

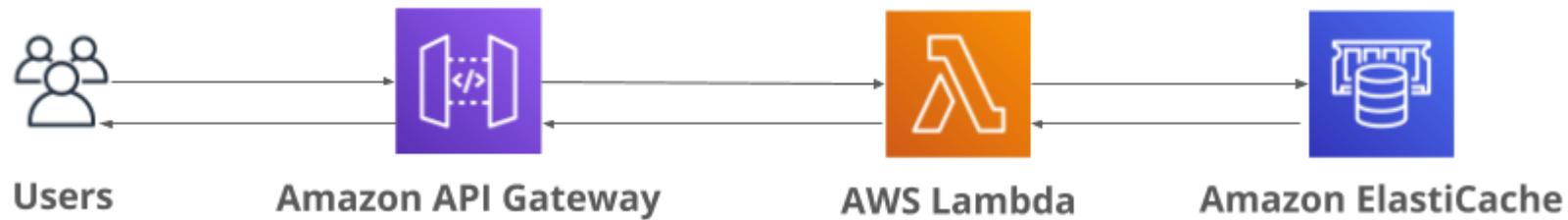
# Serverless concept

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.

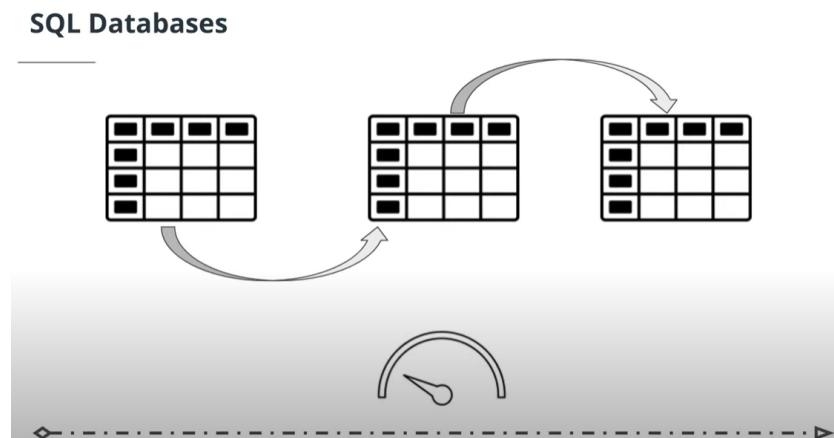
That Lambda will either set or get the user's name from a managed, in-memory database service. The Lambda will return the name back to the user via the API Gateway.



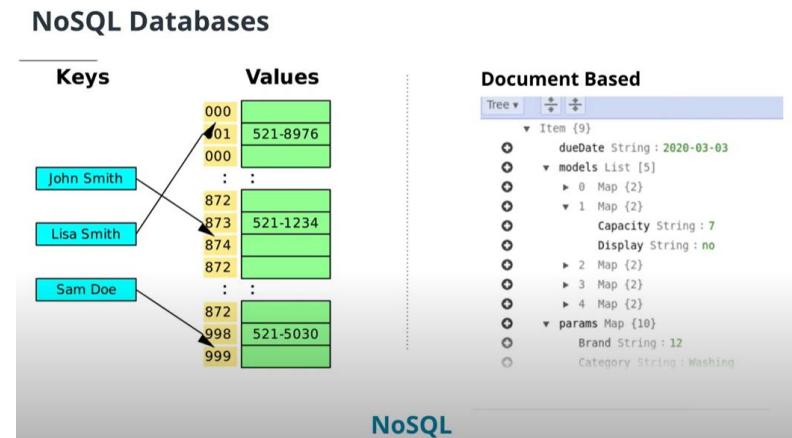
# ElastiCache

## 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 that 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

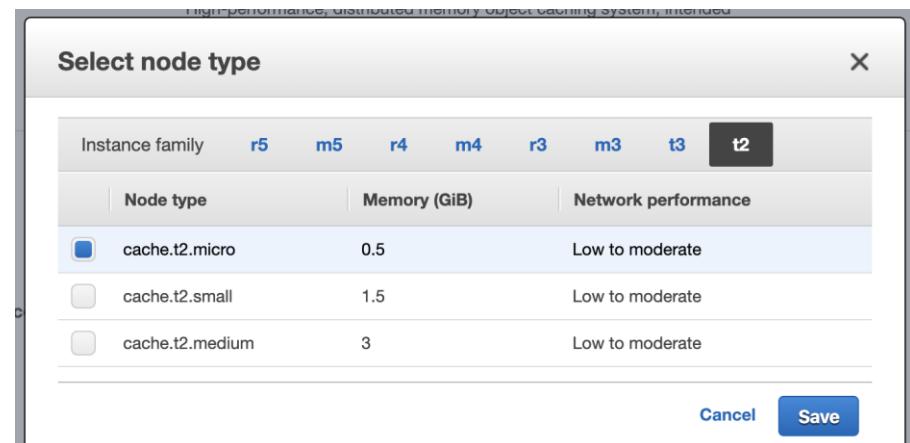
1. Open the [VPC console](#)
2. Note the default VPC ID

The screenshot shows the AWS VPC console for a VPC with ID vpc-036ef207f1f23c608. The 'Description' tab is selected. A 'Copy to clipboard' button is highlighted with a callout. The VPC details are as follows:

VPC ID	vpc-036ef207f1f23c608
State	available
IPv4 CIDR	172.31.0.0/16
IPv6 CIDR	-

## 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

▼ Advanced Redis settings  
Advanced settings have common defaults set to give you the fastest way to get started. You can modify these now or after your cluster is created.

Subnet group Create new

Name: redis-subnet

Description: Redis Subnet

VPC ID: vpc-036ef207f1f23c608

Subnets

Subnet ID	Availability zone	CIDR Block
subnet-0715be96ca93795d7	us-east-1e	172.31.48.0/20
subnet-0281cefdaa6d0b3b7	us-east-1b	172.31.32.0/20
subnet-0e1daeaabdb0a4eda1	us-east-1a	172.31.16.0/20
subnet-06f592dc54516186a	us-east-1f	172.31.64.0/20
subnet-0d1196ce2497a2091	us-east-1c	172.31.0.0/20
subnet-09d26c1ef33ce3aca	us-east-1d	172.31.80.0/20

Filter: Search Clusters... X

	Cluster Name	Mode	Shards	Nodes	Node Type	Status	Update Action	Status	Encryption in-transit	Encryption at-rest
	redis	Redis	0	1 node	cache.t2.micro	available	up to date	green	No	No
	redis	Redis	0	1 node	cache.t2.micro	available	up to date	green	No	No

Global Datastore: -  
Creation Time: April 9, 2020 at 7:26:23 PM UTC-7  
Status: available  
Update Status: up to date  
Reader Endpoint: -  
Node type: cache.t2.micro  
Shards: 0  
Multi-AZ: Disabled  
Parameter Group: default.redis5.0 (in-sync)  
Security Group(s): sg-080e56ef0039b4739 (VPC) (active)  
Maintenance Window: wed:07:00-wed:08:00  
Backup Window: Disabled  
Redis AUTH: No  
Customer Managed CMK: -

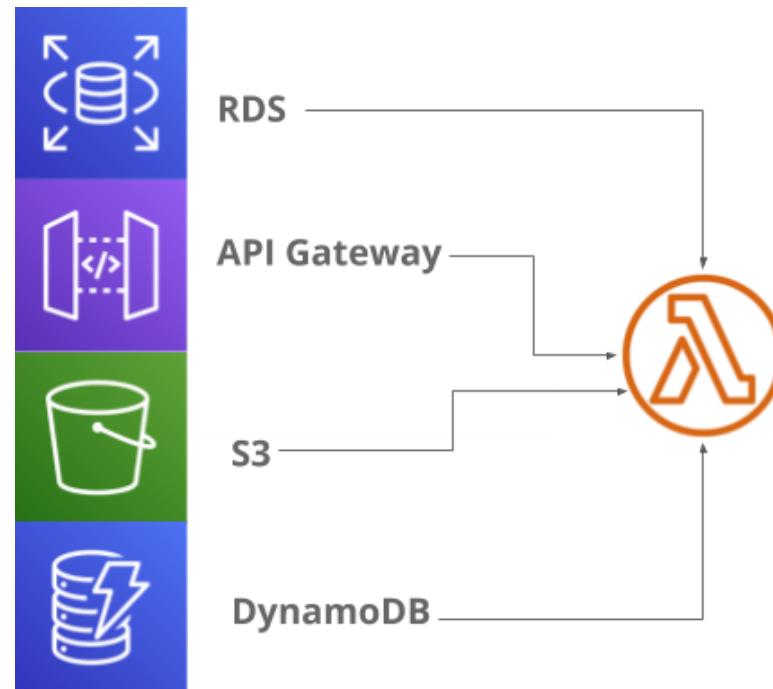
12. Scroll down to the **Backup** section and deselect **Enable Automatic backups**

