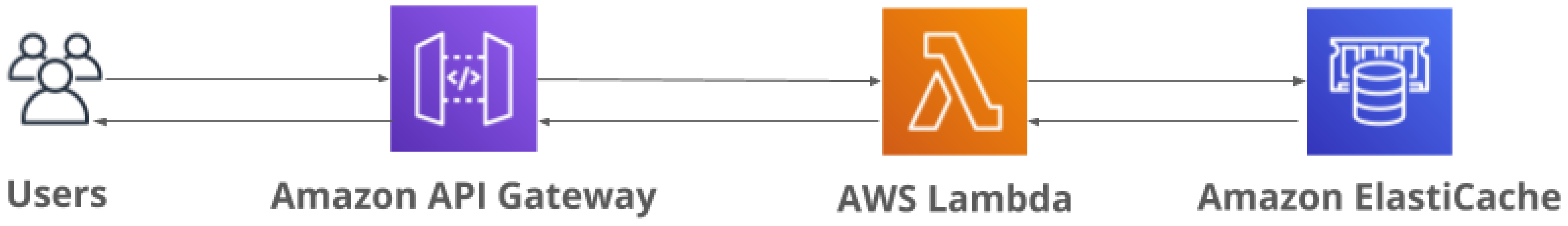
Serverless concept

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.

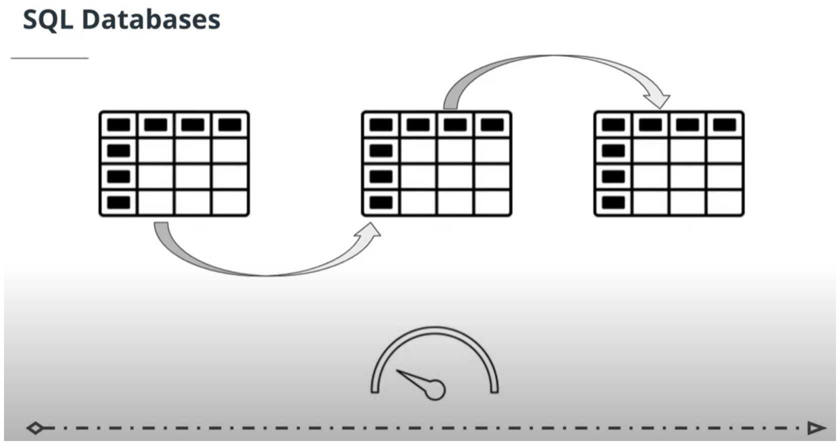


Intro to Cloud Computing https://classroom.udacity.com/courses/ud080/lessons/6860cb92-05c5-4de6-bba8-679709fb21f0/concepts/3425934a-5c11-4136-b5be-0df48c937f7a

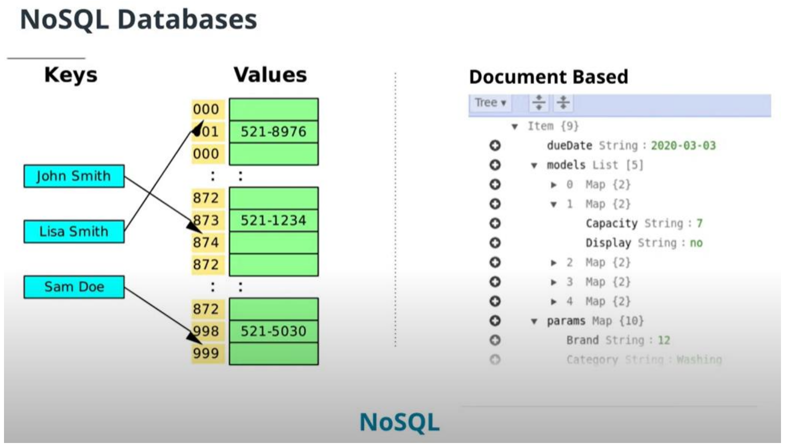
ElastiCache

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.

