** section
2. Select **Custom VPC** and select the **default vpc**
3. Select all the subnets under **Subnets**
4. Under the **Security groups** select the **(default)** security group - this enables the Lambda to access the ElastiCache for Redis
5. Click **Save**




VPC
When you connect a function to a VPC in your account, it does not have access to the internet unless your VPC provides access. To give your function access to the internet, route outbound traffic to a NAT gateway in a public subnet. Learn more
VPC connection Info
Connect to a virtual private cloud (VPC) to access network resources without exposing them to the internet.
None
Custom VPC
VPC
Choose a VPC for your function to access.
vpc-3754a051 (172.31.0.0/16)
Subnets
Select the VPC subnets for Lambda to use to set up your VPC configuration.
subnet-d3c462b5 (172.31.16.0/20) us-west-1a
subnet-675aa93d (172.31.0.0/20) us-west-1c


Security groups
Choose the VPC security groups for Lambda to use to set up your VPC configuration. The table below shows the inbound and outbound rules for the security groups that you choose.
sg-b05075c3 (default)
default VPC security group


Successfully updated the function hello.

Exercise: Testing Lambda

- Configure a test event with:

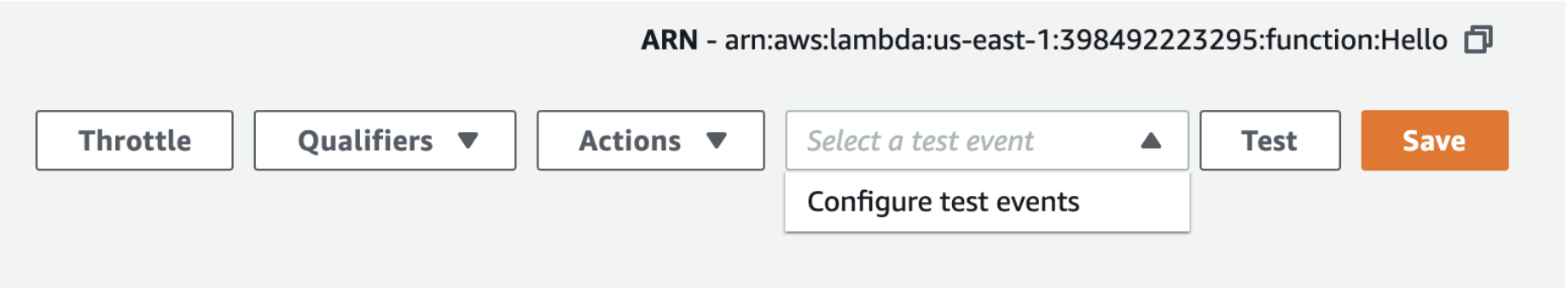
  ```
  { "name": "John Doe"}
  ```

  Test and review the log to ensure name creation is successful

- Create a blank event `{}` to test for reading the name from Redis
  Verify by reading the logs

