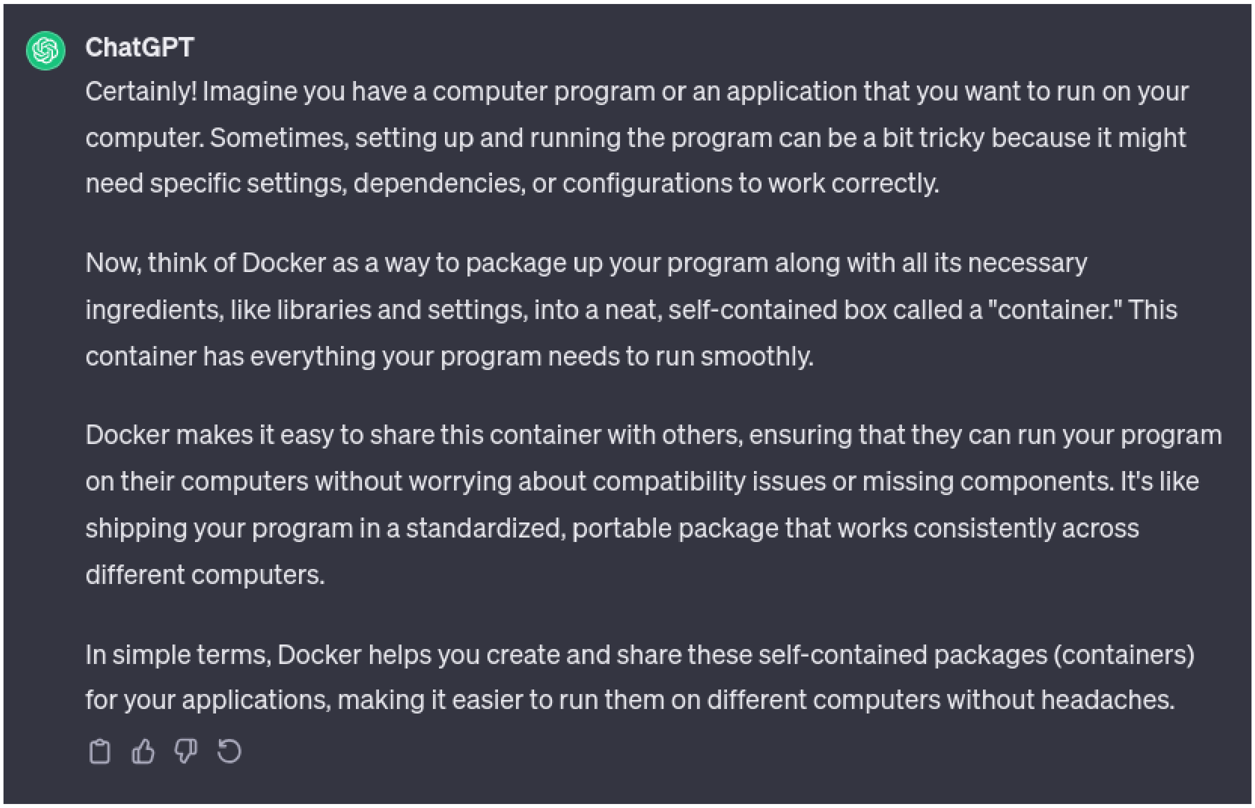
Other Compute Services

To cover

- ECS
- Lambda
- Batch
- Lightsail

Docker

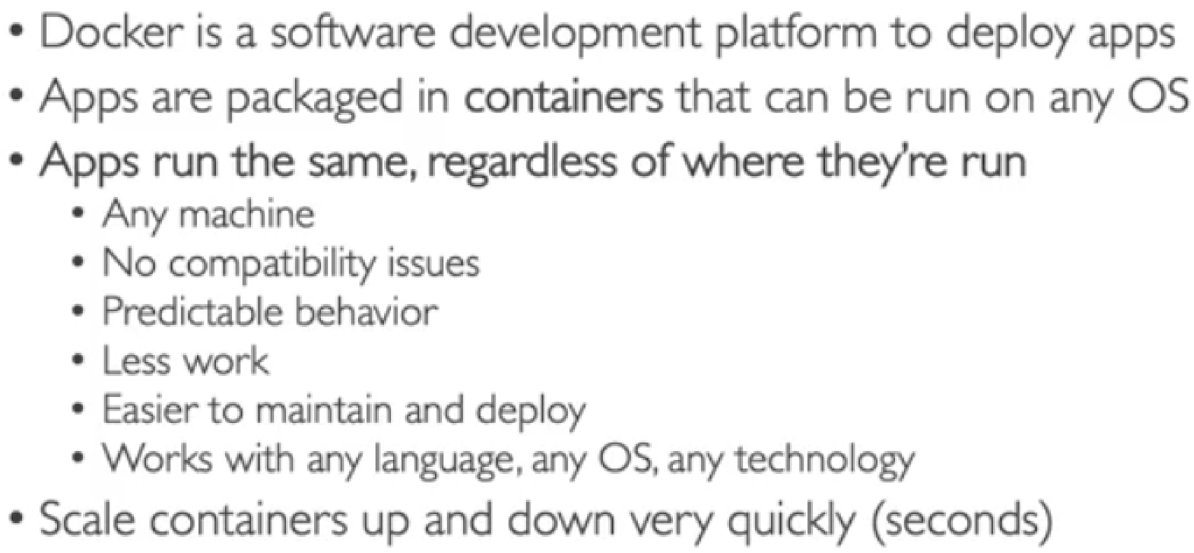


Certainly! Imagine you have a computer program or an application that you want to run on your computer. Sometimes, setting up and running the program can be a bit tricky because it might need specific settings, dependencies, or configurations to work correctly.

Now, think of Docker as a way to package up your program along with all its necessary ingredients, like libraries and settings, into a neat, self-contained box called a "container." This container has everything your program needs to run smoothly.

Docker makes it easy to share this container with others, ensuring that they can run your program on their computers without worrying about compatibility issues or missing components. It's like shipping your program in a standardized, portable package that works consistently across different computers.

In simple terms, Docker helps you create and share these self-contained packages (containers) for your applications, making it easier to run them on different computers without headaches.



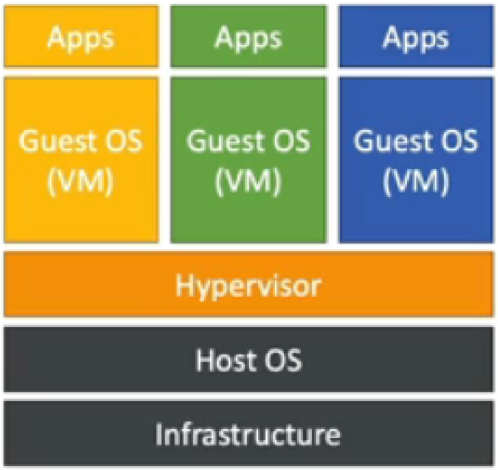
- Docker is a software development platform to deploy apps
- Apps are packaged in **containers** that can be run on any OS
- **Apps run the same, regardless of where they're run**
  - Any machine
  - No compatibility issues
  - Predictable behavior
  - Less work
  - Easier to maintain and deploy
  - Works with any language, any OS, any technology
- Scale containers up and down very quickly (seconds)

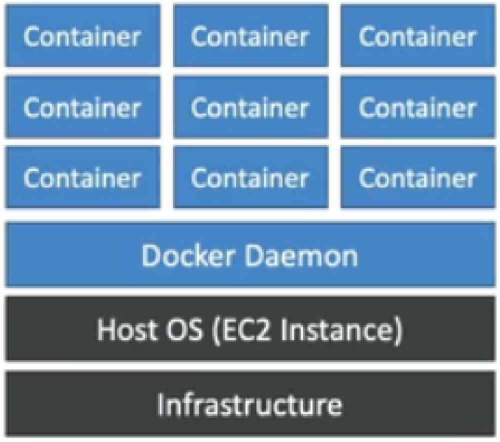
- The basic idea is, if we are able to containerize our application, it will become very easy for us to run it anywhere we like.
- These docker images or containers are stored in something called a docker repository.
- **Amazon has its own private docker repo called ECR (Elastic Container Registry)**
- AWS has a service called **ECS (Elastic Container Service)** for running containerized (generally said to run docker containers) applications or programs.
- Docker is not exactly a virtualization technology.