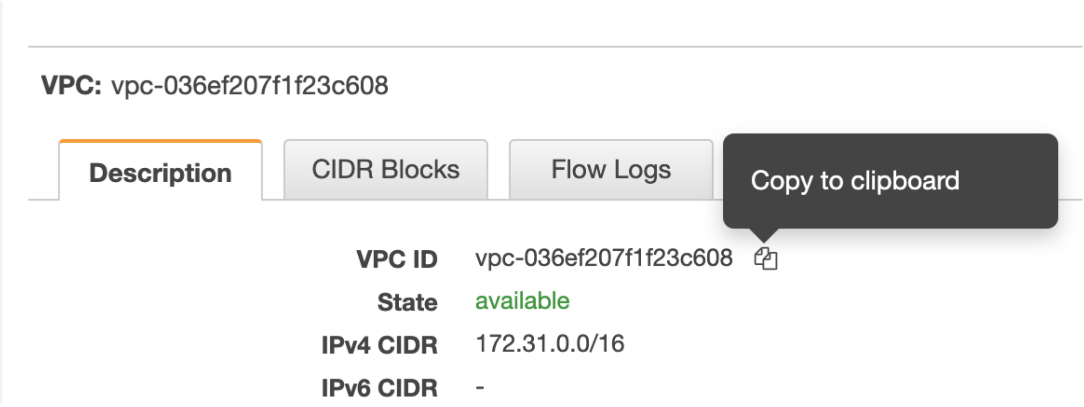
Instructions

- **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

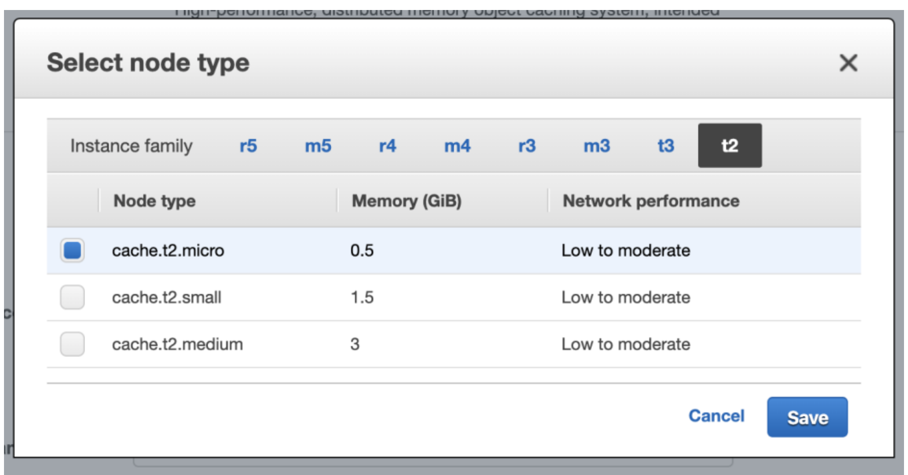


VPC: vpc-036ef207f1f23c608

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

Create the ElastiCache Redis Instance

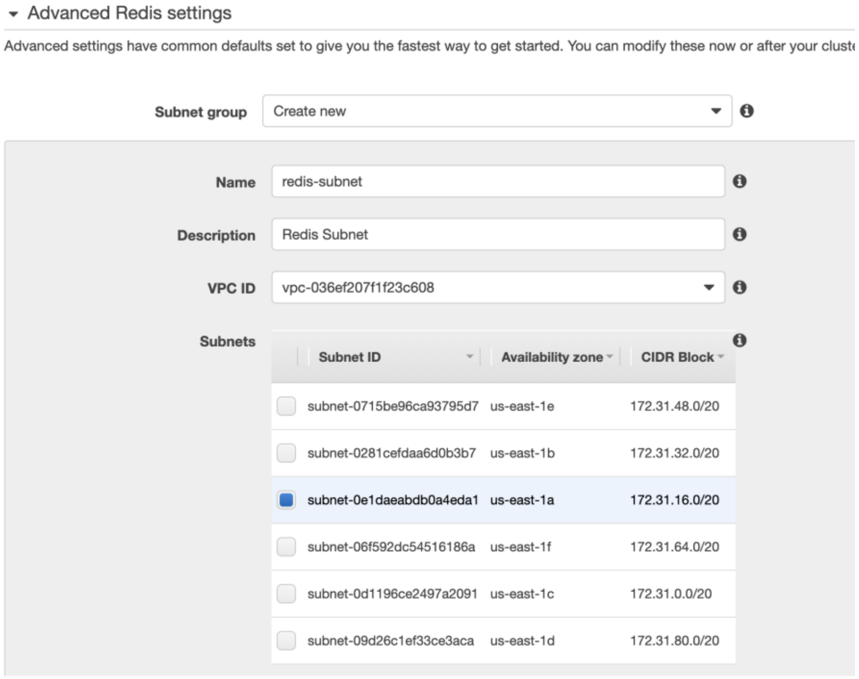
1. Open the ElastiCache console
2. Change the region on the top right to be **us-east-1 N.Virginia**
3. Click **Redis** on the sidebar menu, then click the **Create** button
4. Name the instance **redis**
5. Change the number of replicas to 0
6. Change the **Node type** to be **t2.micro**
7. Save



Select node type

	Node type	Memory (GiB)	Network performance
■	cache.t2.micro	0.5	Low to moderate
□	cache.t2.small	1.5	Low to moderate
□	cache.t2.medium	3	Low to moderate

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 clust

Subnet group: Create new

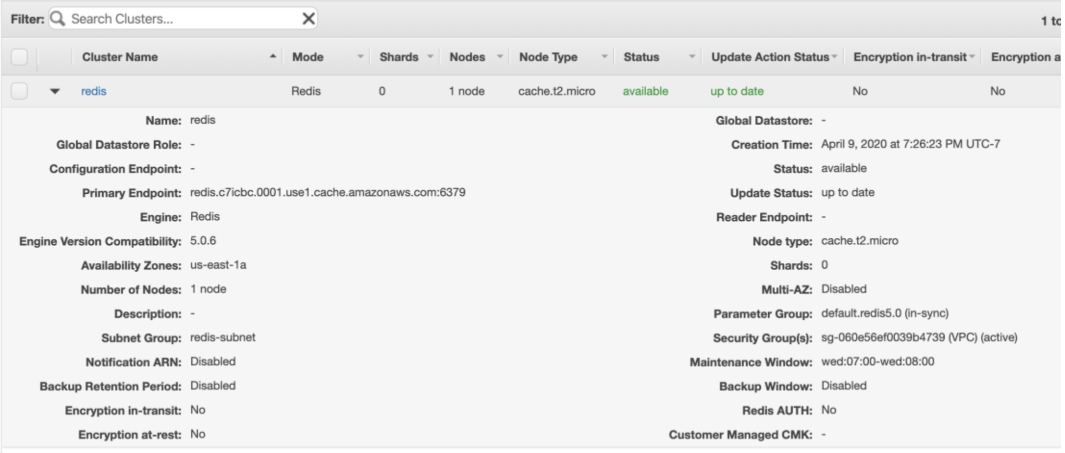
Name: redis-subnet

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-0e1daeabdb0a4eda1	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... 1 to

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

Name: redis
Global Datastore Role: -
Configuration Endpoint: -
Primary Endpoint: redis.c7icbc.0001.use1.cache.amazonaws.com:6379
Engine: Redis
Engine Version Compatibility: 5.0.6
Availability Zones: us-east-1a
Number of Nodes: 1 node
Description: -
Subnet Group: redis-subnet
Notification ARN: Disabled
Backup Retention Period: Disabled
Encryption in-transit: No
Encryption at-rest: No