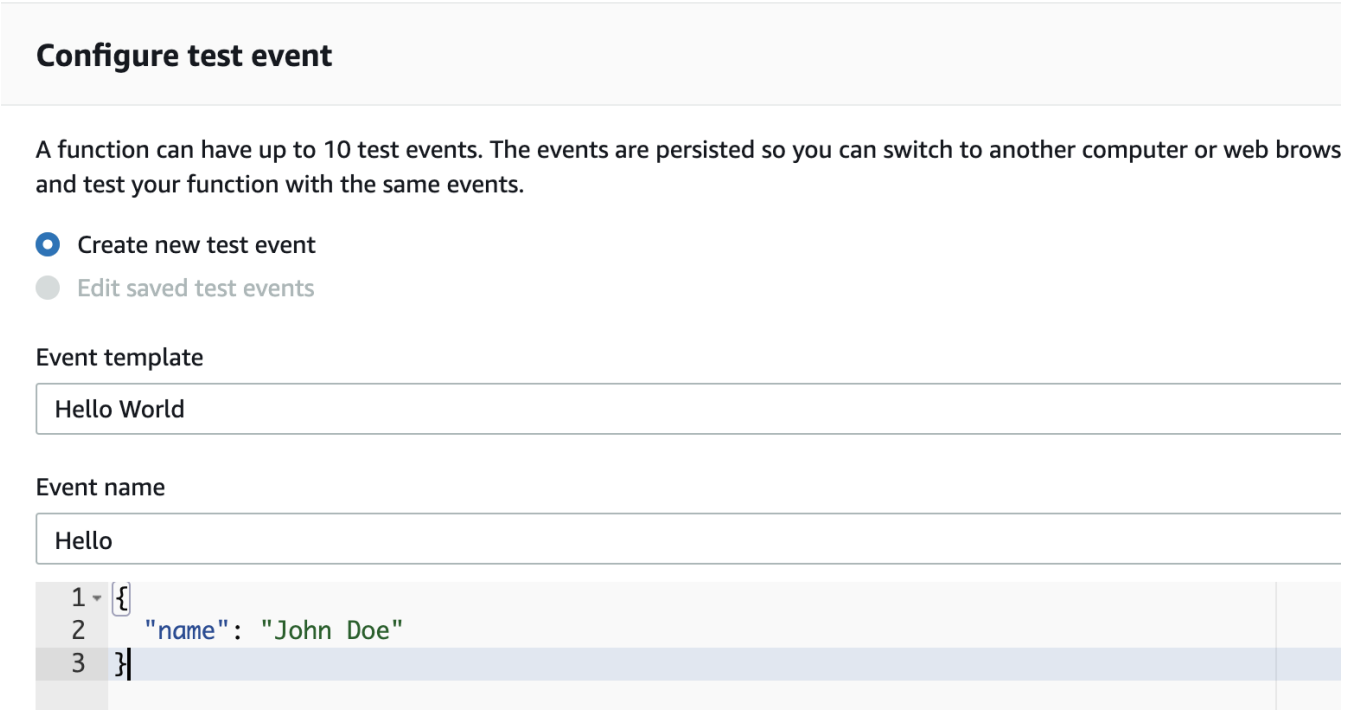
Configuring Lambda Test Event

1. Configure a new test event

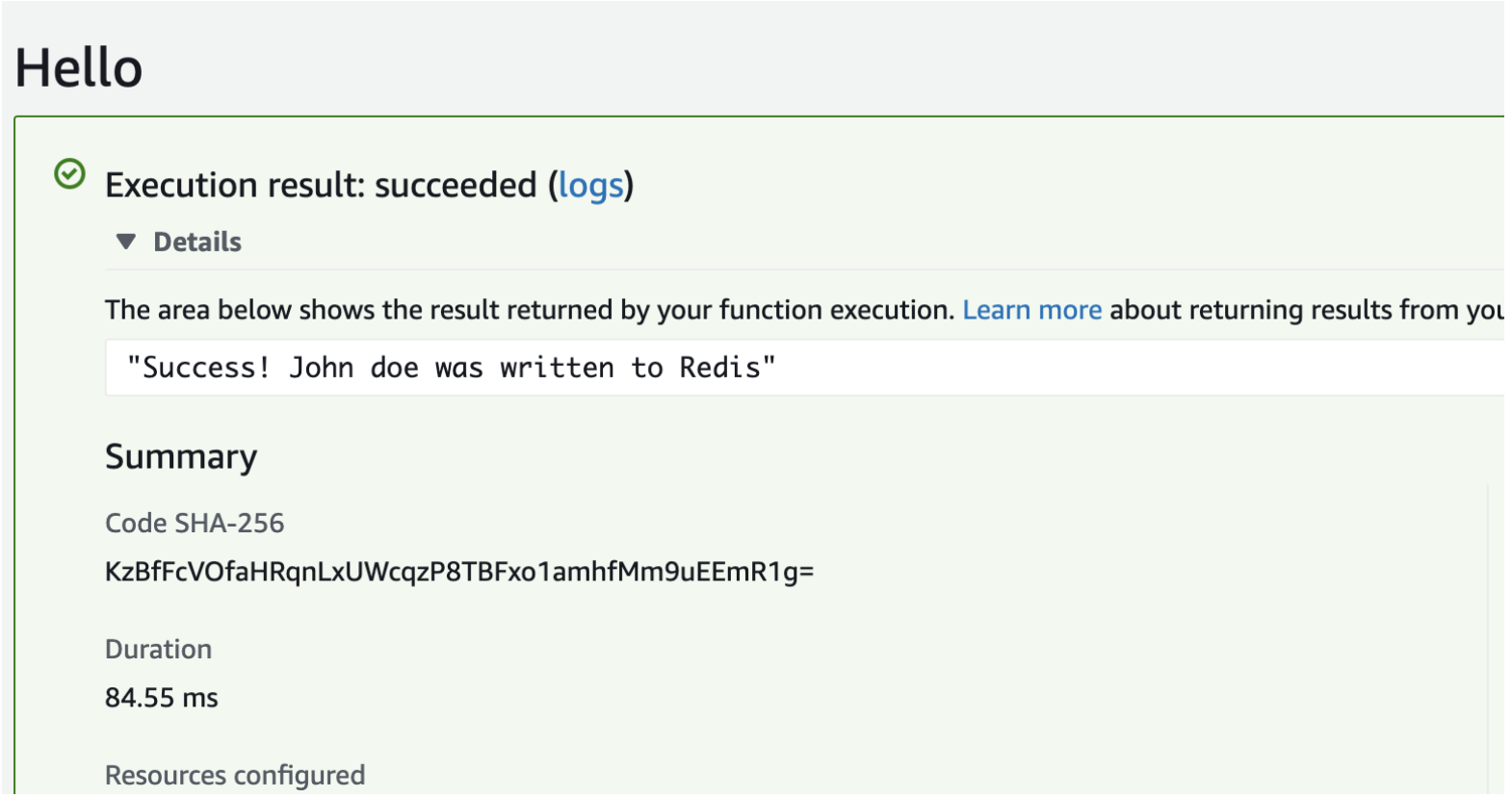


2. Set the **event name** to **Hello**
3. Set the content to be

```
{ "name": "John Doe"}
```



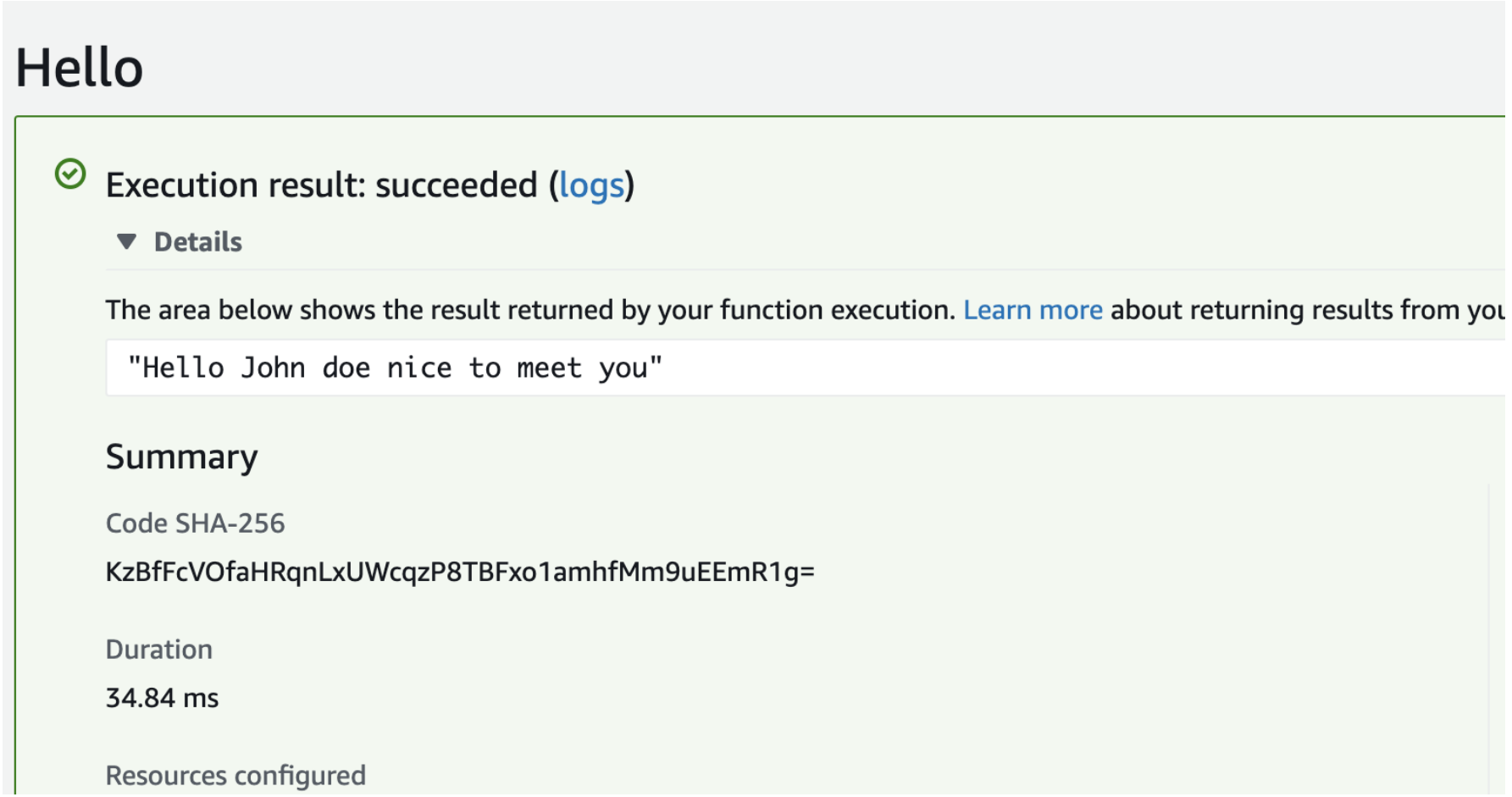
While **Hello** is selected next to the **Test** button, click on the **Test** button to invoke the lambda



Now let's invoke it with an empty event

Configure a new test event with the name **Empty** and the content **{}**

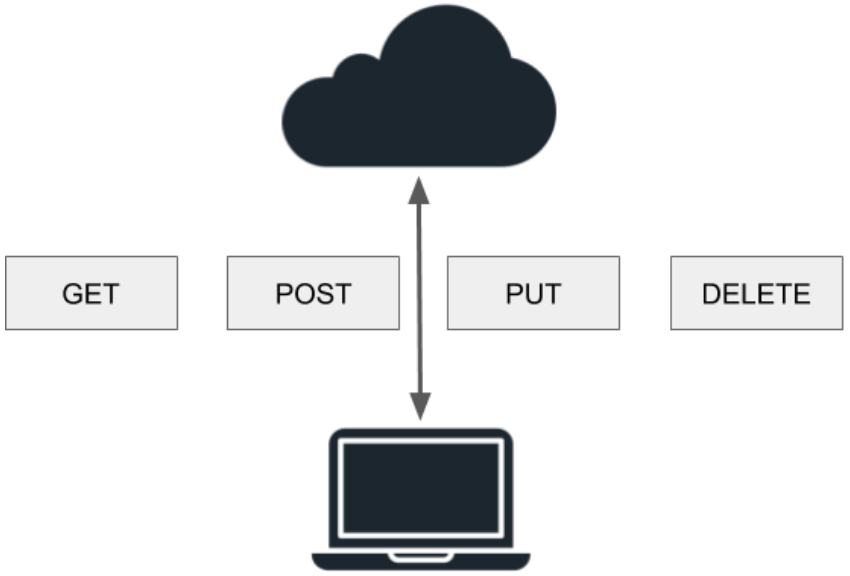
Click on **Test** again, to see the message **"Hello John doe nice to meet you"**



Feel free to test it again using a different name.
Note: do not delete the Redis instance and Lambda until the end of this lesson exercises

API Gateway

- API Gateway is a serverless HTTP app that can connect with a backend Lambda
- The simplest way to configure API Gateway is via the Lambda itself, as a trigger
- API Gateway can include API keys for additional security or it can be open to the world
- The Lambda event is wrapped inside a **body** field and sent as a JSON string that must be parsed by the Lambda Function