- **Docker VS Virtual Machines**

VMs	Docker
VMs have apps deployed on top of guest OS on top of the host machine's OS.	Docker containers are deployed on top of docker daemon and share the OS of the host OS.
	Docker containers share the kernel of the host operating system. This means they don't require a full operating system stack like

	virtual machines do. Containers are isolated from each other but share the underlying host OS kernel.
Virtual machines use hypervisors to create isolated environments with their own complete operating systems.	Docker uses containers, which are lightweight and share the same operating system (OS) kernel as the host machine.
VMs are more heavyweight than containers because each VM includes a full OS, which consumes more resources.	Containers package an application and its dependencies together, making them portable and consistent across different environments.





ECS (Elastic Container Service)

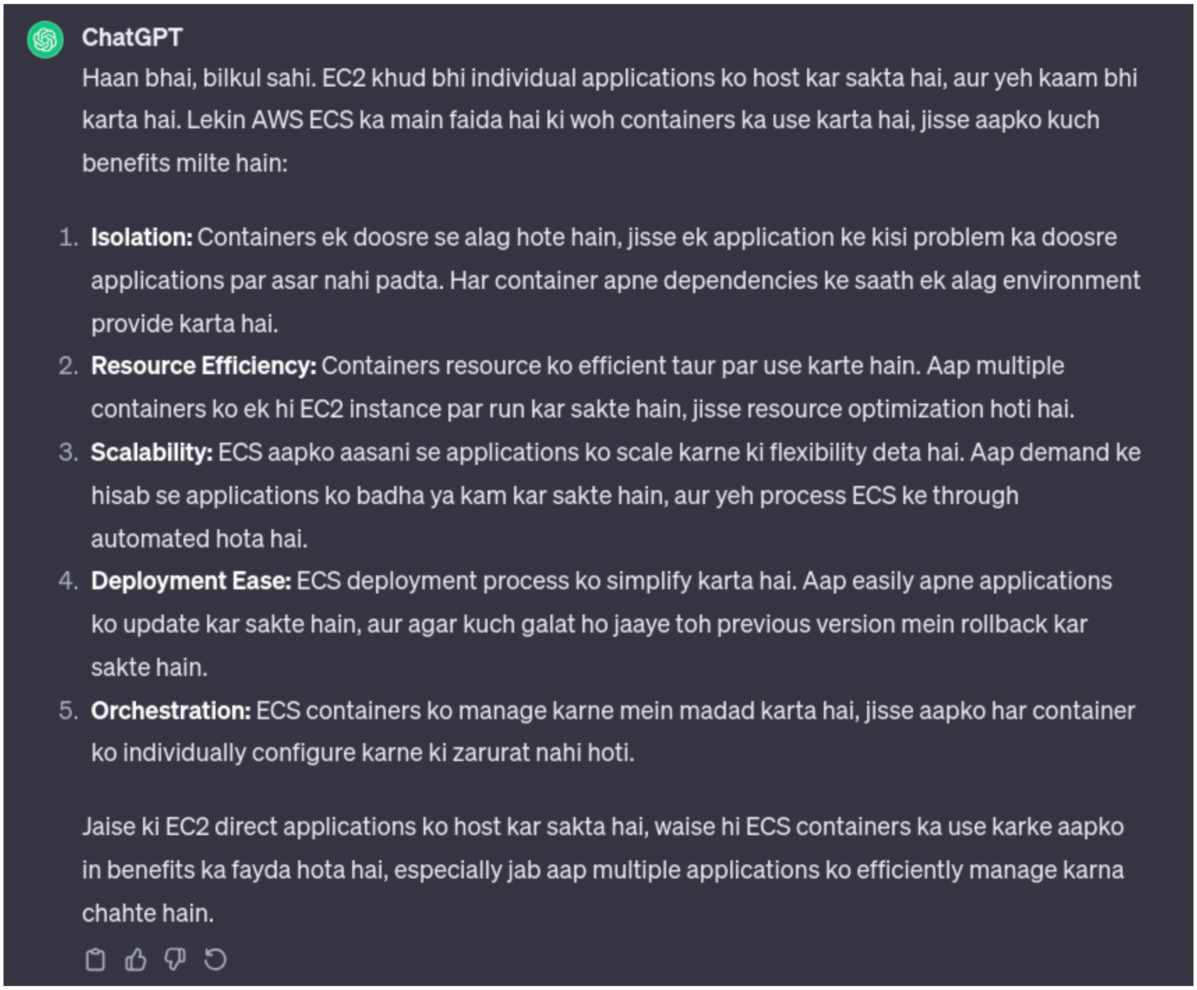
- Deploying and running your usual application on aws is done directly through ec2 instances. But if you want to run containerized applications then for that you have the ECS service, which btw, may or may not use ec2 instances as infrastructure or server.

DOCKER???

- ECS is a managed service by aws to run containers (docker containers generally).

- Now, to run these containers you need some infrastructure for which you got two options:
    - serverless (with fargate) = much easier to use as you don't have to manage the underlying infrastructure
    - ec2 instance servers = you are responsible for managing the provisioned instances - extra work

- **If you choose to use ec2 instances as servers:**
- Why is ECS even needed tho?



ChatGPT

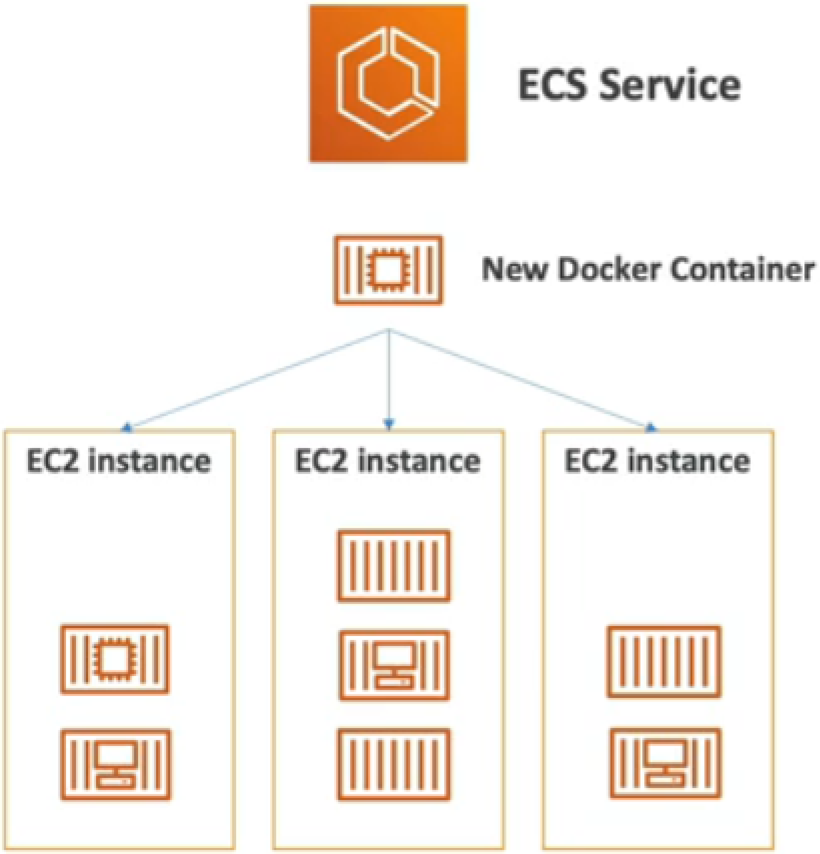
Haan bhai, bilkul sahi. EC2 khud bhi individual applications ko host kar sakta hai, aur yeh kaam bhi karta hai. Lekin AWS ECS ka main faida hai ki woh containers ka use karta hai, jisse aapko kuch benefits milte hain:

1. **Isolation:** Containers ek doosre se alag hote hain, jisse ek application ke kisi problem ka doosre applications par asar nahi padta. Har container apne dependencies ke saath ek alag environment provide karta hai.
2. **Resource Efficiency:** Containers resource ko efficient taur par use karte hain. Aap multiple containers ko ek hi EC2 instance par run kar sakte hain, jisse resource optimization hoti hai.
3. **Scalability:** ECS aapko aasani se applications ko scale karne ki flexibility deta hai. Aap demand ke hisab se applications ko badha ya kam kar sakte hain, aur yeh process ECS ke through automated hota hai.
4. **Deployment Ease:** ECS deployment process ko simplify karta hai. Aap easily apne applications ko update kar sakte hain, aur agar kuch galat ho jaaye toh previous version mein rollback kar sakte hain.
5. **Orchestration:** ECS containers ko manage karne mein madad karta hai, jisse aapko har container ko individually configure karne ki zarurat nahi hoti.

Jaise ki EC2 direct applications ko host kar sakta hai, waise hi ECS containers ka use karke aapko in benefits ka fayda hota hai, especially jab aap multiple applications ko efficiently manage karna chahte hain.