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



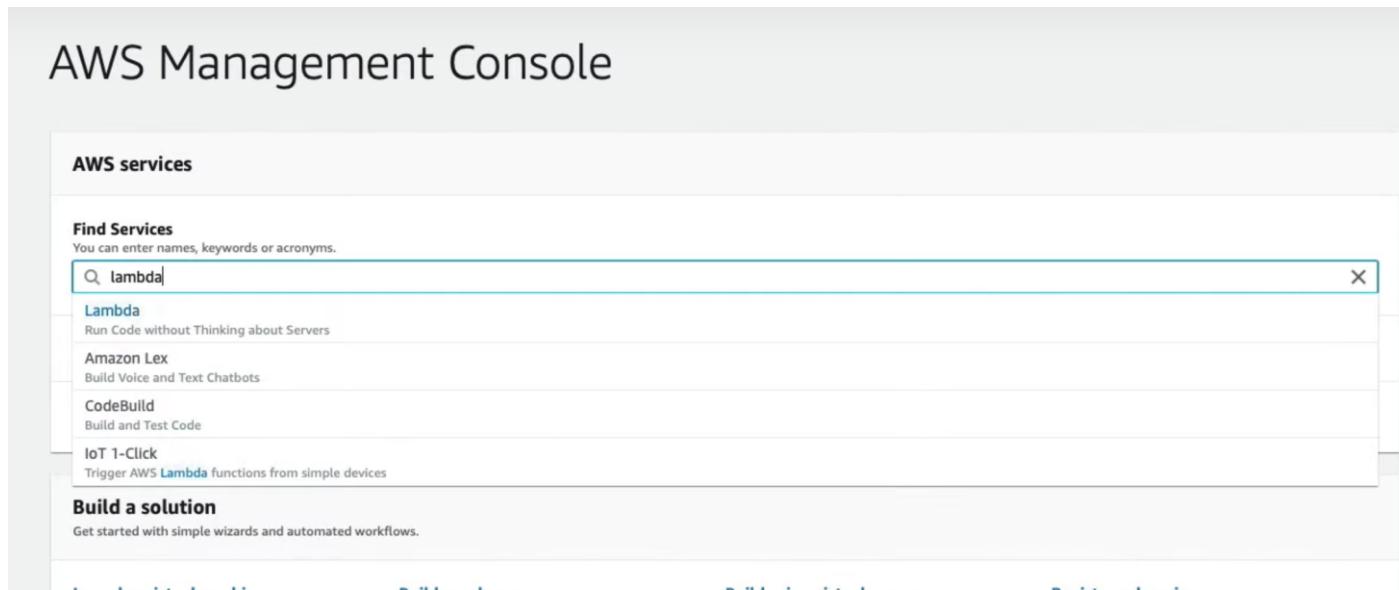
# Exercise: Hello from Lambda

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

Lambda > Functions > Create function

### Create function Info

Choose one of the following options to create your function.

**Author from scratch**

Start with a simple Hello World example.



hello

**Use a blueprint**

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



**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.  
 hello  
Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime Info**  
Choose the language to use to write your function.  
 Python 3.8

---

#### Permissions Info

Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▶ Choose or create an execution role

Cancel **Create function**

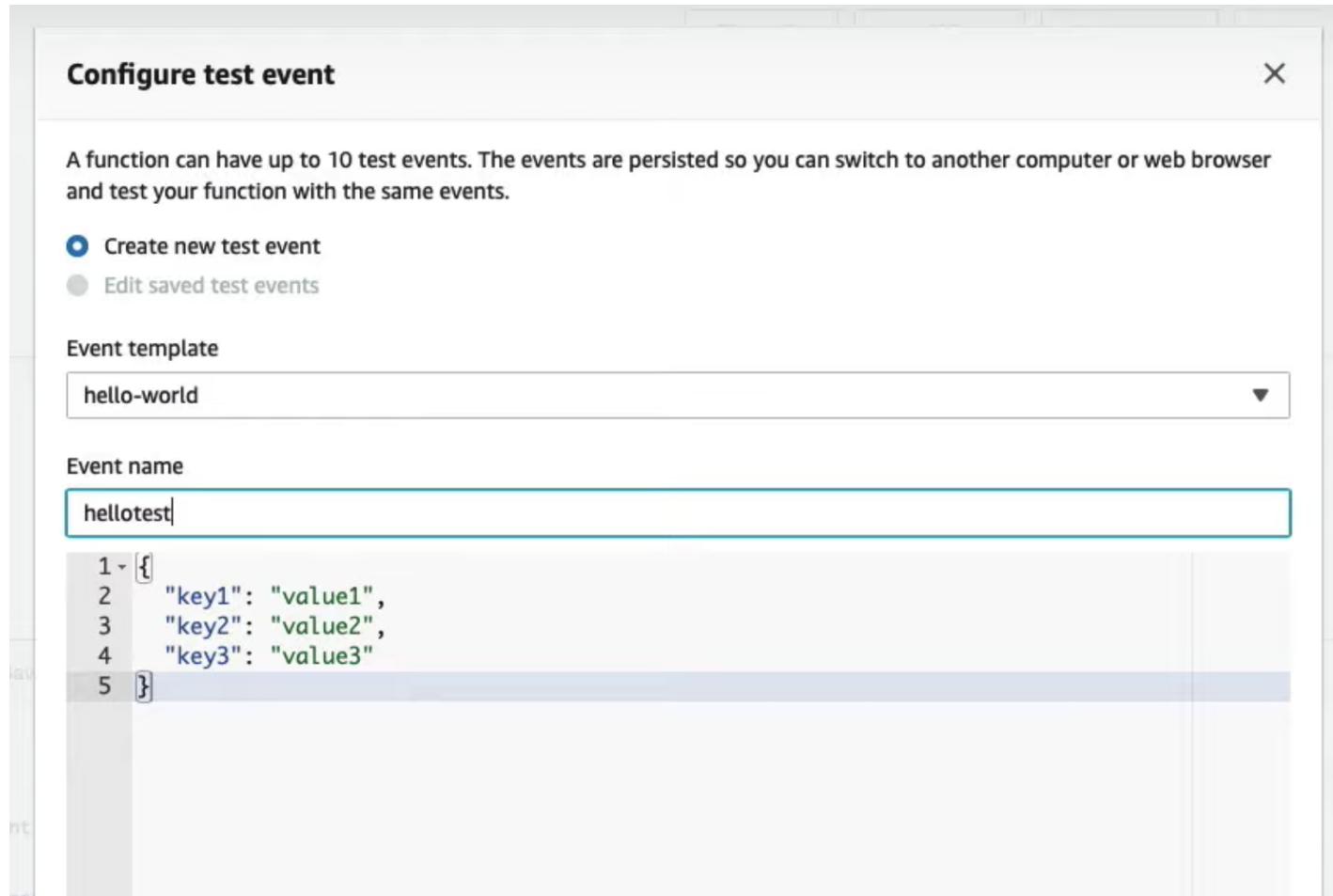
6. Notice the **handler** field on the embedded code page. Its format is **file-name.function-name**. In our case, the file name is **lambda\_function.py** and the code function name is **lambda\_handler**, which is why the handler is pre-set to **lambda\_function.lambda\_handler**.