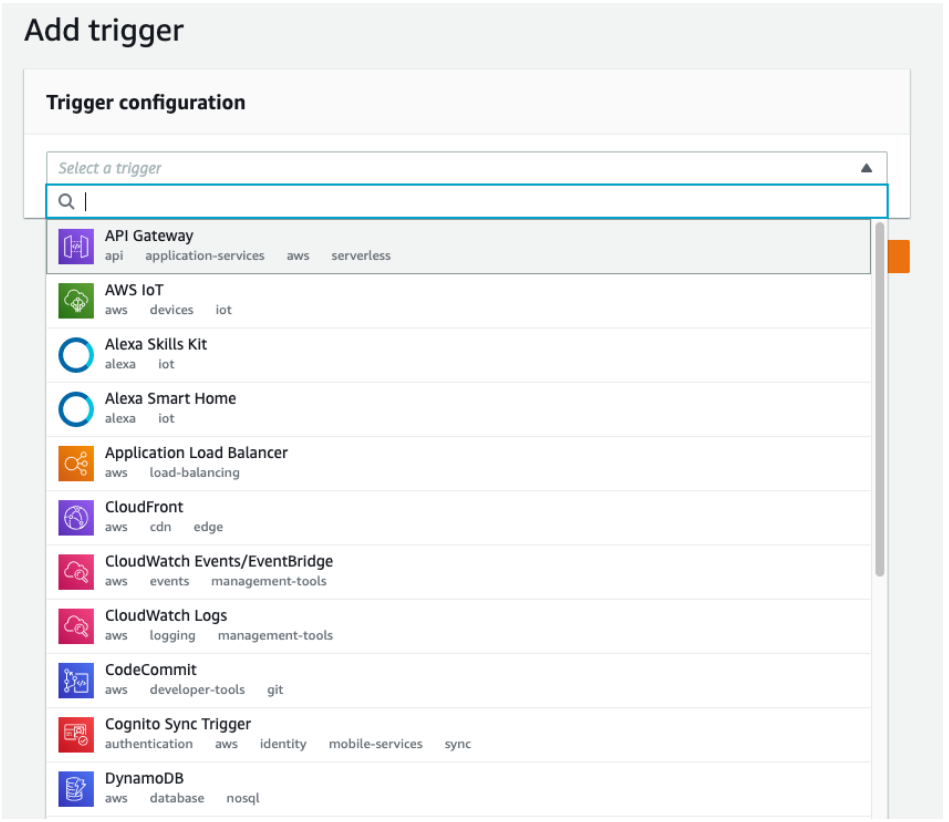
Exercise: API Gateway

In this exercise solution, you connect Lambda with API Gateway via a trigger.