- Running docker containers on aws simply means ECS service.

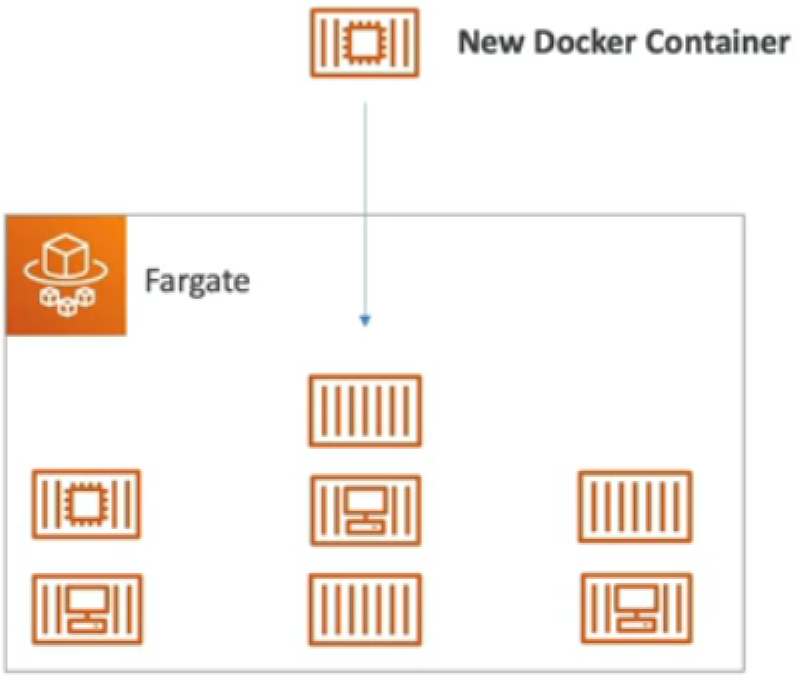
- ECS lets you run multiple different containers (multiple applications) on the same ec2 instance that too with effective resource utilization.



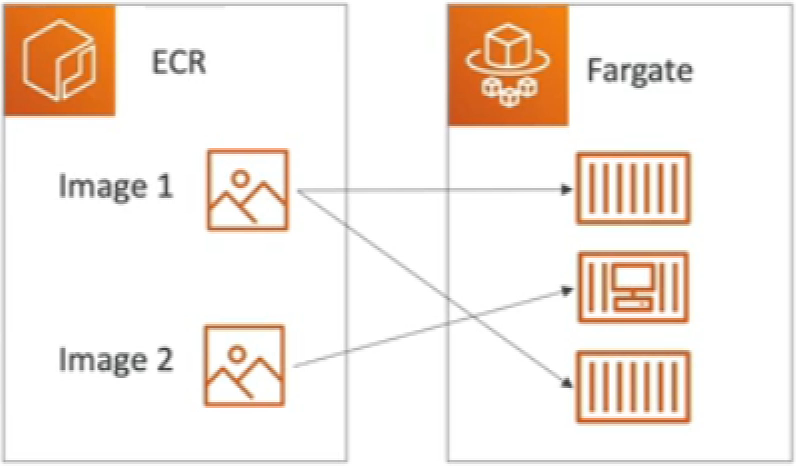
- Whenever a new container enters the scene, ECS places it onto an ec2 instance and it is smart enough to know which one to put it on for effective resource utilization and load balancing.

- **If you choose to go serverless (fargate):**
- Fargate also lets you launch docker containers on AWS, except without having to provision infrastructure.
- It is serverless and hence super easy to manage.
- In this case, AWS will simply run containers for you. In this case you won't need ec2 instances as servers. You can simply ask aws to run your containers

using the fargate service and it will do so based on the resources (CPU / RAM) required by particular containers.



- ECR (Elastic Container Registry)
- ECR is aws' private docker registry which lets you store docker images from which fargate can create containers and run them.



What is “serverless”?

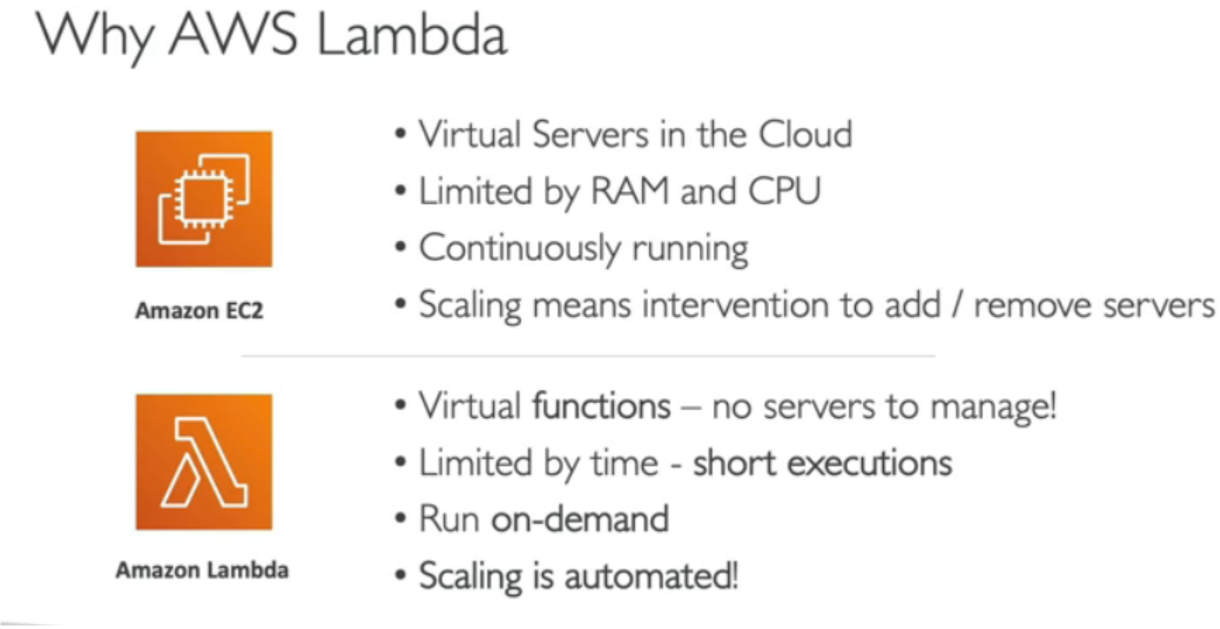
Serverless means developers don’t have to manage the servers anymore. Businesses don’t need to maintain the servers anymore. Everyone can focus on whatever is more important or helpful for the business instead of managing the damn servers.

Serverless does not mean that there are no servers. Servers are there obviously. Hawa me thodi chalega application. But you don’t have to manage or provision or even see them.

Some serverless services:

- **Lambda** was the first serverless service by aws. This service was where the idea of serverless came from.
- It was initially called as FaaS (function as a service) bec it lets you run functions on the cloud.

Lambda



Why AWS Lambda

Amazon EC2

- Virtual Servers in the Cloud
- Limited by RAM and CPU
- Continuously running
- Scaling means intervention to add / remove servers

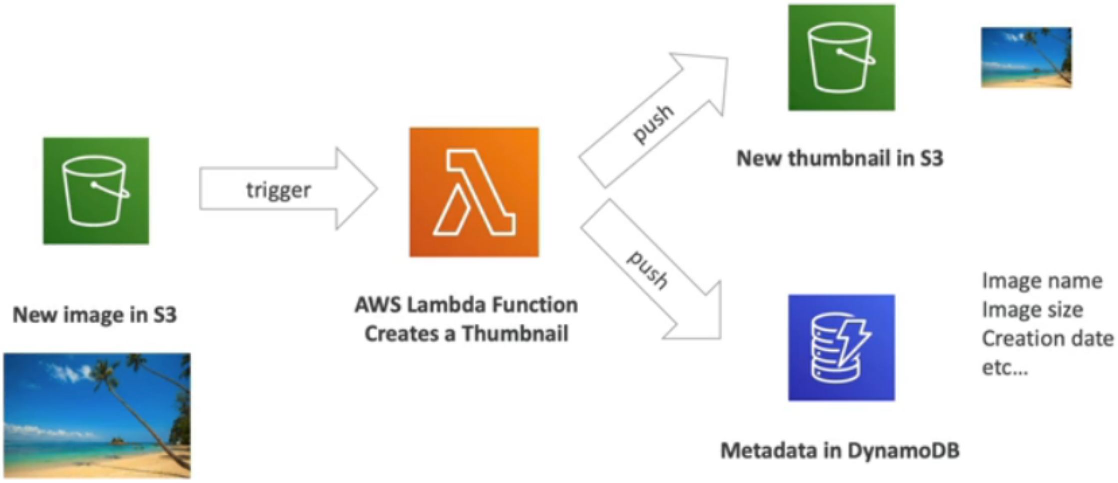
Amazon Lambda

- Virtual **functions** – no servers to manage!
- Limited by time - **short executions**
- Run **on-demand**
- **Scaling is automated!**