The screenshot shows the AWS Lambda Embedded Code Editor for a function named "hello". The top navigation bar includes "Throttle", "Qualifiers", "Actions ▾", "Select a test event ▾", "Test", and "Save". The main area is titled "Function code Info". Under "Code entry type", it says "Edit code inline". Under "Runtime", it is set to "Python 3.8". Under "Handler", it is set to "lambda\_function.lambda\_handler". The code editor window displays the "lambda\_function" file with the following Python code:

```
1 import json
2
3 def lambda_handler(event, context):
4     # TODO implement
5     return {
6         'statusCode': 200,
7         'body': json.dumps('Hello from Lambda!')
8     }
9
```

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

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

Lambda > Functions > hello ARN - arn:aws:lambda:us-west-1

**hello**

Configuration Permissions Monitoring

Throttle Qualifiers Actions ▾ hellotest Test

## 10. Expand the Execution result

## 11. Verify the message: "Hello From Lambda!"

Lambda > Functions > hello ARN - arn:aws:lambda:us-west-1:91546476855:

### hello

Throttle Qualifiers Actions ▾ hellotest Test Save

Execution result: succeeded (logs)

▼ Details

The area below shows the result returned by your function execution. [Learn more about returning results from your function.](#)

```
{  
  "statusCode": 200,  
  "body": "\\"Hello from Lambda!\\\""  
}
```

**Summary**

Code SHA-256	Request ID
LD0MjQ04TMan6oFyOz3Y3mJqsPxm9lEhtimXGdIb8c=	32d0d3cc-806e-4c83-b2c1-3bdbb6e96e97
Duration	Billed duration
1.19 ms	100 ms
Resources configured	Max memory used
128 MB	51 MB

**Log output**

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

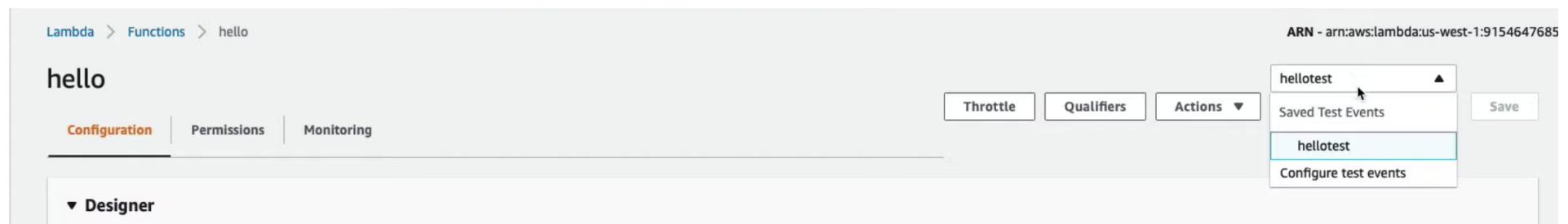
```
START RequestId: 32d0d3cc-806e-4c83-b2c1-3bdbb6e96e97 Version: $LATEST  
END RequestId: 32d0d3cc-806e-4c83-b2c1-3bdbb6e96e97  
REPORT RequestId: 32d0d3cc-806e-4c83-b2c1-3bdbb6e96e97 Duration: 1.19 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 51 MB
```