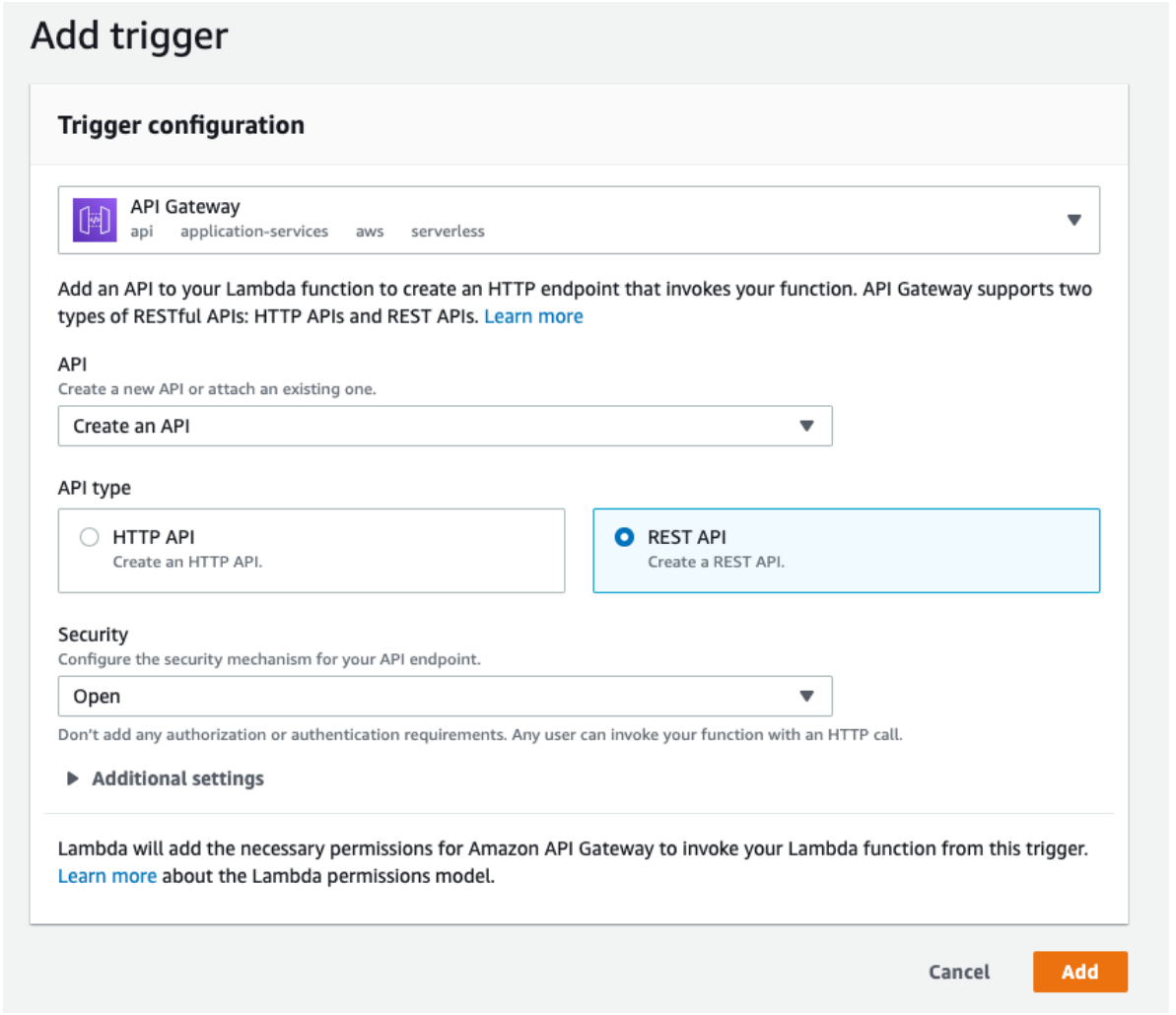
Create the API Gateway Trigger

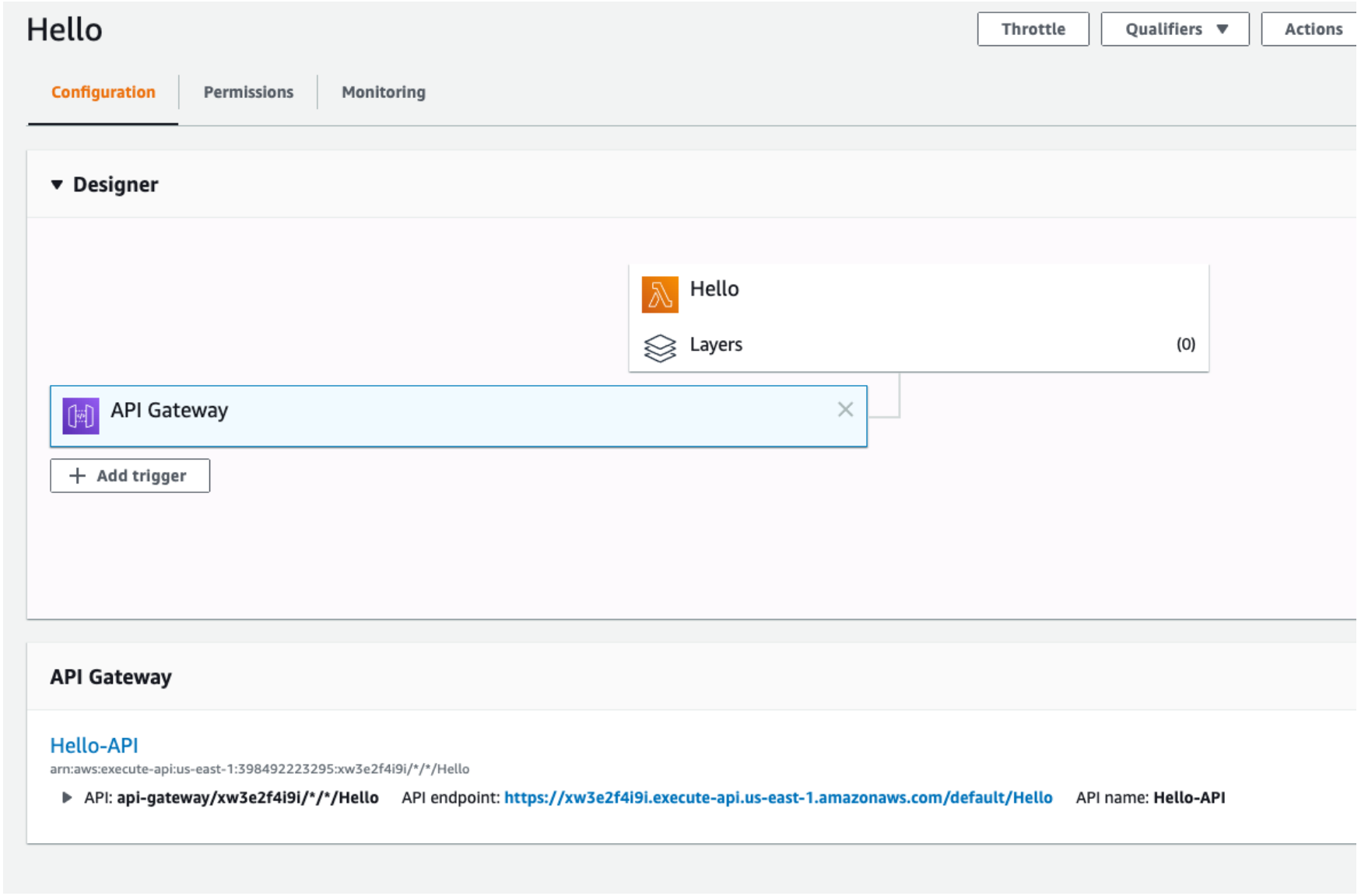
1. From the Designer section of the `hello` Lambda Function console, click the **Add trigger** button

2. Select **API Gateway** from the trigger configuration list



3. Select **Create a new API**
4. Select the **REST API** option
5. Choose **Open** for the **Security** field
6. Click the **Add** button





if we click on the **API endpoint** URL, we should see the "Hello" message from our `hello` Lambda Function.