- **Lambda is a reactive type of service**. This means that it is **event-driven**. Basically, the functions are invoked or called only when something happens and when needed.
- Pricing model is easier and pretty generous
    - pay per request and compute time for processing of that request
    - 1000000 lambda req allowed for free tier usage per month
- It can be integrated with all the aws services and even many programming languages
- Can be monitored through CloudWatch.
- SUPER EASY to increase resources (up to 10GB of RAM) per function and as you increase resources, the cpu and network performance also improve.
- **Lambda Container Image Service** - Lambda also allows you to run docker containers but this is possible only if the container image has implemented the Lambda Runtime API. (some very specific docker images)

- Serverless Thumbnail creation example:

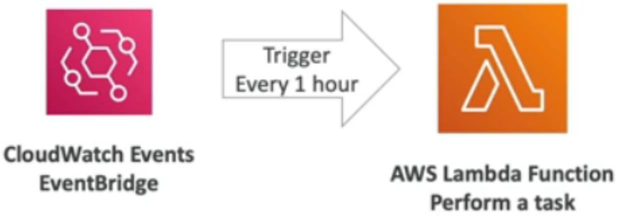
  This serverless thumbnail creation process will scale very well as we haven't provisioned servers at any point (S3, Lambda, DynamoDB are all serverless).



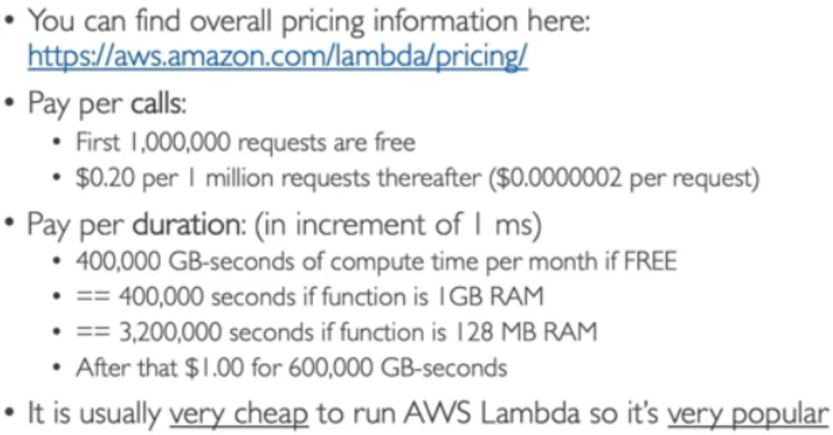
- Serverless CRON Job example:

  Generally, CRON Jobs are scheduled on a UNIX or Linux machine. But, since we need to go serverless here, we can use CloudWatch Events EventBridge service that will trigger a lambda function after every x hours or minutes to perform a job.

  So, effectively we do not need to provision any servers to get a job done.



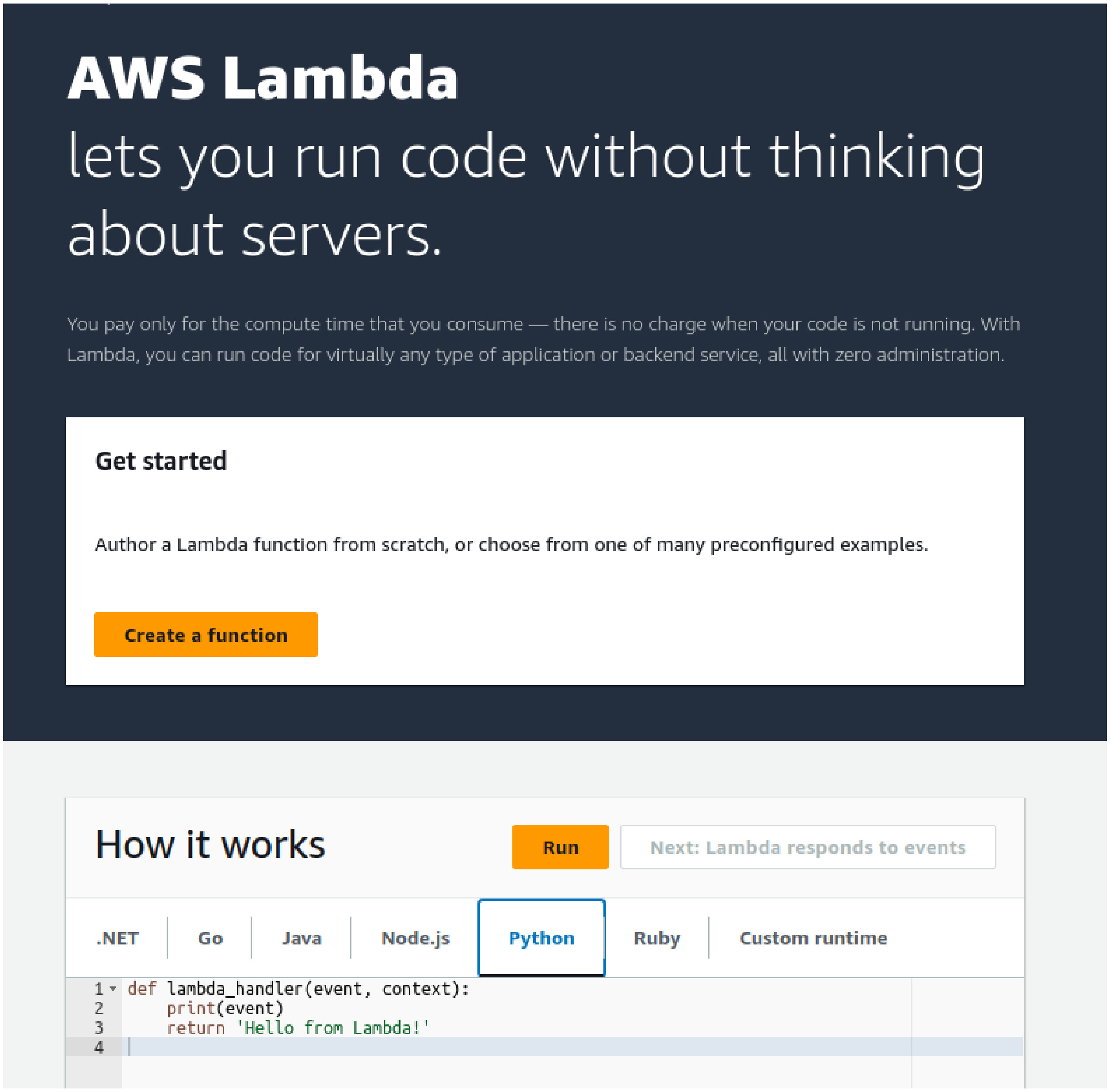
- **AWS Lambda pricing** is based on calls (the number of times you call or trigger a function) and duration (the duration for which the function was run). Lambda is consider very cheap.



- You can find overall pricing information here: https://aws.amazon.com/lambda/pricing/
- Pay per **calls**:
  - First 1,000,000 requests are free
  - $0.20 per 1 million requests thereafter ($0.0000002 per request)
- Pay per **duration**: (in increment of 1 ms)
  - 400,000 GB-seconds of compute time per month if FREE
  - == 400,000 seconds if function is 1GB RAM
  - == 3,200,000 seconds if function is 128 MB RAM
  - After that $1.00 for 600,000 GB-seconds
- It is usually very cheap to run AWS Lambda so it's very popular

Lambda Hands-on

- Select Lambda from the console. Then, replace ‘discover’ by ‘begin’ at the end of the URL.



AWS Lambda

lets you run code without thinking about servers.

You pay only for the compute time that you consume — there is no charge when your code is not running. With Lambda, you can run code for virtually any type of application or backend service, all with zero administration.

Get started

Author a Lambda function from scratch, or choose from one of many preconfigured examples.