# Exercise: Lambda Events

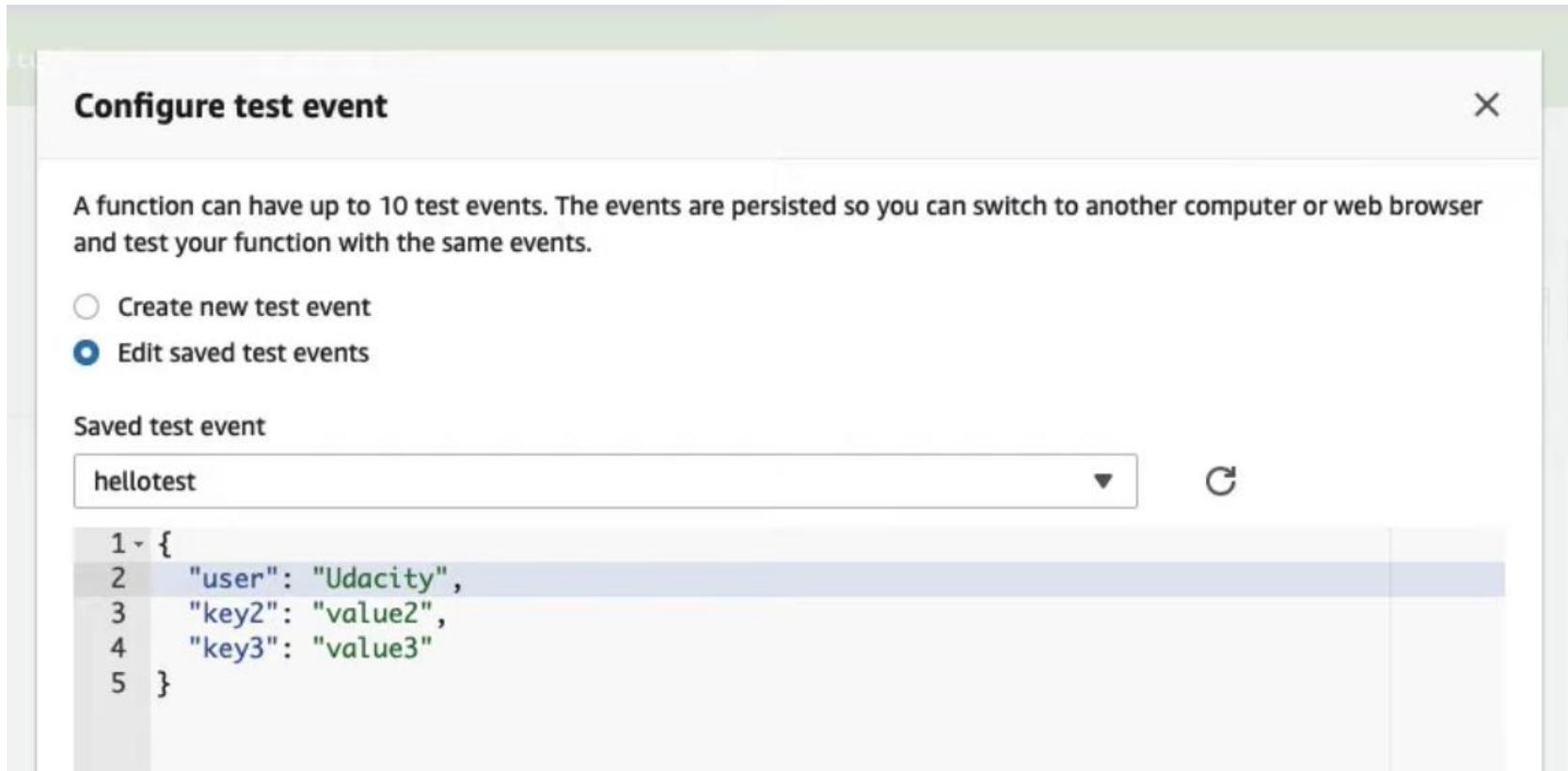
Modify the **testhello** event for the Lambda with the existing field:

```
{"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**



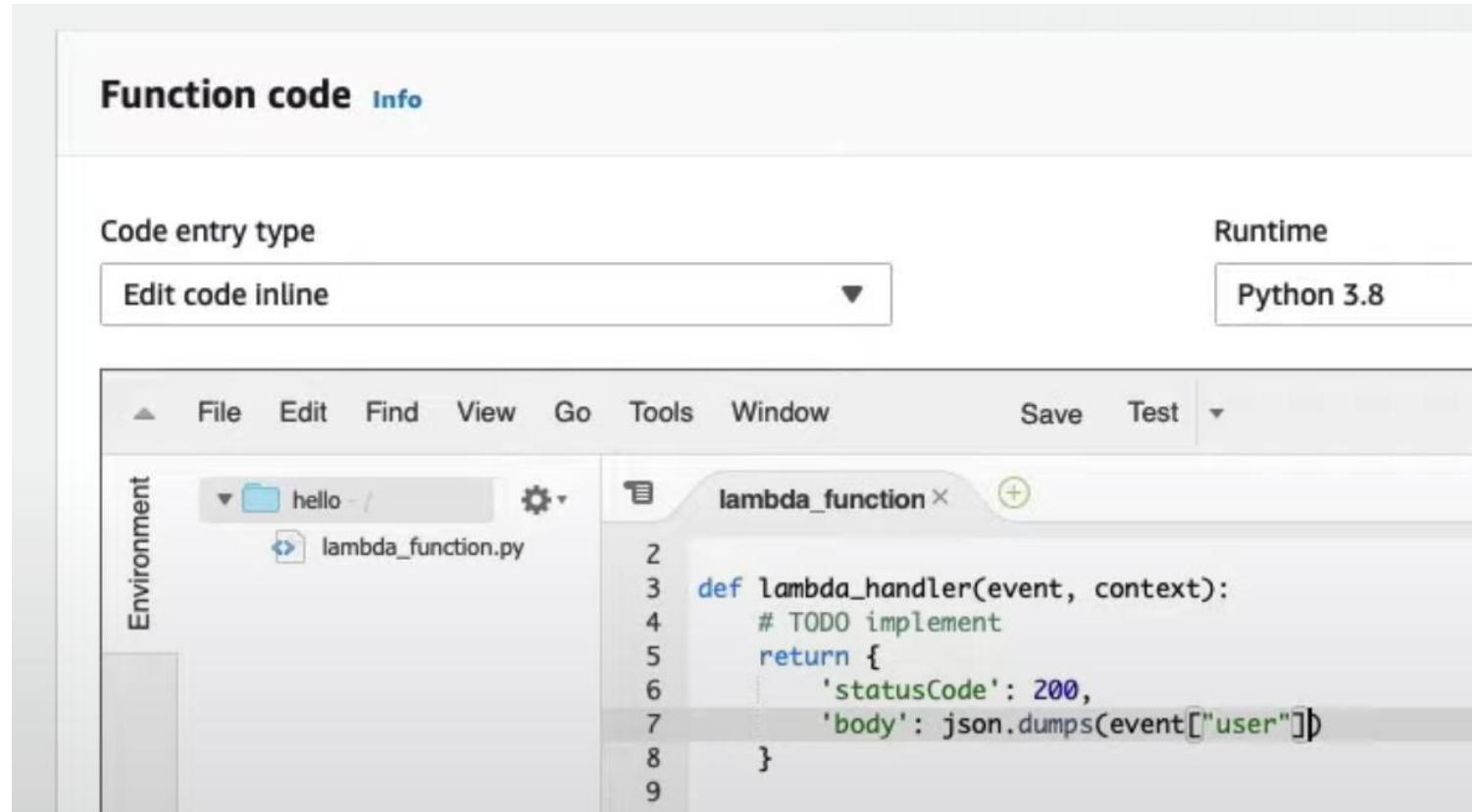
2. Select **Edit saved test events** option (it should be pre-selected already)
3. Change **key1** to be "**user**"
4. Change the value for "**user**" to be "**Udacity**"
5. Click Save Event



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

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

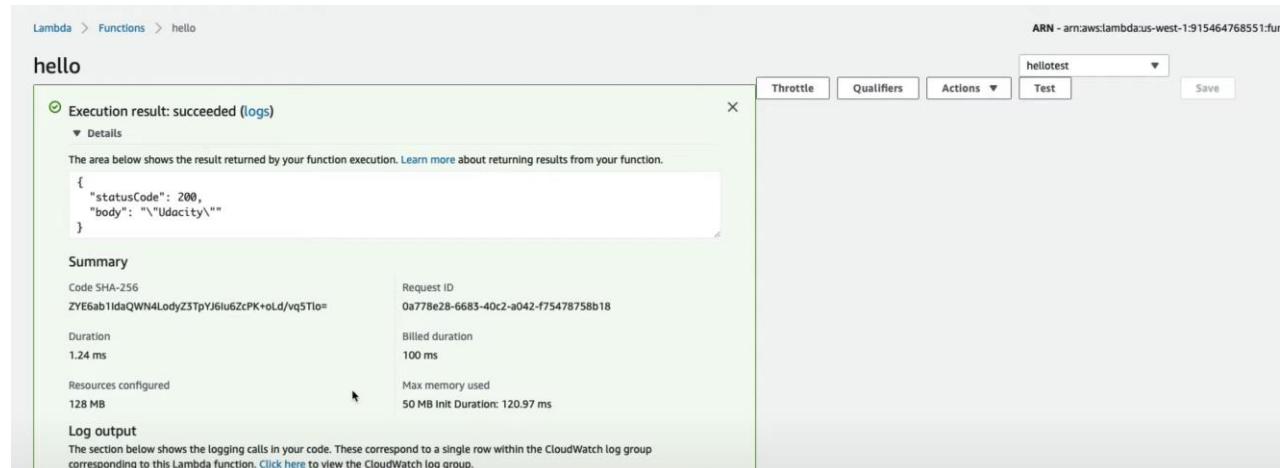
## 7. Click Save.



## 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 event 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**

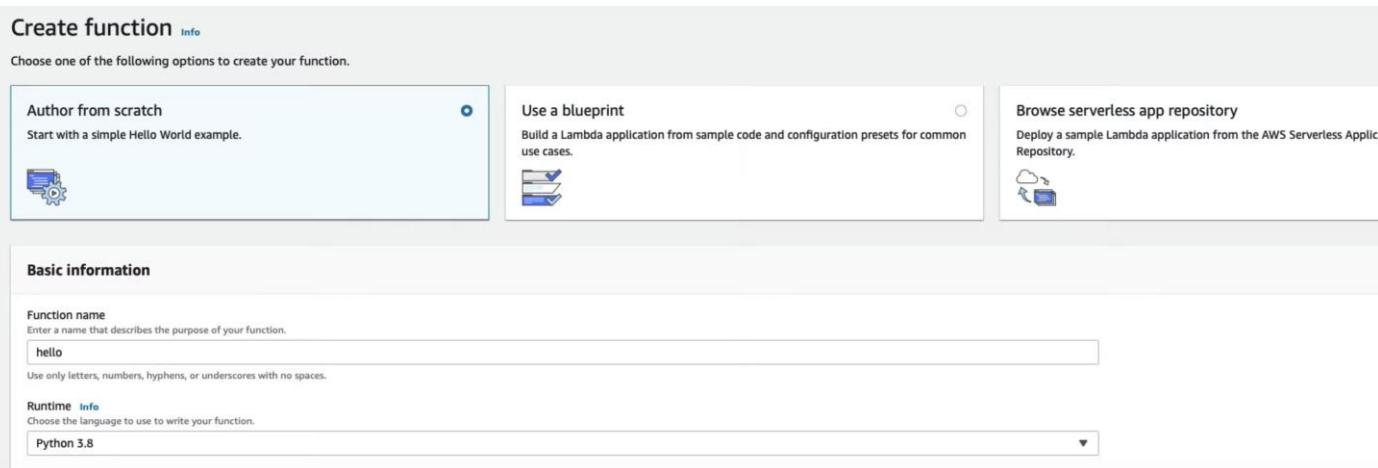
# Exercise: Uploading a Lambda Function

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

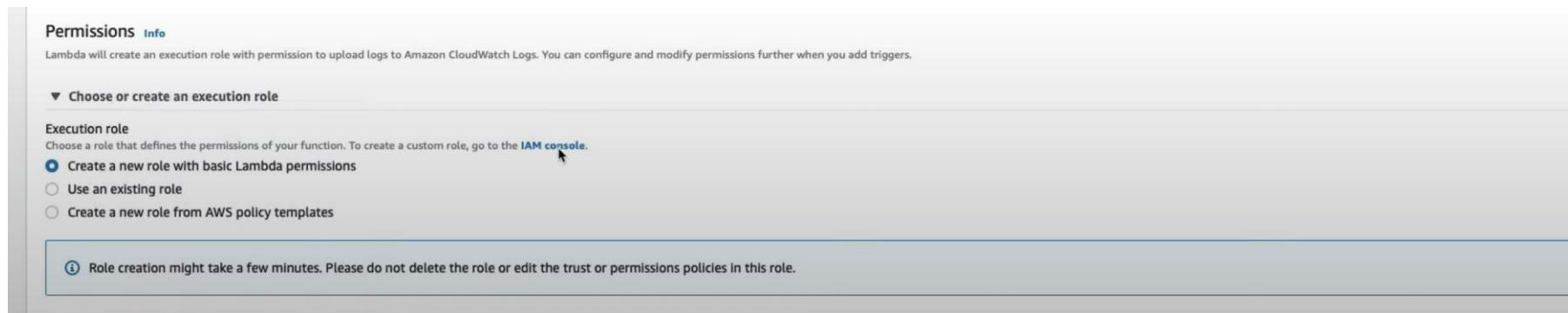
## Prerequisites

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

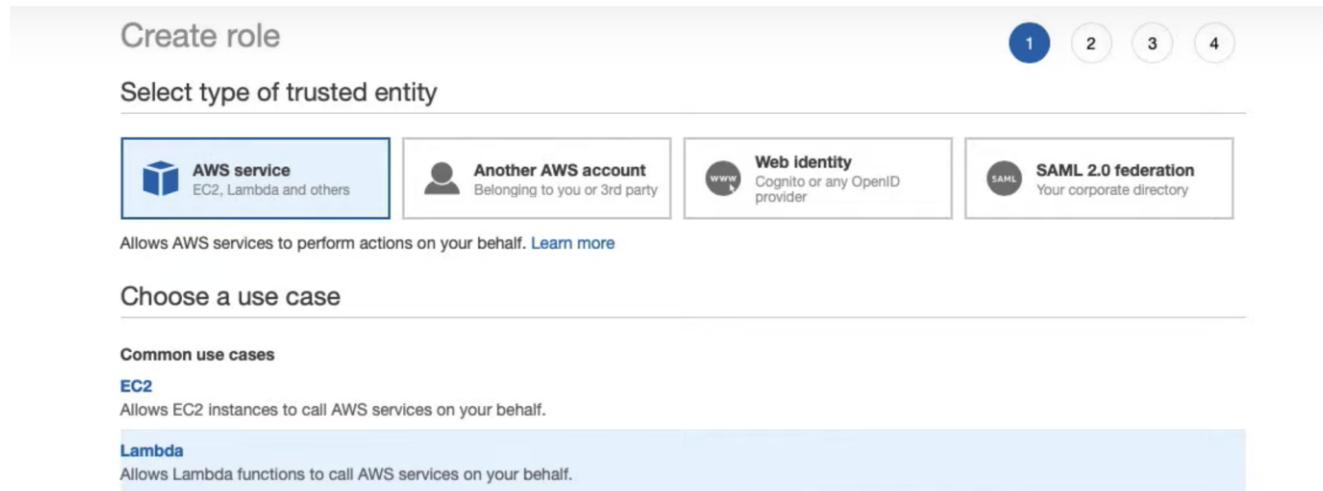
1. Navigate to the Lambda Console
2. Create a new Lambda Function
3. Use **Author from scratch, Python3.8**, and name the function **hello**



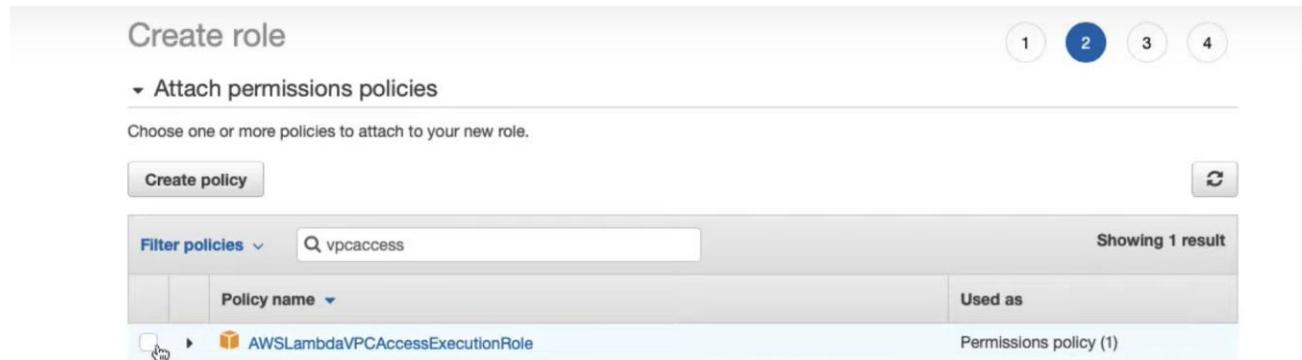
4. Expand the permissions section
5. Click the **IAM Console** link



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**

Create role

Review

Provide the required information below and review this role before you create it.

Role name\*  Use alphanumeric and '+,-,@-\_ characters. Maximum 64 characters.

Role description  Maximum 1000 characters. Use alphanumeric and '+,-,@-\_ characters.

Trusted entities

Policies  AWSLambdaVPCAccessExecutionRole

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

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

hello

+ Add trigger

Throttle Qualifiers Actions Select a test event Test Save

+ Add destination

Function code Info

File Edit Find View Go Tools Window Save Test

Environment hello / lambda\_function.py