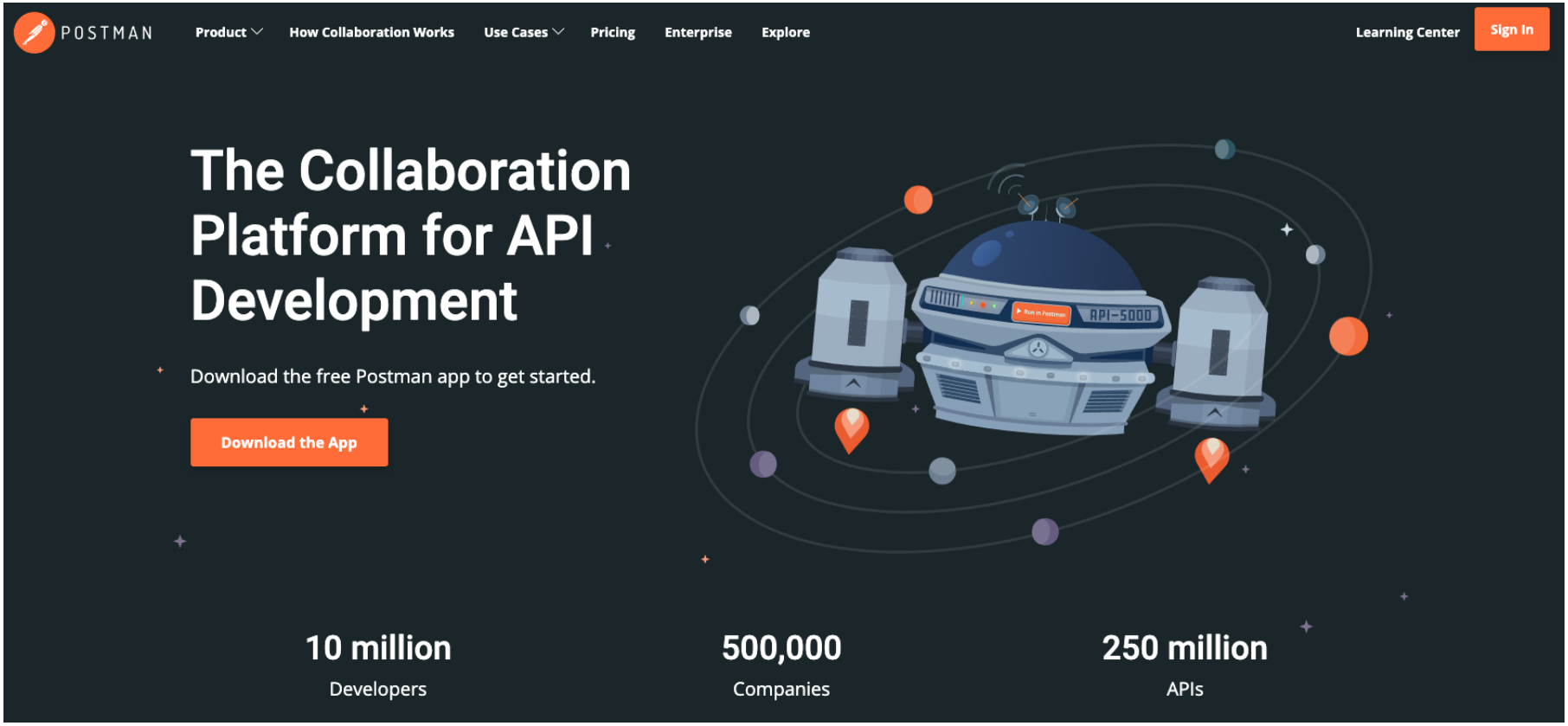
Exercise: Postman

Instructions

- Download PostMan
- Run the downloaded Postman application
- Send a **POST** request to the API Gateway URL for the `hello` Lambda Function
  - Update the request body to store a name in **ElasticCache Redis**
- Test the API Gateway URL with a **GET** request to verify that the correct name is returned

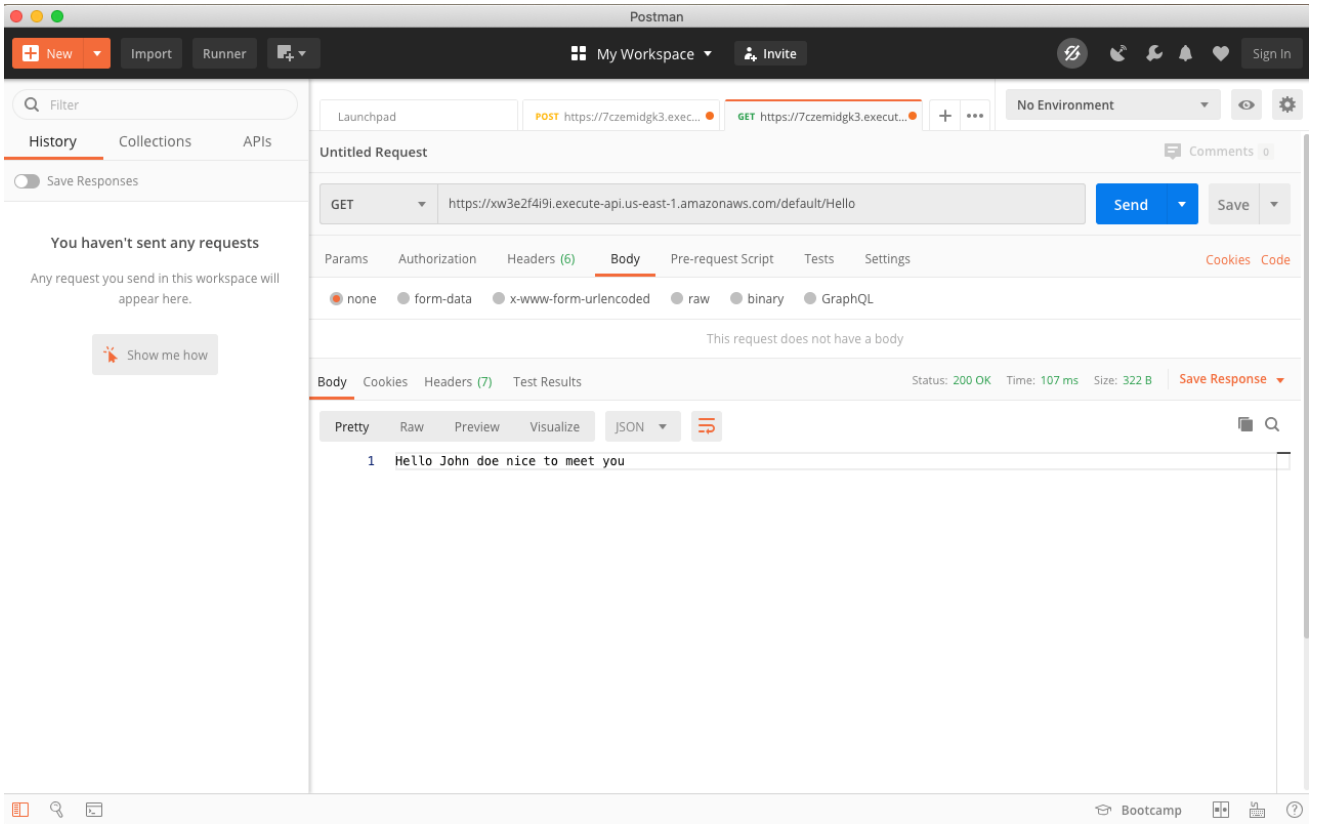
RESTful (REpresentational State Transfer) **APIs** (Application Programming Interface) support actions such as create, read, update, and delete (CRUD). Often these actions are implemented by HTTP methods (e.g. POST, GET, PUT, DELETE).