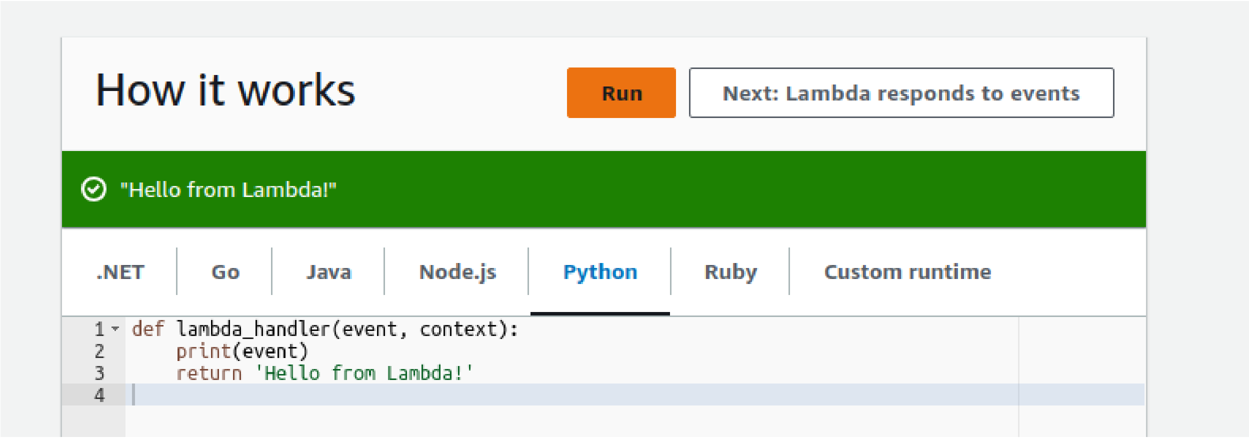
How it works

```python
def lambda_handler(event, context):
    print(event)
    return 'Hello from Lambda!'

```

- Now, you just need to write the code for a function in a language of your choice and Lambda will run it for you.

- Let’s try with python:



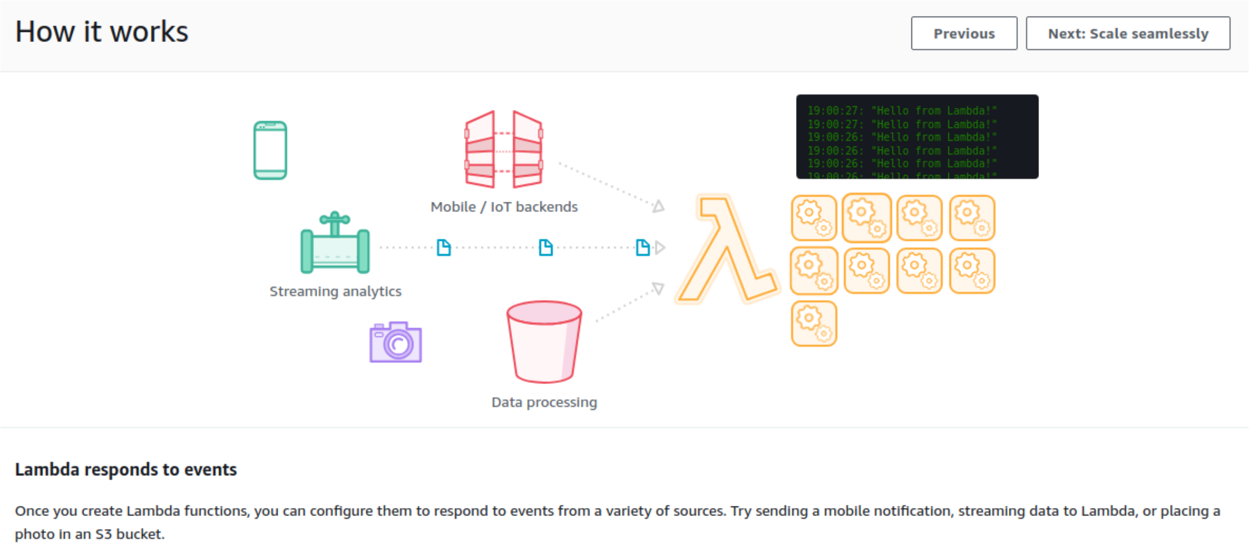
How it works

"Hello from Lambda!"

```python
def lambda_handler(event, context):
    print(event)
    return 'Hello from Lambda!'

```

- To learn more on how it works:

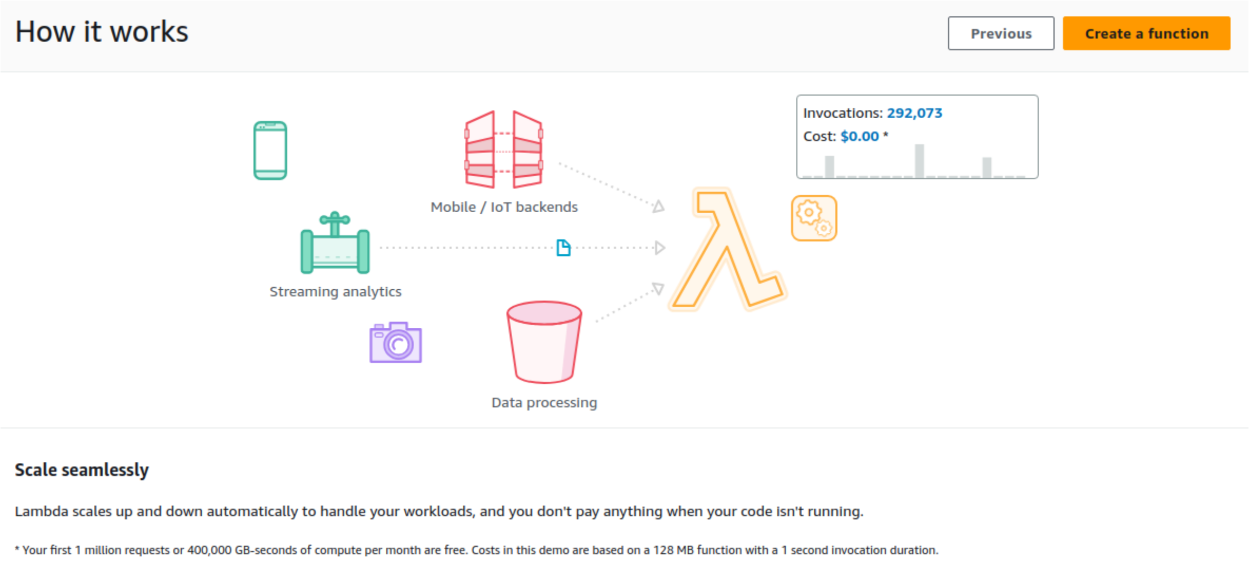


How it works

```
19:00:27: "Hello from Lambda!"
19:00:27: "Hello from Lambda!"
19:00:26: "Hello from Lambda!"
19:00:26: "Hello from Lambda!"
19:00:26: "Hello from Lambda!"
19:00:26: "Hello from Lambda!"
```

Lambda responds to events

Once you create Lambda functions, you can configure them to respond to events from a variety of sources. Try sending a mobile notification, streaming data to Lambda, or placing a photo in an S3 bucket.



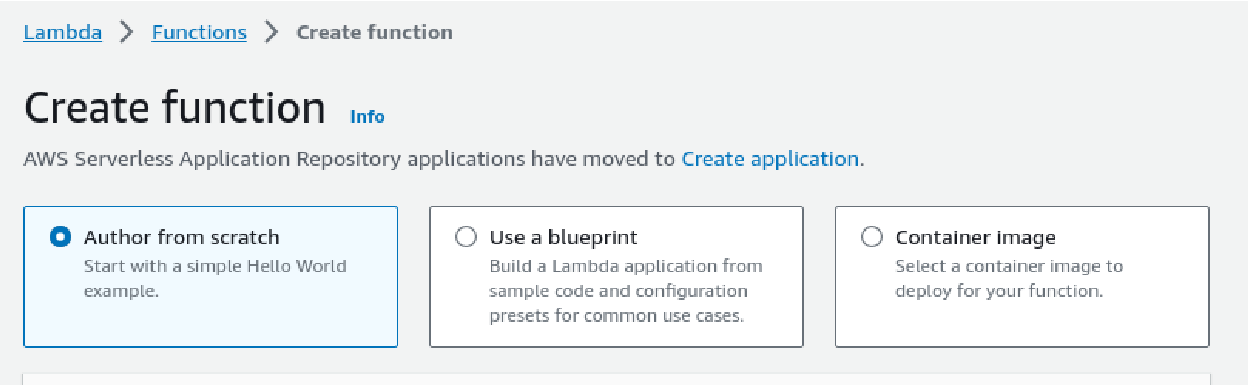
Scale seamlessly

Lambda scales up and down automatically to handle your workloads, and you don't pay anything when your code isn't running.