Actions ▲

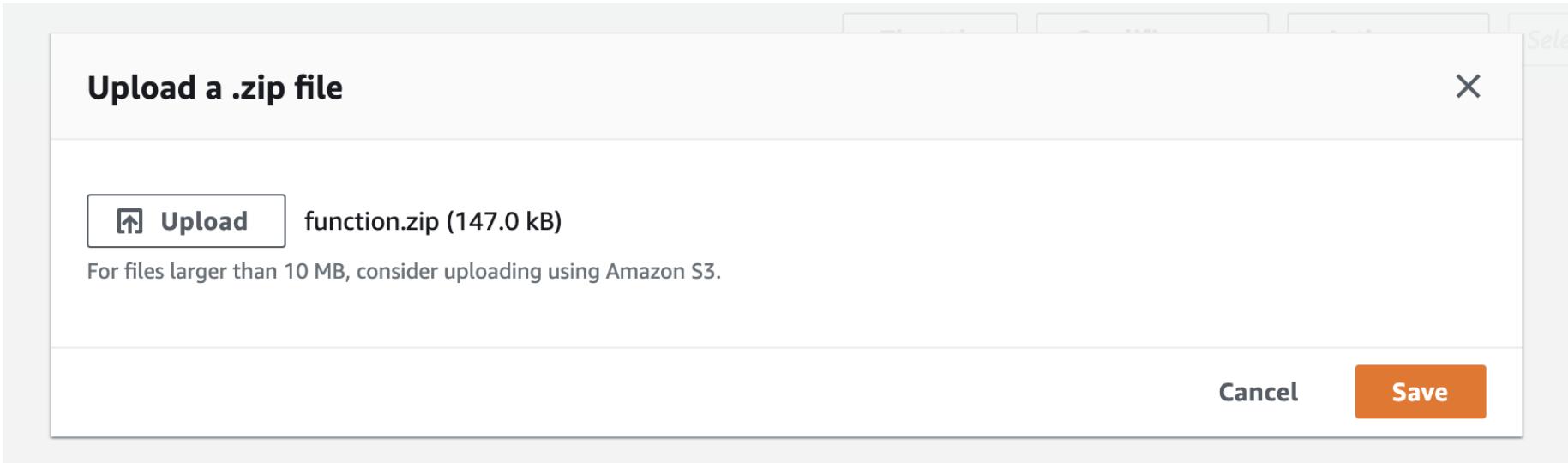
Import json

def lambda\_handler(event, context):  
 # TODO implement  
 return {  
 'statusCode': 200,  
 'body': json.dumps('Hello from Lambda!')  
 }

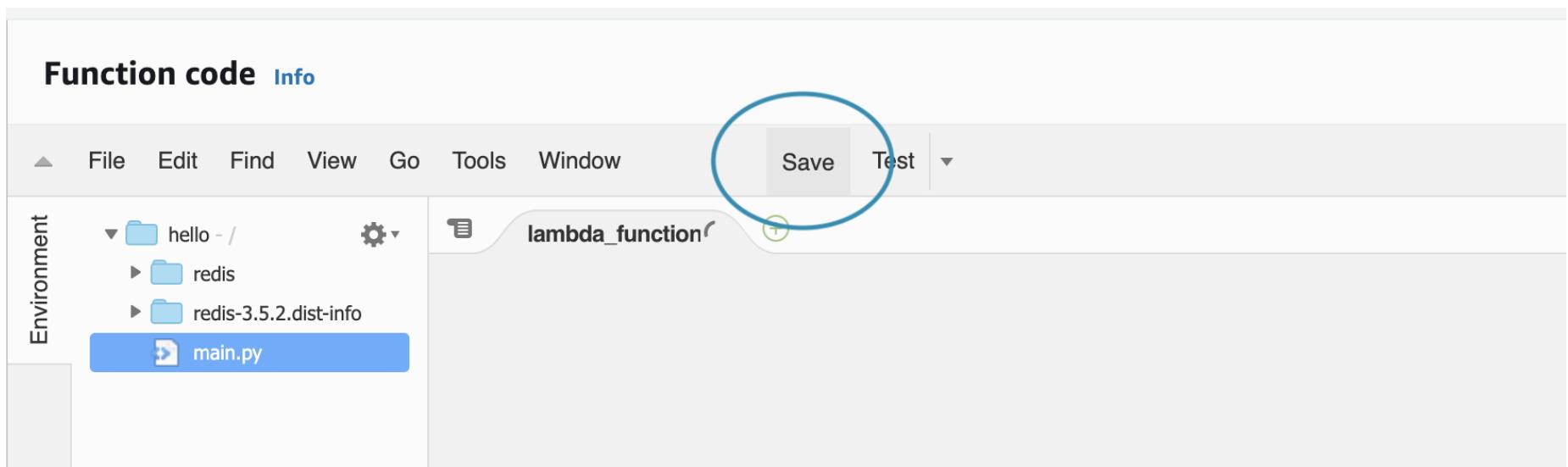
Upload a .zip file

Upload a file from Amazon S3

### 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

Basic settings		<a href="#">Edit</a>
Description	-	Runtime
Handler	<a href="#">Info</a>	Memory (MB)
	lambda_function.lambda_handler	<a href="#">Info</a>
Timeout	<a href="#">Info</a>	128
	0 min 3 sec	

## 16. Set the Handler field to `main.handler`

### Edit basic settings

#### Basic settings

Description - *optional*

#### Runtime

Python 3.8 ▾

#### Handler [Info](#)

main.handler

#### Memory (MB) [Info](#)

Your function is allocated CPU proportional to the memory configured.



128 MB

#### Timeout [Info](#)

0 min 3 sec

17. Select **Use an existing role**

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

## Edit basic settings

### Basic settings

Description - *optional*

Runtime

Python 3.8

Handler [Info](#)

main.handler

Memory (MB) [Info](#)  
Your function is allocated CPU proportional to the memory configured.

128 MB

Timeout [Info](#)

0 min 3 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role

Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

lambda-vpc-role

[View the lambda-vpc-role role on the IAM console.](#)

# 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

The screenshot shows the ElastiCache Dashboard interface. On the left, there is a sidebar with navigation links: ElastiCache Dashboard, Create, Actions, Memcached, Redis (selected), Global Datastore, Service Updates, Reserved Nodes, Backups, Parameter Groups, Subnet Groups, Events, and ElastiCache Cluster Client. The main area displays a table titled 'Clusters' with a search bar at the top. The columns are: Cluster Name, Mode, Shards, Nodes, Node Type, Status, Update Action