These HTTP methods are supported by web browsers, but web browsers are designed primarily to send GET requests to fetch webpage. Postman makes it easy to send any type of request, with any type of data.



Get A New Name

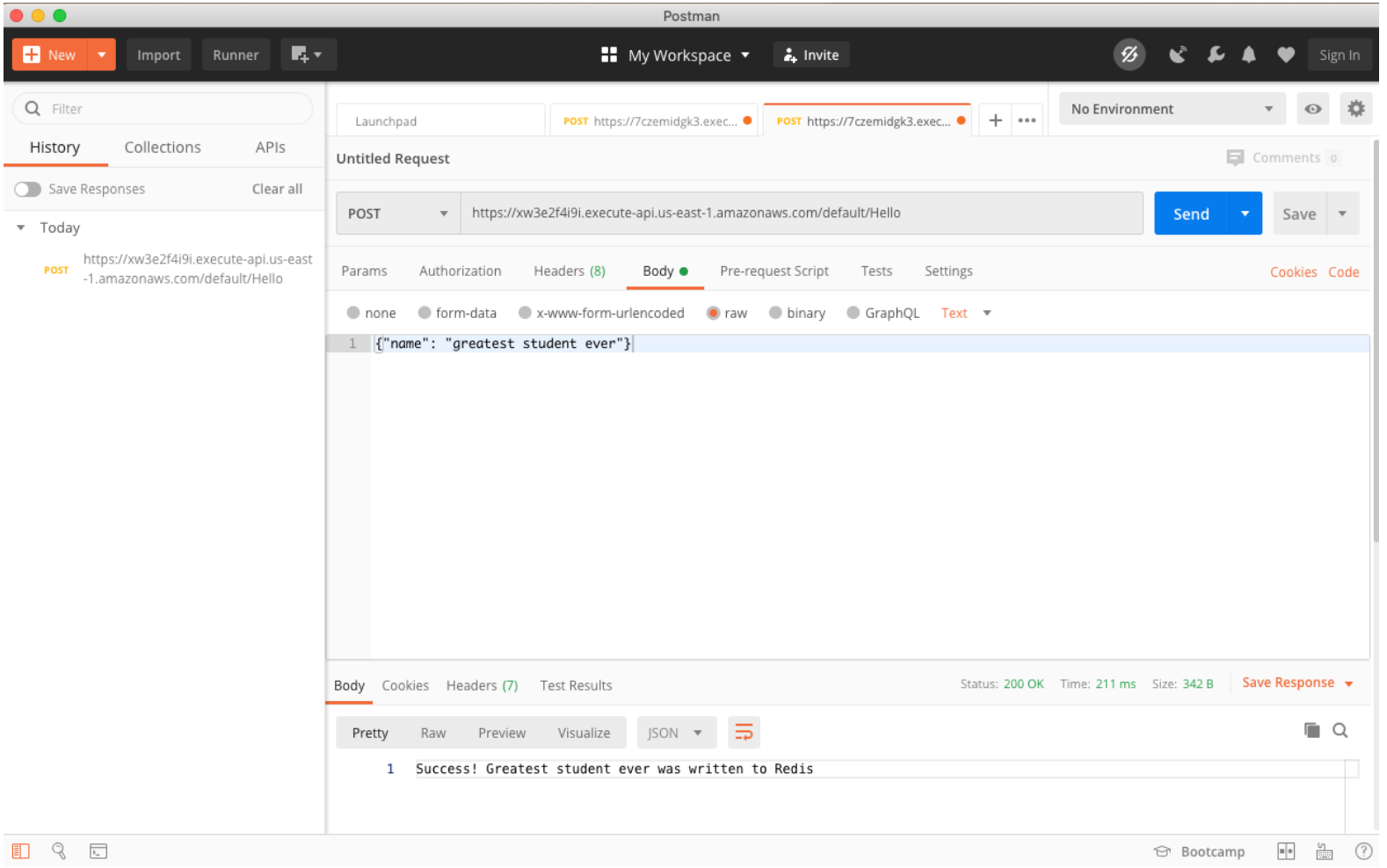
1. Download and run Postman
2. Click **Skip Sign In** on the bottom
3. Click **+ New** button on the top left corner
4. Select **Request** to create a basic request
5. Click **Cancel** to skip the **New Collection** window
6. Get the **Invocation URL** from the API Gateway created in the previous exercise
7. Paste this URL into the **Enter request URL** field
8. Click **Send** to see the "hello" message from the Lambda