* Your first 1 million requests or 400,000 GB-seconds of compute per month are free. Costs in this demo are based on a 128 MB function with a 1 second invocation duration.

- Now, we create a function

You will get 3 options to choose from, each with varying customizability. For now, we will go with the blueprint option for ease.



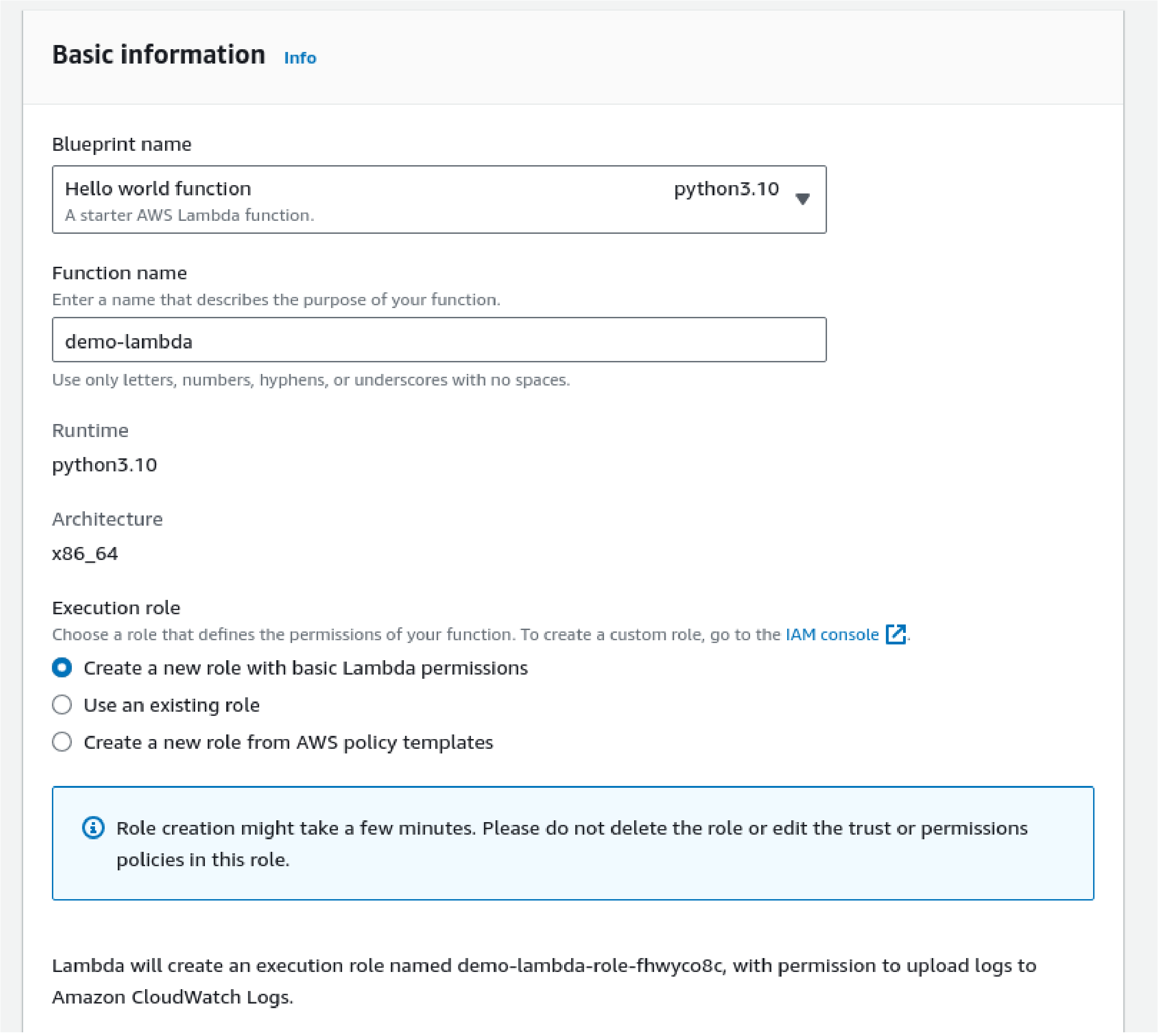
Create function Info

AWS Serverless Application Repository applications have moved to Create application.

● 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.

○ Container image
Select a container image to deploy for your function.



Basic information Info

Blueprint name

Hello world function python3.10
A starter AWS Lambda function.

Function name

Enter a name that describes the purpose of your function.

demo-lambda

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

Runtime

python3.10

Architecture

x86_64

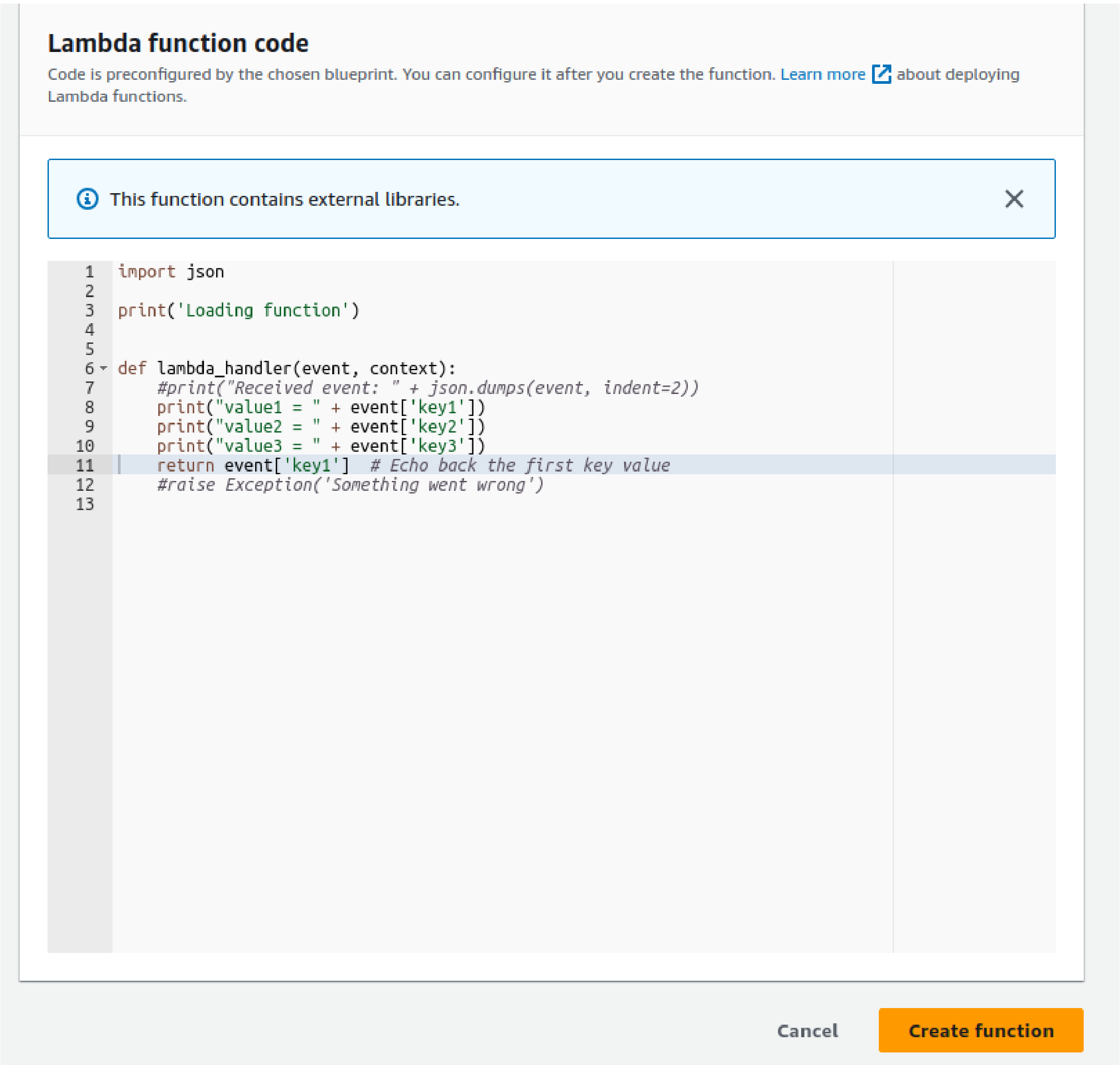
Execution role

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.

Lambda will create an execution role named demo-lambda-role-fhwyco8c, with permission to upload logs to Amazon CloudWatch Logs.