Status, Encryption in-transit, Encryption at-rest, Global Datastore, and Global. A cluster named 'redis' is listed, showing 0 shards, 1 node, node type 'cache.t2.micro', status 'available', update action 'up to date', and encryption settings. The 'redis' link in the Cluster Name column is being clicked.

The screenshot shows the ElastiCache Dashboard interface, specifically the details for the 'redis' cluster. The left sidebar is identical to the previous screenshot. The main area has a title 'Name: redis' with tabs for Description and Nodes (selected). Below the tabs are buttons for 'Add Replication' and 'Actions'. The table below lists the cluster's nodes. The columns are: Node Name, Status, Port, Endpoint, Parameter Group Status, Zone, and Created on. One node is listed: 'redis' with status 'available', port '6379', endpoint 'redis.2ulaxe.0001.usw1.cache.amazonaws.com', parameter group status 'in-sync', zone 'us-west-1a', and created on 'May 27, 2020 at 9:44:19 PM UTC-7'.

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**

Lambda > Functions > hello > Edit environment variables

## Edit environment variables

### Environment variables

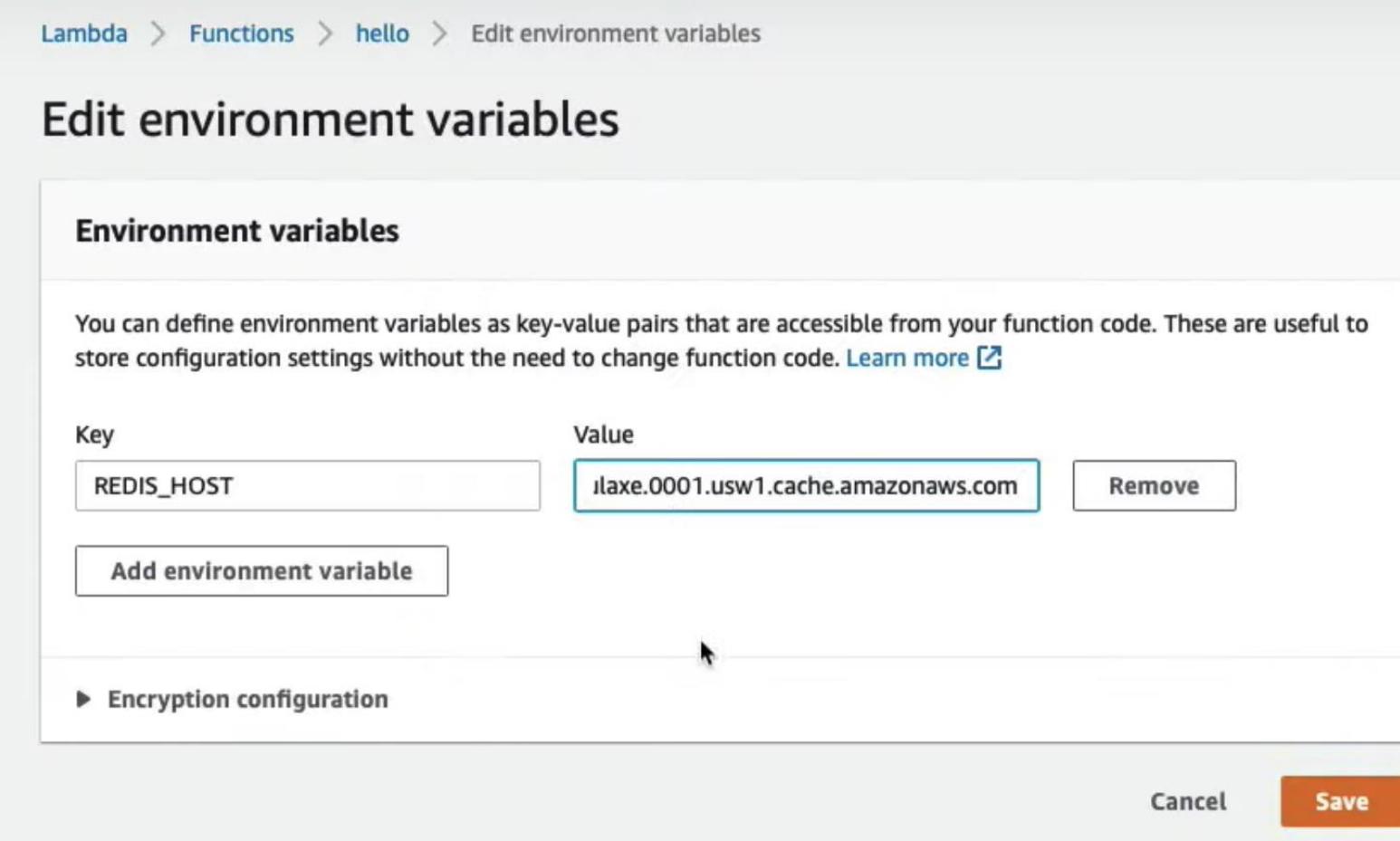
You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	Remove
REDIS_HOST	ilaxe.0001.usw1.cache.amazonaws.com	<button>Remove</button>

[Add environment variable](#)

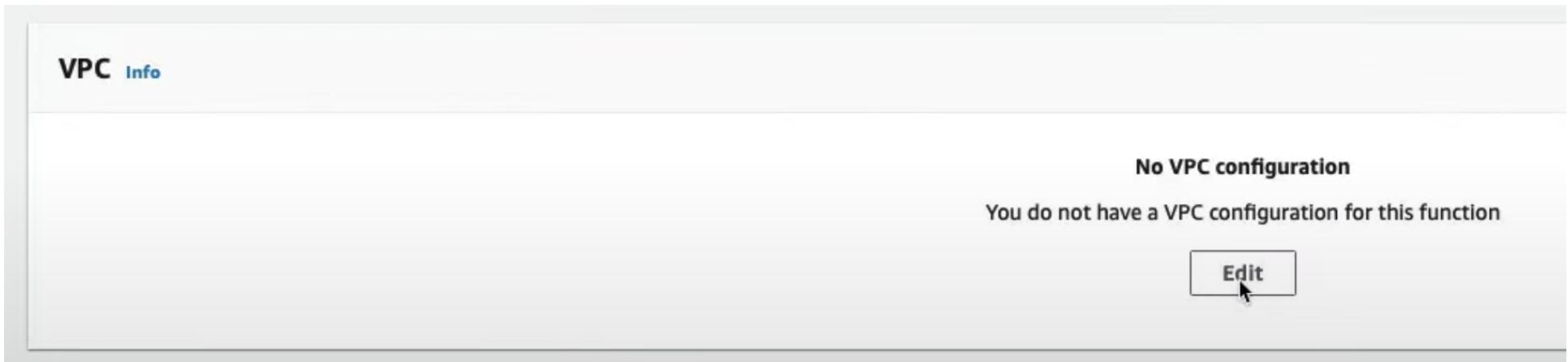
▶ Encryption configuration

[Cancel](#) [Save](#)



## 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

i 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.

▲	
<input type="text"/> Q	
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.

▲	
<input type="text"/> Q	
sg-b05075c3 (default)	
default VPC security group	



✓ Successfully updated the function hello.

# Exercise: Testing Lambda

- Configure a test event with:

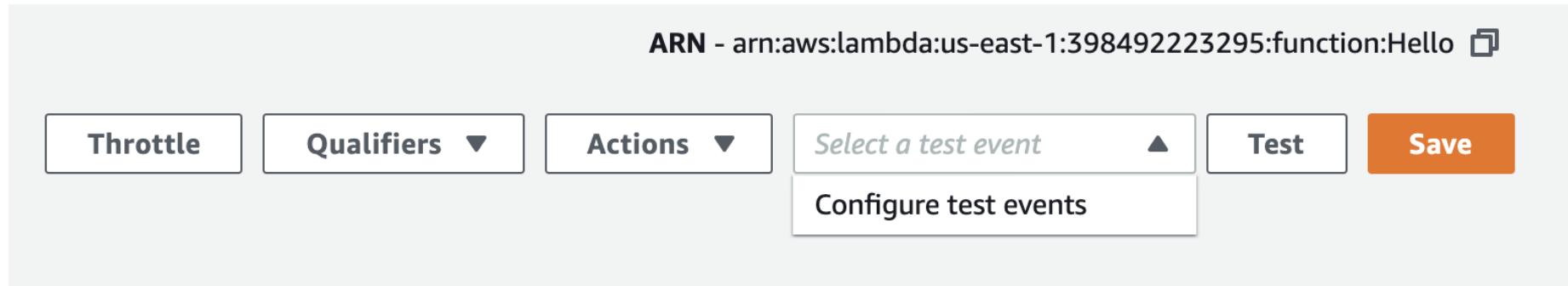
```
{ "name": "John Doe" }
```

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"}
```