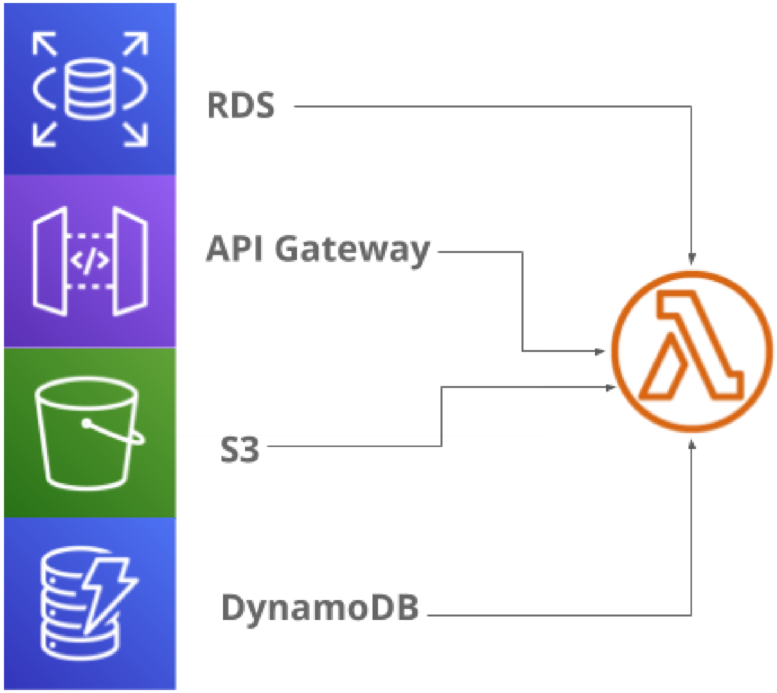
Global Datastore: -
Creation Time: April 9, 2020 at 7:26:23 PM UTC-7
Status: available
Update Status: up to date
Reader Endpoint: -
Node type: cache.t2.micro
Shards: 0
Multi-AZ: Disabled
Parameter Group: default.redis5.0 (in-sync)
Security Group(s): sg-060e56ef0039b4739 (VPC) (active)
Maintenance Window: 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



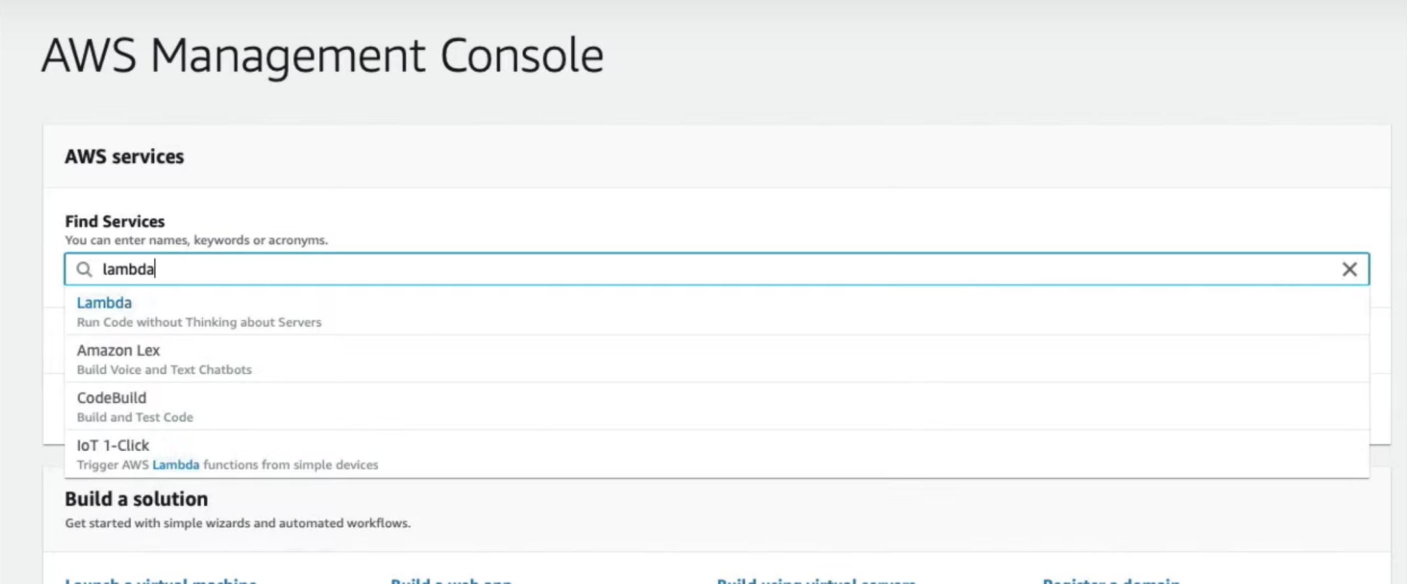
Exercise: Hello from Lambda

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**



AWS Management Console

AWS services

Find Services

You can enter names, keywords or acronyms.