Lambda function code

Code is preconfigured by the chosen blueprint. You can configure it after you create the function. Learn more about deploying Lambda functions.

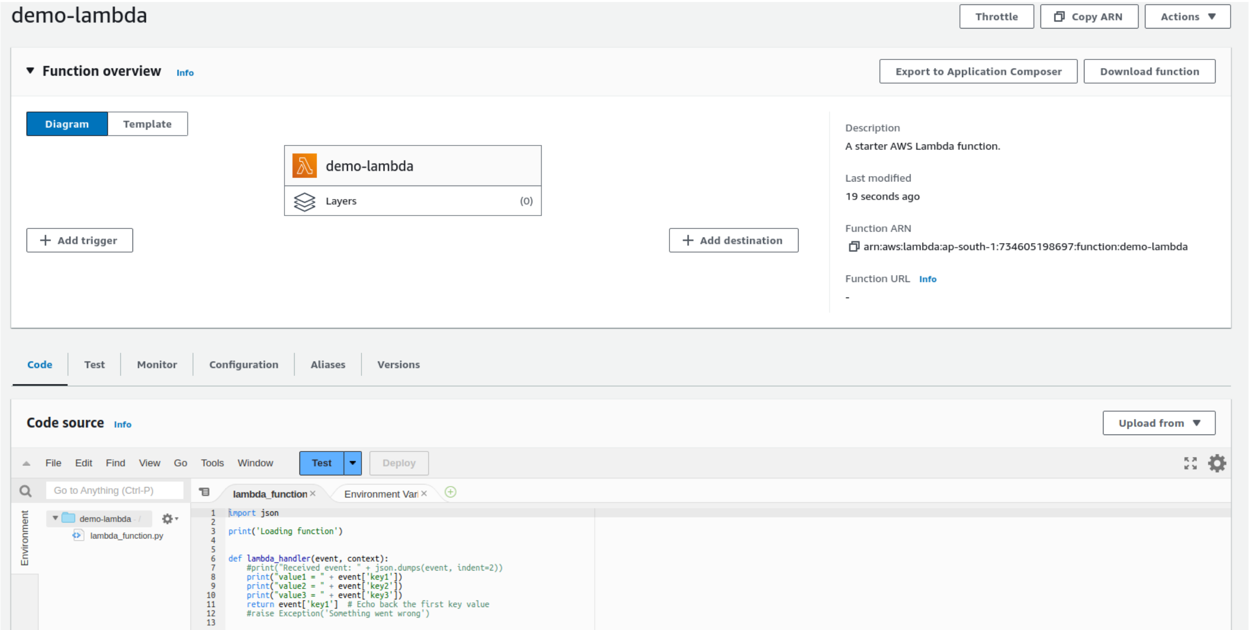
This function contains external libraries.

```
import json

print('Loading function')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    print("value1 = " + event['key1'])
    print("value2 = " + event['key2'])
    print("value3 = " + event['key3'])
    return event['key1']  # Echo back the first key value
    #raise Exception('Something went wrong')

```



Description
A starter AWS Lambda function.

Last modified
19 seconds ago

Function ARN
arn:aws:lambda:ap-south-1:734605198697:function:demo-lambda

Function URL
-

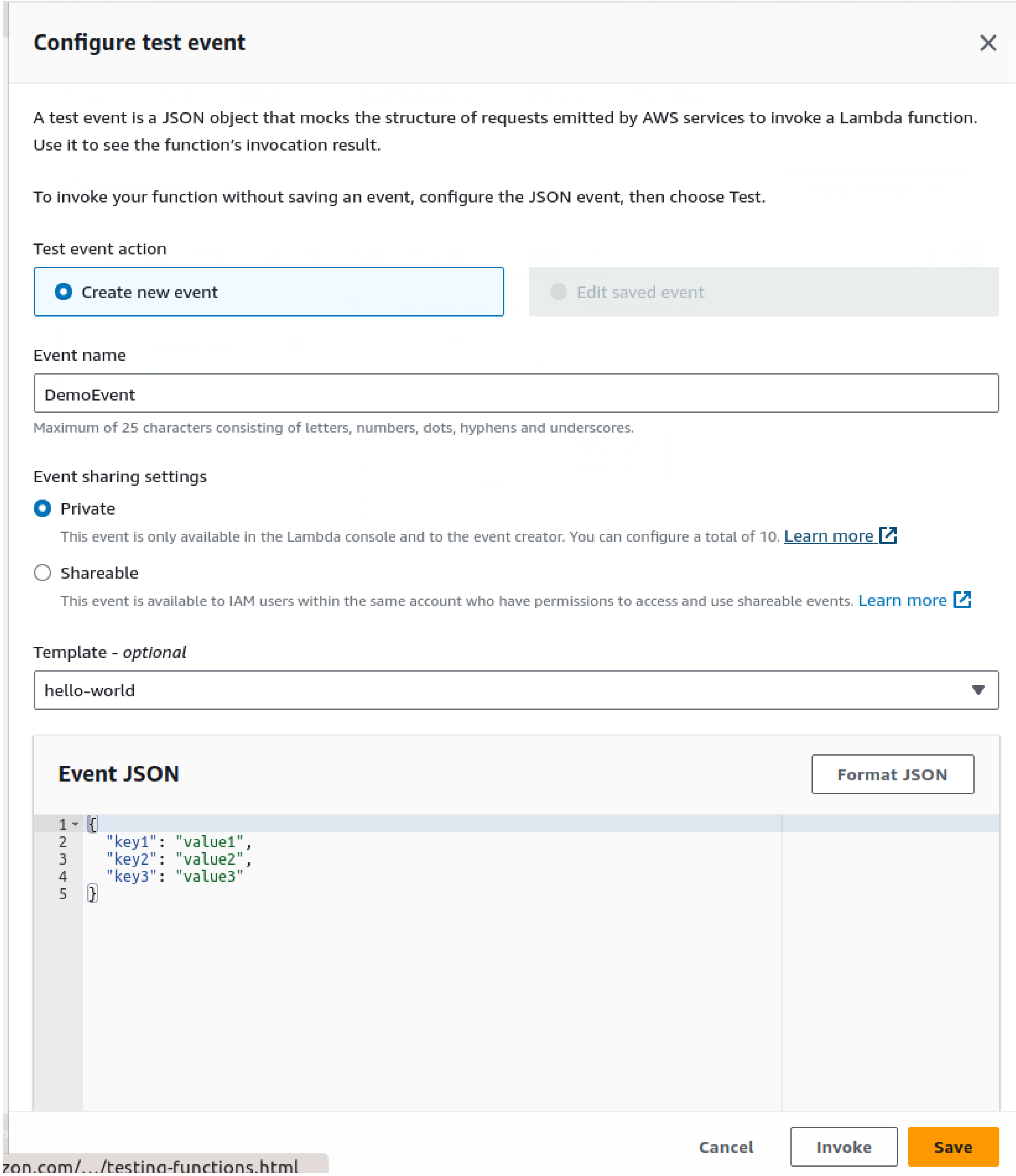
```python
import json

print('Loading function')


def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    print("value1 = " + event['key1'])
    print("value2 = " + event['key2'])
    print("value3 = " + event['key3'])
    return event['key1']  # Echo back the first key value
    #raise Exception('Something went wrong')