**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 brows and test your function with the same events.

Create new test event  
 Edit saved test events

Event template

```
Hello World
```

Event name

```
Hello
```

1 {  
2 "name": "John Doe"  
3 }

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

## Hello

- Execution result: succeeded ([logs](#))

### ▼ Details

The area below shows the result returned by your function execution. [Learn more about returning results from your functions](#)

```
"Success! John doe was written to Redis"
```

### Summary

Code SHA-256

KzBfFcVOfaHRqnLxUWcqzP8TBFxo1amhfMm9uEEmR1g=

Duration

84.55 ms

Resources configured

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**"

## Hello

Execution result: succeeded ([logs](#))

▼ Details

The area below shows the result returned by your function execution. [Learn more about returning results from your functions](#)

```
"Hello John doe nice to meet you"
```

### Summary

Code SHA-256

KzBfFcVOfaHRqnLxUWcqzP8TBFxo1amhfMm9uEEmR1g=

Duration

34.84 ms

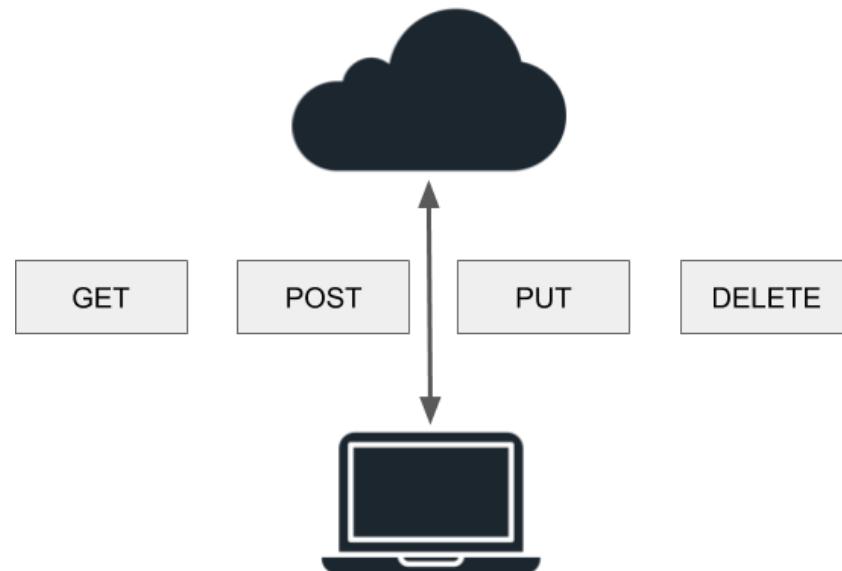
Resources configured

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

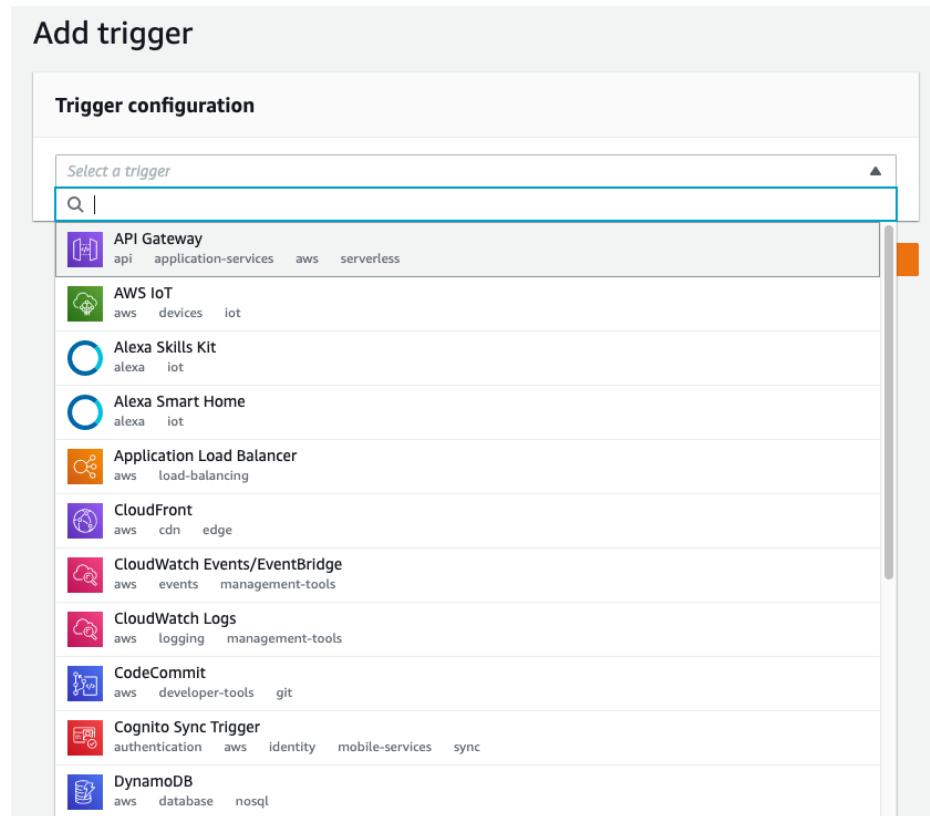


# Exercise: API Gateway

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

## Add trigger

**Trigger configuration**

 API Gateway  
api application-services aws serverless

Add an API to your Lambda function to create an HTTP endpoint that invokes your function. API Gateway supports two types of RESTful APIs: HTTP APIs and REST APIs. [Learn more](#)