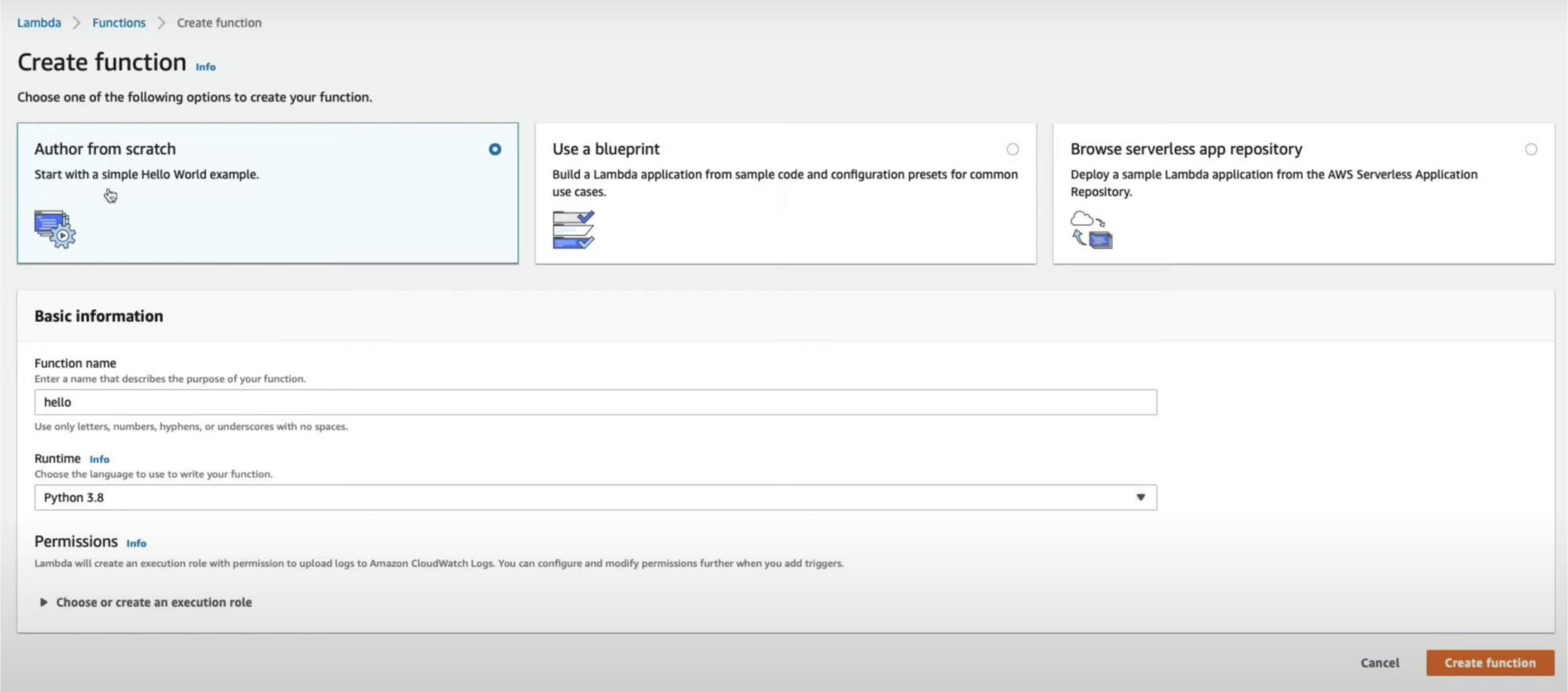
lambda

Lambda
Run Code without Thinking about Servers
Amazon Lex
Build Voice and Text Chatbots
CodeBuild
Build and Test Code
IoT 1-Click
Trigger AWS Lambda functions from simple devices

Build a solution

Get started with simple wizards and automated workflows.

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



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.