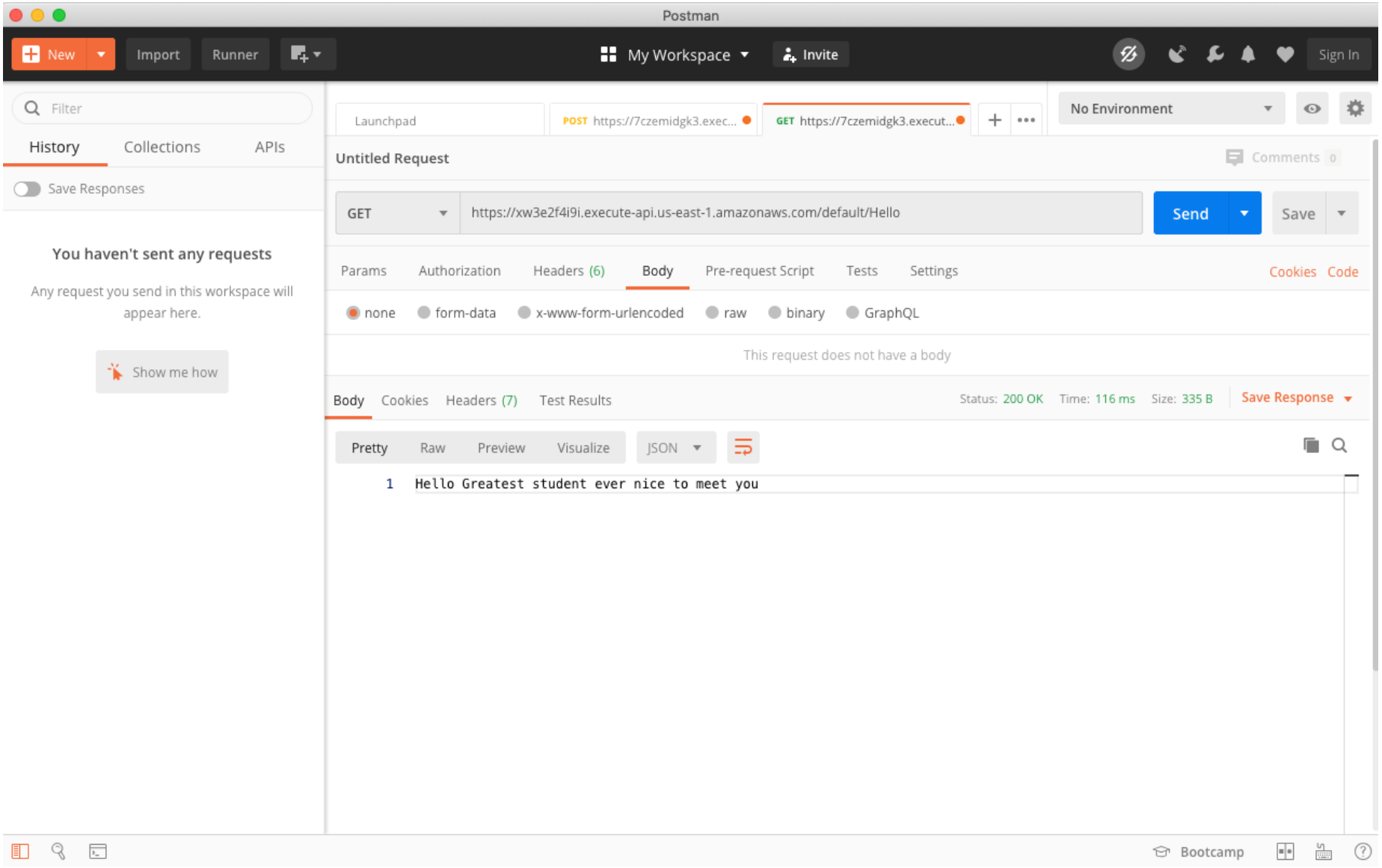
Set a Name

1. Change the method from **GET** to **POST**
2. Click on **Body** in the row below the URL
3. Select **Raw**
4. Set the content to:

```
{"name": "greatest student ever"}
```



Verify the **POST** request succeeded by sending another **GET** request (clear the Body first)



Clean Up

Make sure you delete **all of the AWS resources used in this course**.

- From the ElastiCache console select the Redis instance
  - From the **Actions** dropdown menu, select **Delete**
- From the `hello` Lambda Function console, click on **Delete** (next to the API trigger) to delete the API Gateway
- From the **Actions** dropdown menu, select **Delete** to delete the Lambda Function itself

Further Reading

Tutorials

- Build a Serverless Web Applciation

Documentation

- AWS Serverless Documentation
- Serverless Applications Lens – AWS Well-Architected Framework

Books

- AWS Serverless Computing For Beginners

Blog Posts

- What is serverless
- AWS Serverless bold posts