**API**  
Create a new API or attach an existing one.

**API type**

**HTTP API**  
Create an HTTP API.

**REST API**  
Create a REST API.

**Security**  
Configure the security mechanism for your API endpoint.  
  
Don't add any authorization or authentication requirements. Any user can invoke your function with an HTTP call.

**► Additional settings**

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

**Cancel** **Add**

# Hello

[Throttle](#)[Qualifiers ▾](#)[Actions](#)[Configuration](#)[Permissions](#)[Monitoring](#)

## ▼ Designer



Hello



Layers

(0)



API Gateway

[+ Add trigger](#)

## API Gateway

### Hello-API

arn:aws:execute-api:us-east-1:398492223295:xw3e2f4i9i/\*/\*/Hello

▶ API: [api-gateway/xw3e2f4i9i/\\*/\\*/Hello](#) API endpoint: <https://xw3e2f4i9i.execute-api.us-east-1.amazonaws.com/default>Hello> API name: Hello-API

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.

The screenshot shows the official Postman website homepage. At the top, there's a navigation bar with links for Product, How Collaboration Works, Use Cases, Pricing, Enterprise, Explore, Learning Center, and Sign In. The main title "The Collaboration Platform for API Development" is prominently displayed. Below the title, there's a callout for "Download the free Postman app to get started." followed by a "Download the App" button. To the right, there's a large, stylized illustration of a robotic rover or satellite in space, surrounded by celestial bodies and orbits. At the bottom, three key statistics are highlighted: "10 million Developers", "500,000 Companies", and "250 million APIs".

POSTMAN

Product ▾ How Collaboration Works Use Cases ▾ Pricing Enterprise Explore

Learning Center Sign In

# The Collaboration Platform for API Development

Download the free Postman app to get started.

Download the App

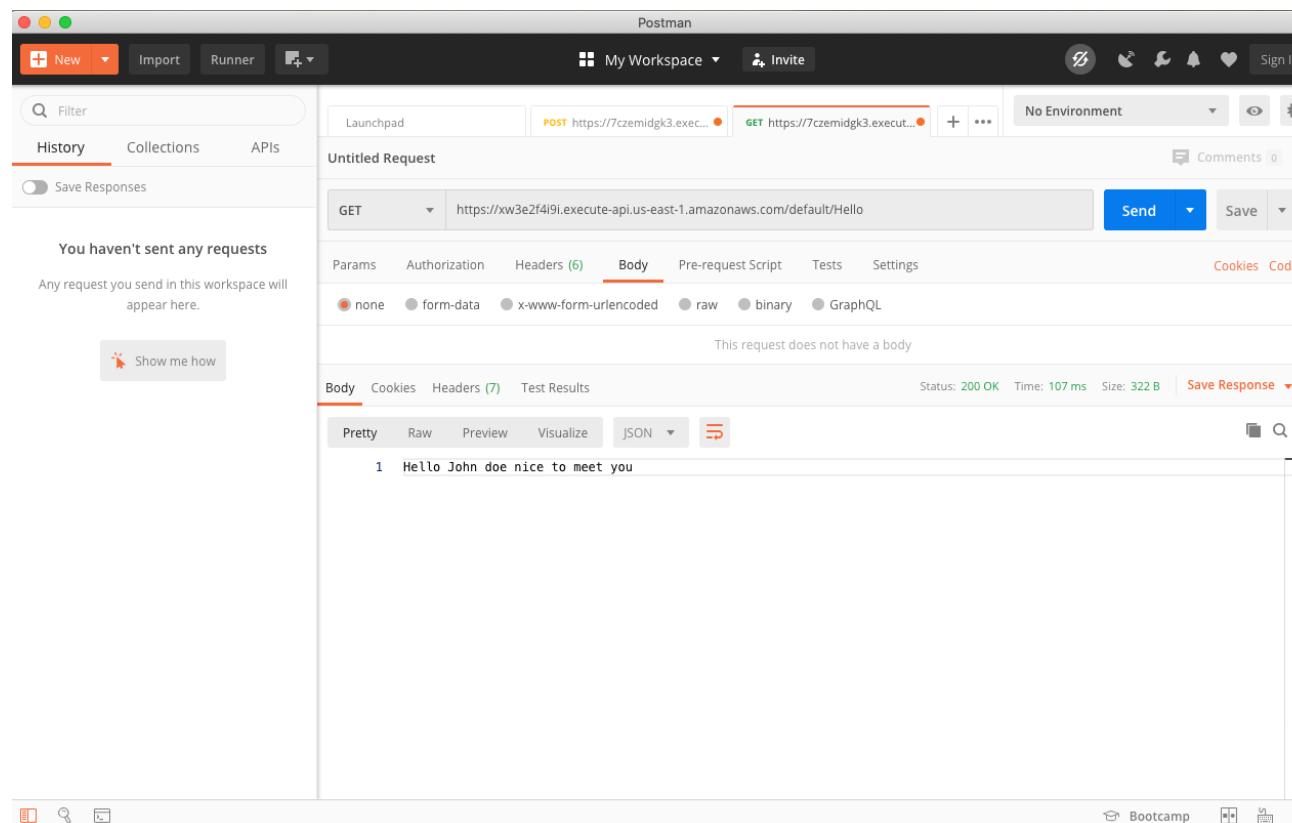
10 million Developers

500,000 Companies

250 million APIs

# 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"}
```

The screenshot shows the Postman application interface. In the top navigation bar, 'My Workspace' is selected. The main workspace displays an 'Untitled Request' for a POST method to the URL `https://xw3e2f4i9i.execute-api.us-east-1.amazonaws.com/default/Hello`. The 'Body' tab is active, showing the JSON payload `{"name": "greatest student ever"}`. Below the request details, the response status is shown as `Status: 200 OK Time: 211 ms Size: 342 B`, and the response body contains the message `Success! Greatest student ever was written to Redis`.

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

The screenshot shows the Postman application interface. At the top, there's a navigation bar with 'Postman' and various icons like 'New', 'Import', 'Runner', 'Invite', and 'Sign In'. Below the navigation bar, there's a search bar labeled 'Filter' and tabs for 'History', 'Collections', and 'APIs'. A 'Save Responses' toggle switch is also present.

In the main workspace, there are two requests listed: a 'POST' request to 'https://7czemidgk3.execute-api.us-east-1.amazonaws.com/default/Hello' and a 'GET' request to the same URL. The 'GET' request is currently selected. The 'Body' tab is active, showing the message 'This request does not have a body'. The 'Headers' tab shows 6 headers. The 'Tests' and 'Settings' tabs are also visible.

At the bottom of the request details, it says 'Status: 200 OK Time: 116 ms Size: 335 B Save Response'. Below this, there are tabs for 'Body', 'Cookies', 'Headers (7)', and 'Test Results'. The 'Body' tab is active, displaying the response content: '1 Hello Greatest student ever nice to meet you'. There are also 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON' buttons.

At the very bottom of the interface, there are several small icons: a file icon, a magnifying glass, a square, a question mark, and a help icon.

## 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 Application](#)

## **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](#)