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

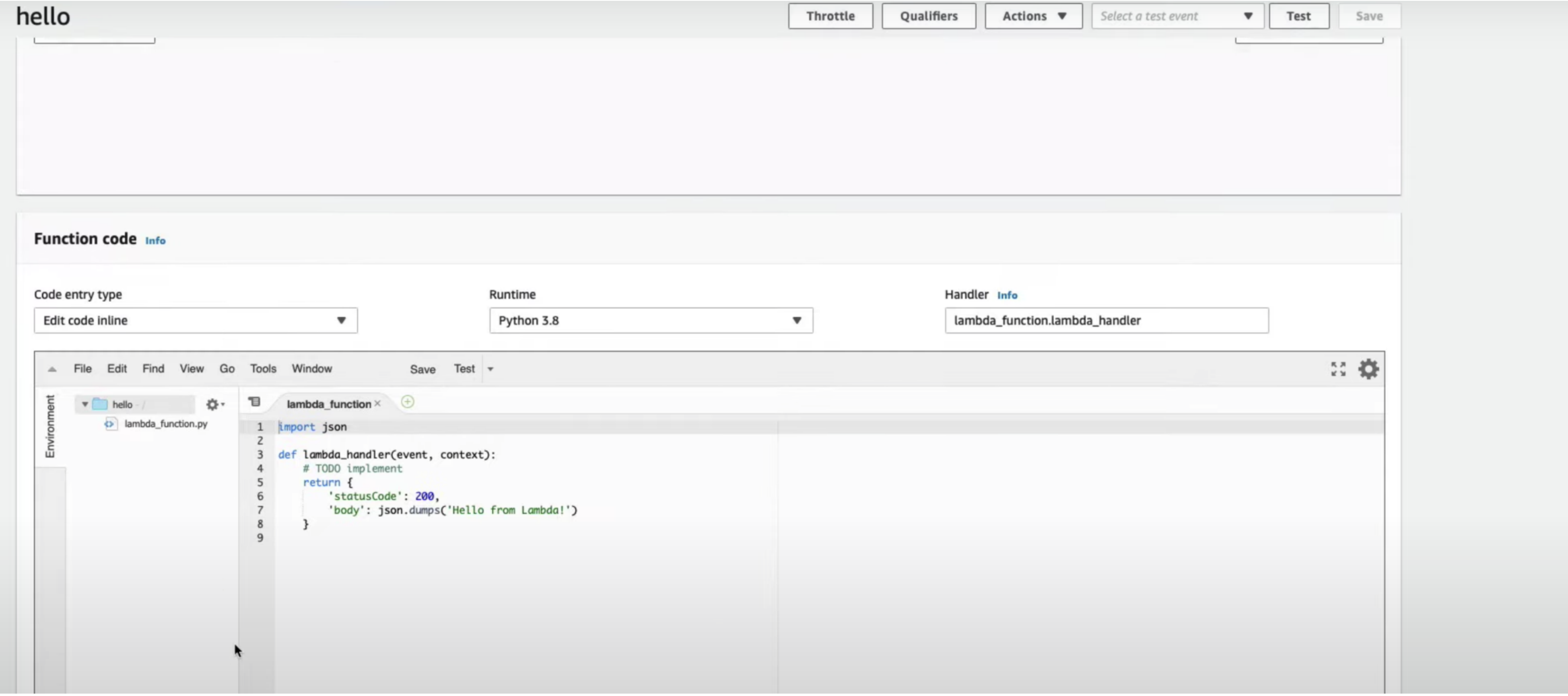
Basic information

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.

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**.



Function code

Code entry type: Edit code inline

Runtime: Python 3.8

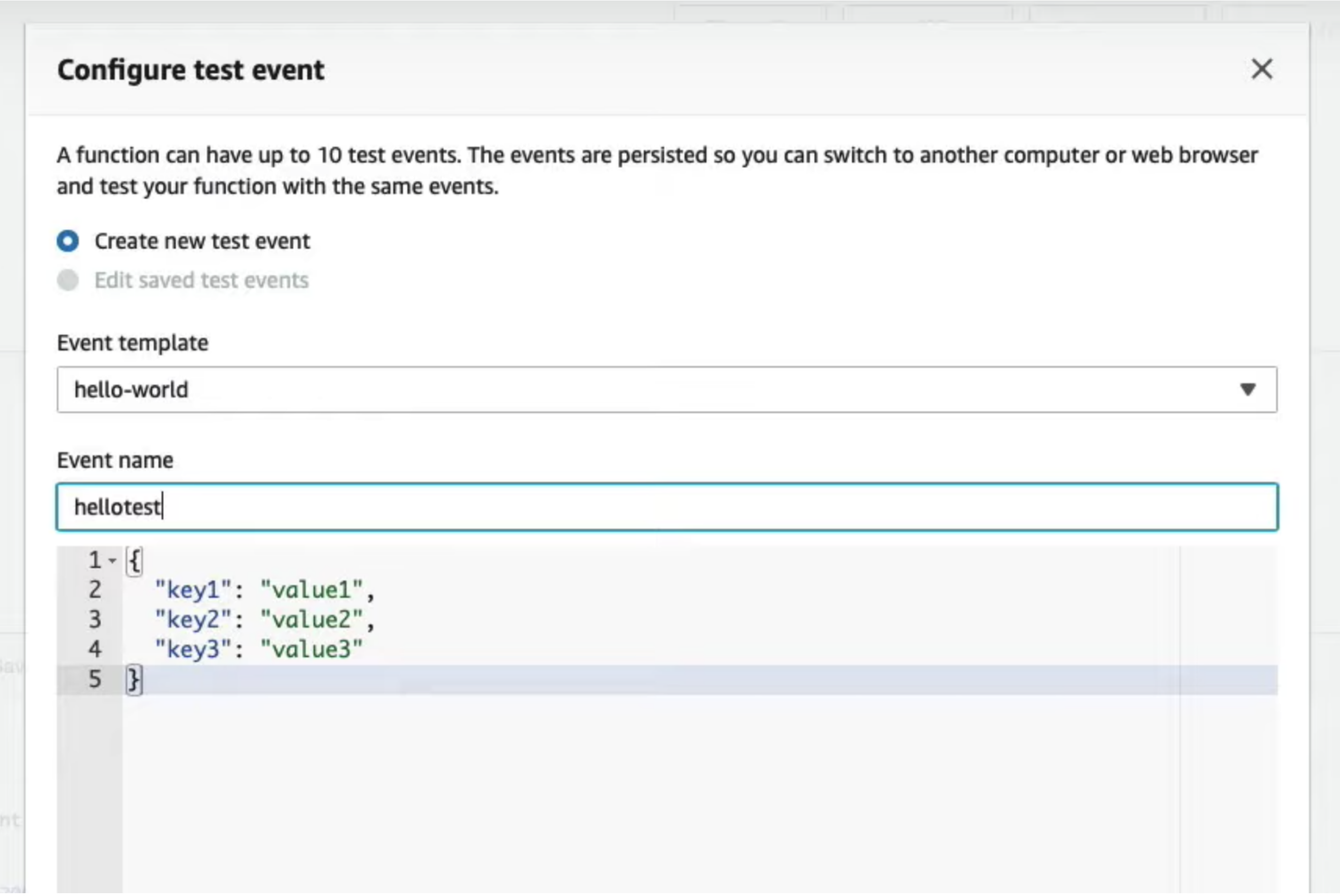
Handler: lambda_function.lambda_handler

```
import json

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

7. Click on the **Test** button and configure a test event.

8. Name the event **hellotest** and save the event.



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

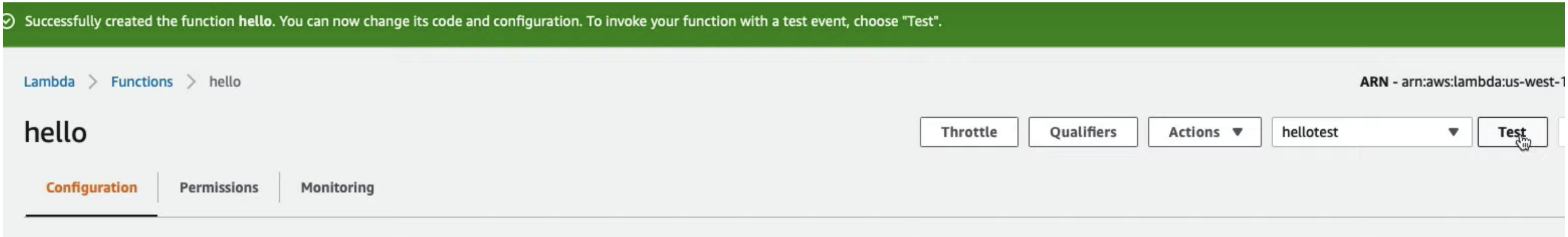
hello-world ▼

Event name

hellotest

```
{
  "key1": "value1",
  "key2": "value2",
  "key3": "value3"
}
```

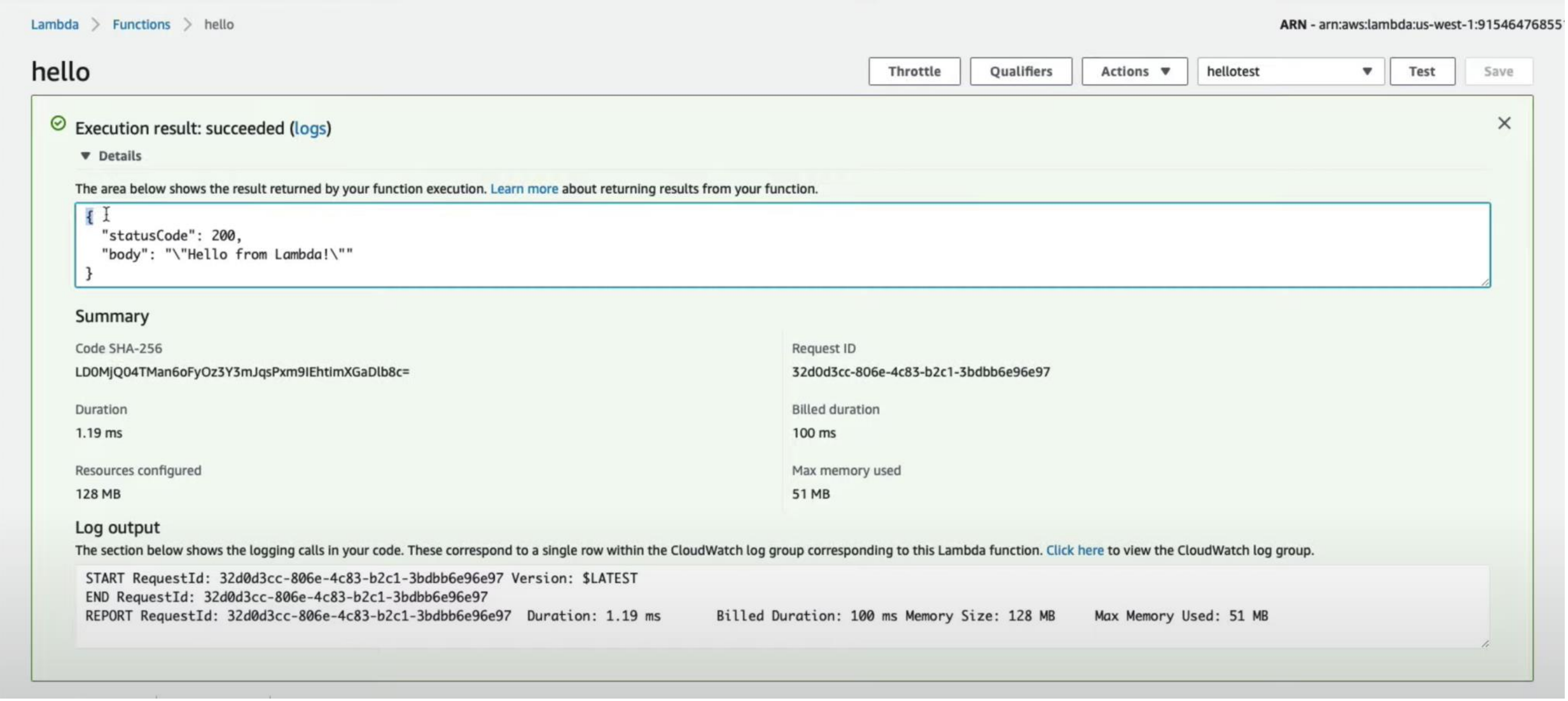
9. Click the **Test** button again while the **hellotest** event is pre-selected



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

10. Expand the **Execution result**

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


Lambda > Functions > hello
ARN - arn:aws:lambda:us-west-1:91546476855
hello
Throttle
Qualifiers
Actions ▼
hellotest ▼
Test
Save
✕
▼ Details

Execution result: succeeded (logs)

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
LD0MjQ04TMan6oFyOz3Y3mJqsPxm9IEhtimXGaDlb8c=

Request ID
32d0d3cc-806e-4c83-b2c1-3bdbb6e96e97

Duration
1.19 ms

Billed duration
100 ms

Resources configured
128 MB

Max memory used
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
```

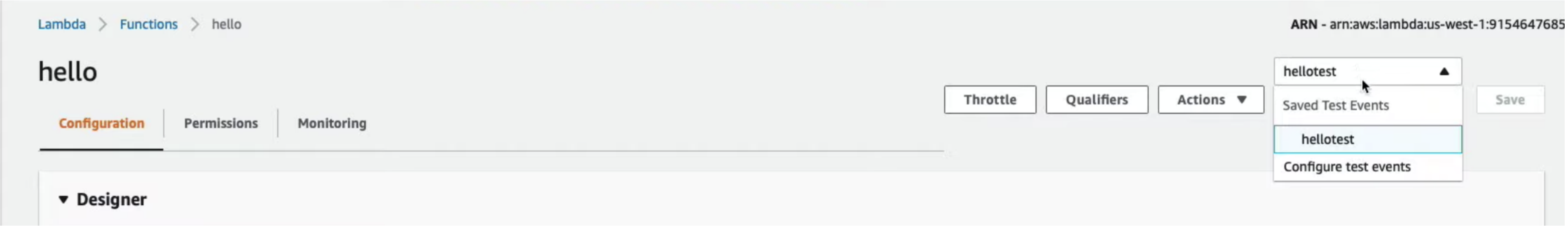
Exercise: Lambda Events

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**

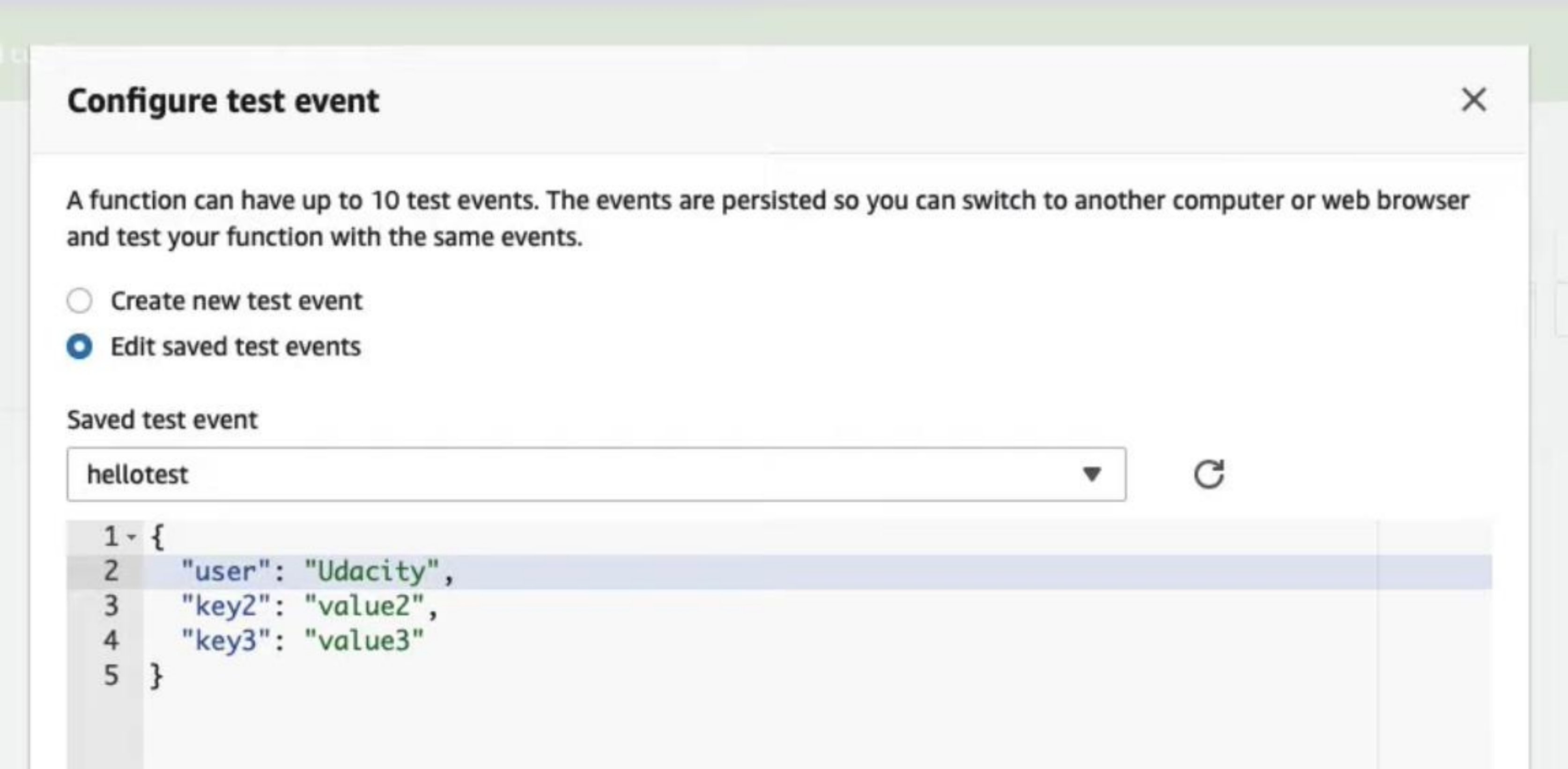

ARN - arn:aws:lambda:us-west-1:9154647685

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



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

Saved test event

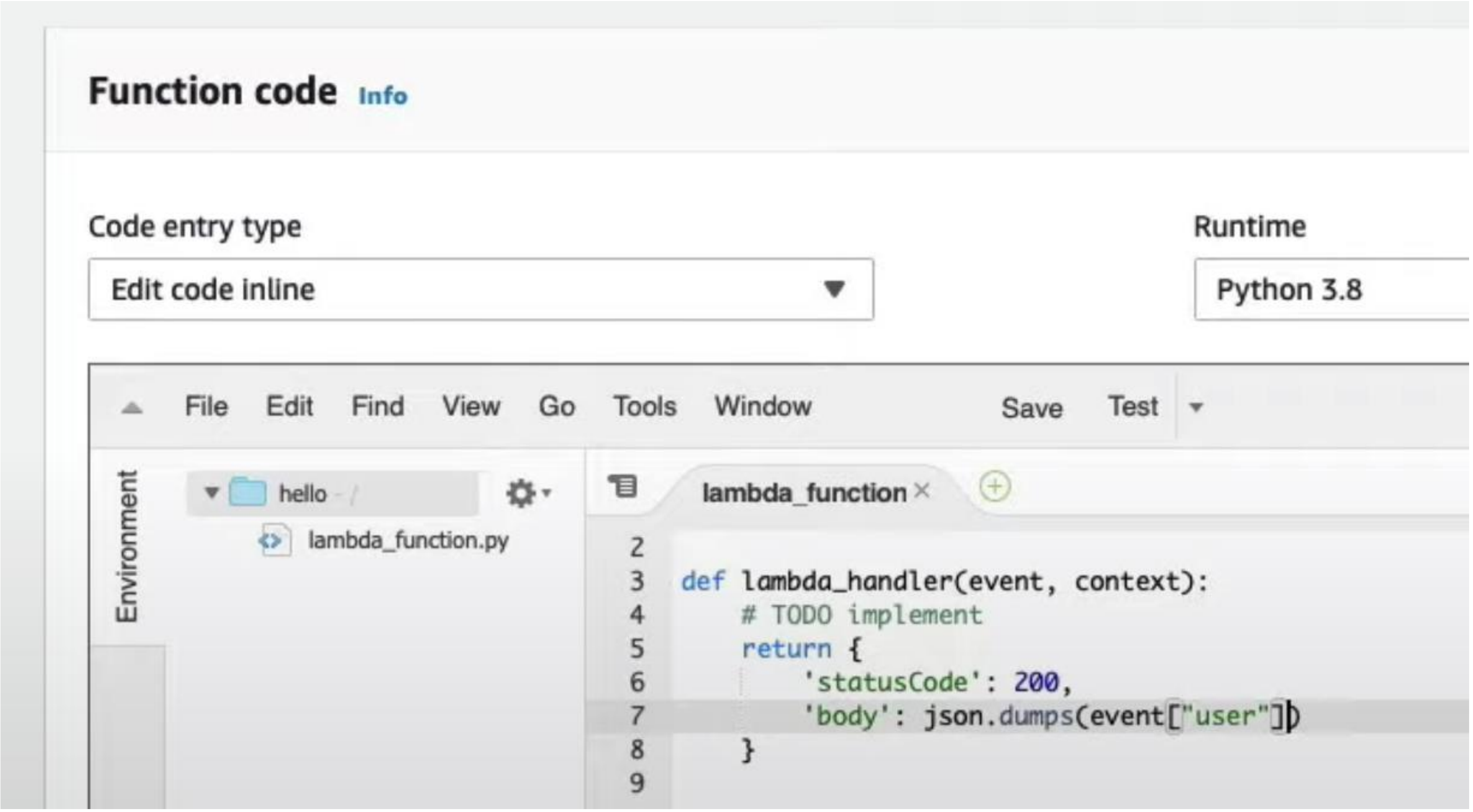
hellotest

```
{
  "user": "Udacity",
  "key2": "value2",
  "key3": "value3"
}
```

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

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

7. Click **Save**.



Function code

Code entry type: Edit code inline

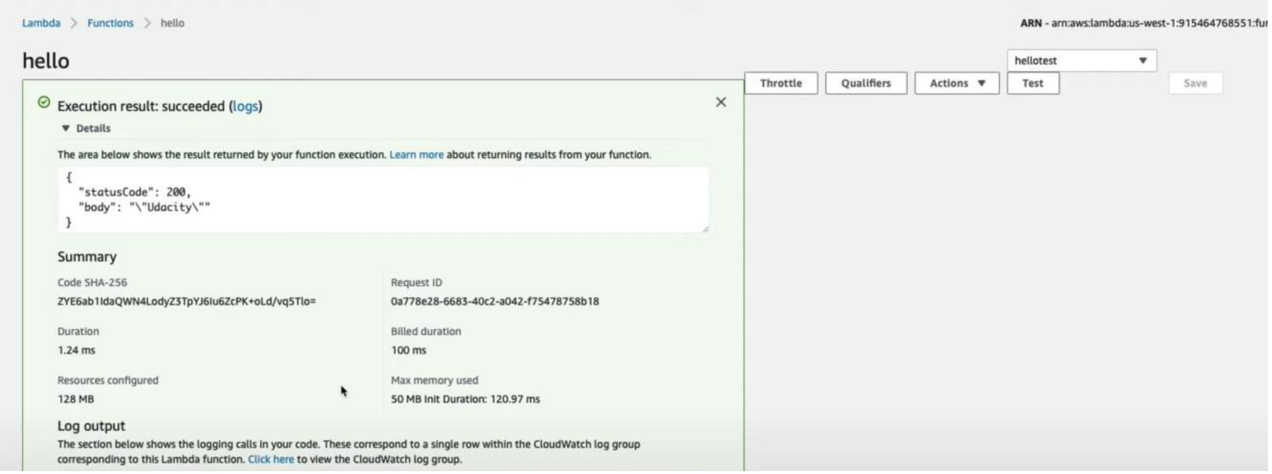
Runtime: Python 3.8

```
def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps(event["user"])
    }
```

8. Click the **Test** button.

9. Expand the results.

10. Verify message: **Udacity**



ARN - arn:aws:lambda:us-west-1:915464768551:fu

hello

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": "\"Udacity\""
}
```

Summary

Code SHA-256: ZYE6ab1IdaQWN4LodyZ3TpYJ6Iu6ZcPK+oLd/vq5Tlo=

Request ID: 0a778e28-6683-40c2-a042-f75478758b18

Duration: 1.24 ms

Billed duration: 100 ms

Resources configured: 128 MB

Max memory used: 50 MB Init Duration: 120.97 ms

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.

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
```