```

- Let’s now test the function



Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

● Create new event ○ Edit saved event

Event name

DemoEvent

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

● Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

○ Shareable

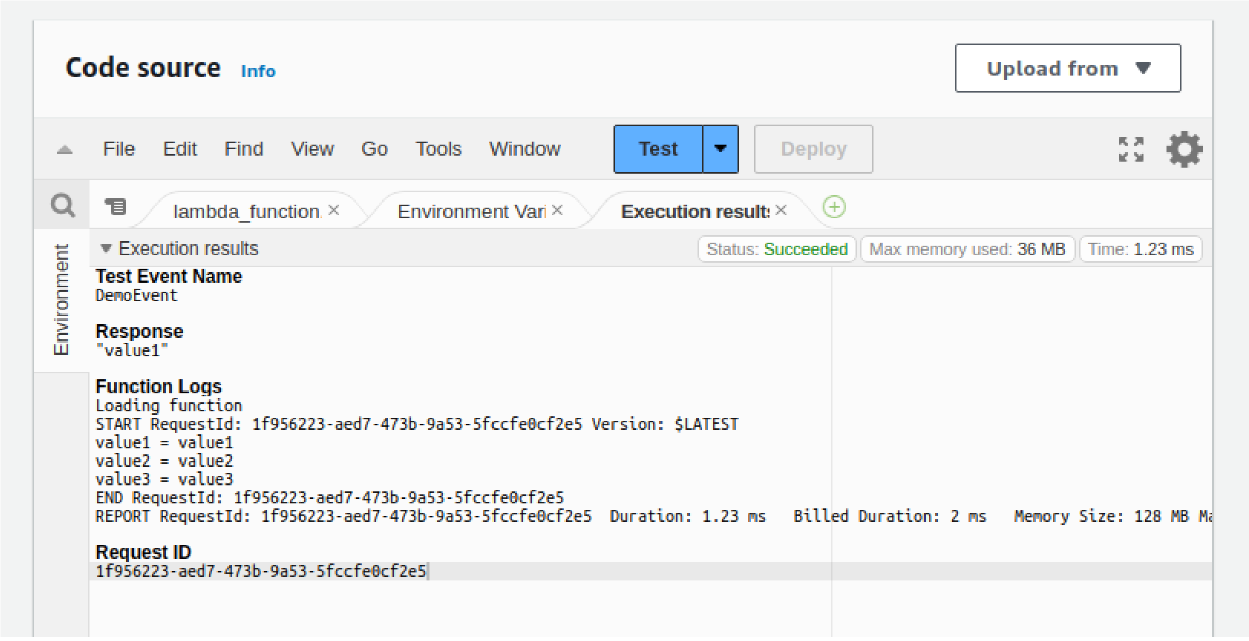
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

hello-world

Event JSON

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



Status: Succeeded Max memory used: 36 MB Time: 1.23 ms

Test Event Name

```
DemoEvent
```

Response

```
"value1"
```

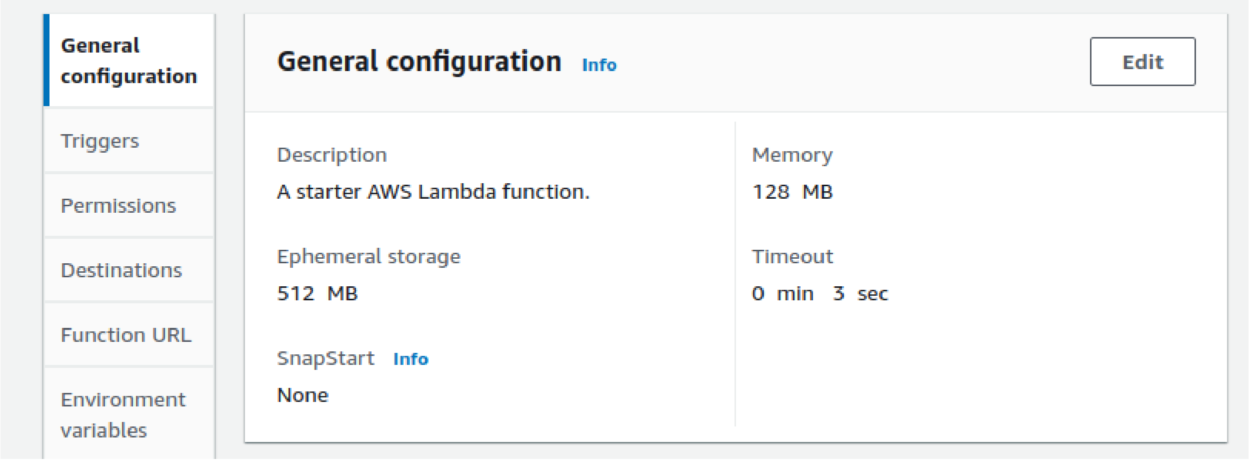
Function Logs

```
Loading function
START RequestId: 1f956223-aed7-473b-9a53-5fccfe0cf2e5 Version: $LATEST
value1 = value1
value2 = value2
value3 = value3
END RequestId: 1f956223-aed7-473b-9a53-5fccfe0cf2e5
REPORT RequestId: 1f956223-aed7-473b-9a53-5fccfe0cf2e5  Duration: 1.23 ms   Billed Duration: 2 ms   Memory Size: 128 MB Ma
```

Request ID

```
1f956223-aed7-473b-9a53-5fccfe0cf2e5
```

- Let us now configure this function. You can play around with these settings and edit them to whatever fits your needs the best.



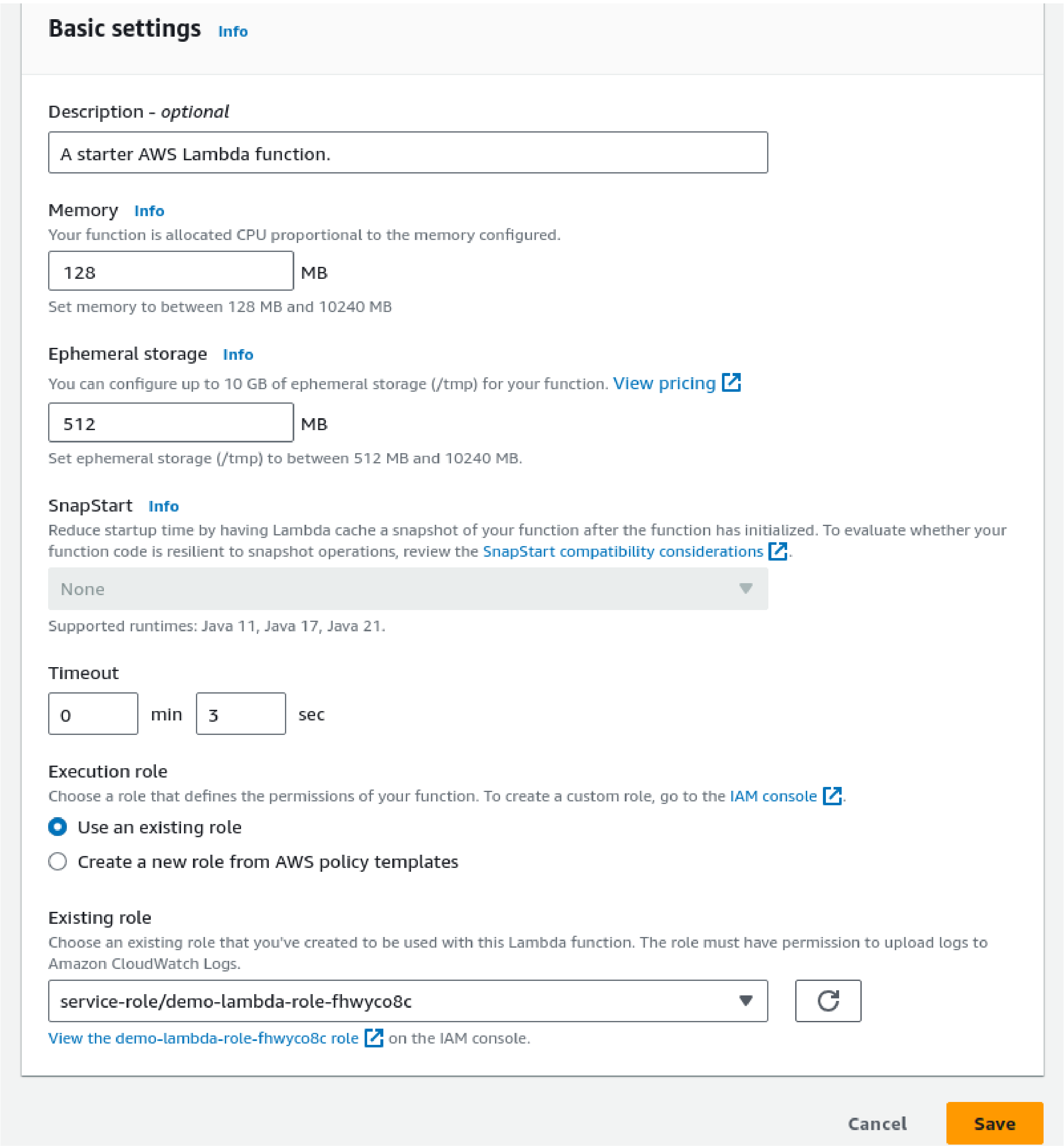
Description: A starter AWS Lambda function.

Memory: 128 MB

Ephemeral storage: 512 MB

Timeout: 0 min 3 sec

SnapStart: None



Basic settings Info

Description - *optional*

A starter AWS Lambda function.

Memory Info

Your function is allocated CPU proportional to the memory configured.

128 MB

Set memory to between 128 MB and 10240 MB

Ephemeral storage Info

You can configure up to 10 GB of ephemeral storage (/tmp) for your function. View pricing

512 MB

Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart Info

Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the SnapStart compatibility considerations.

None

Supported runtimes: Java 11, Java 17, Java 21.

Timeout

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

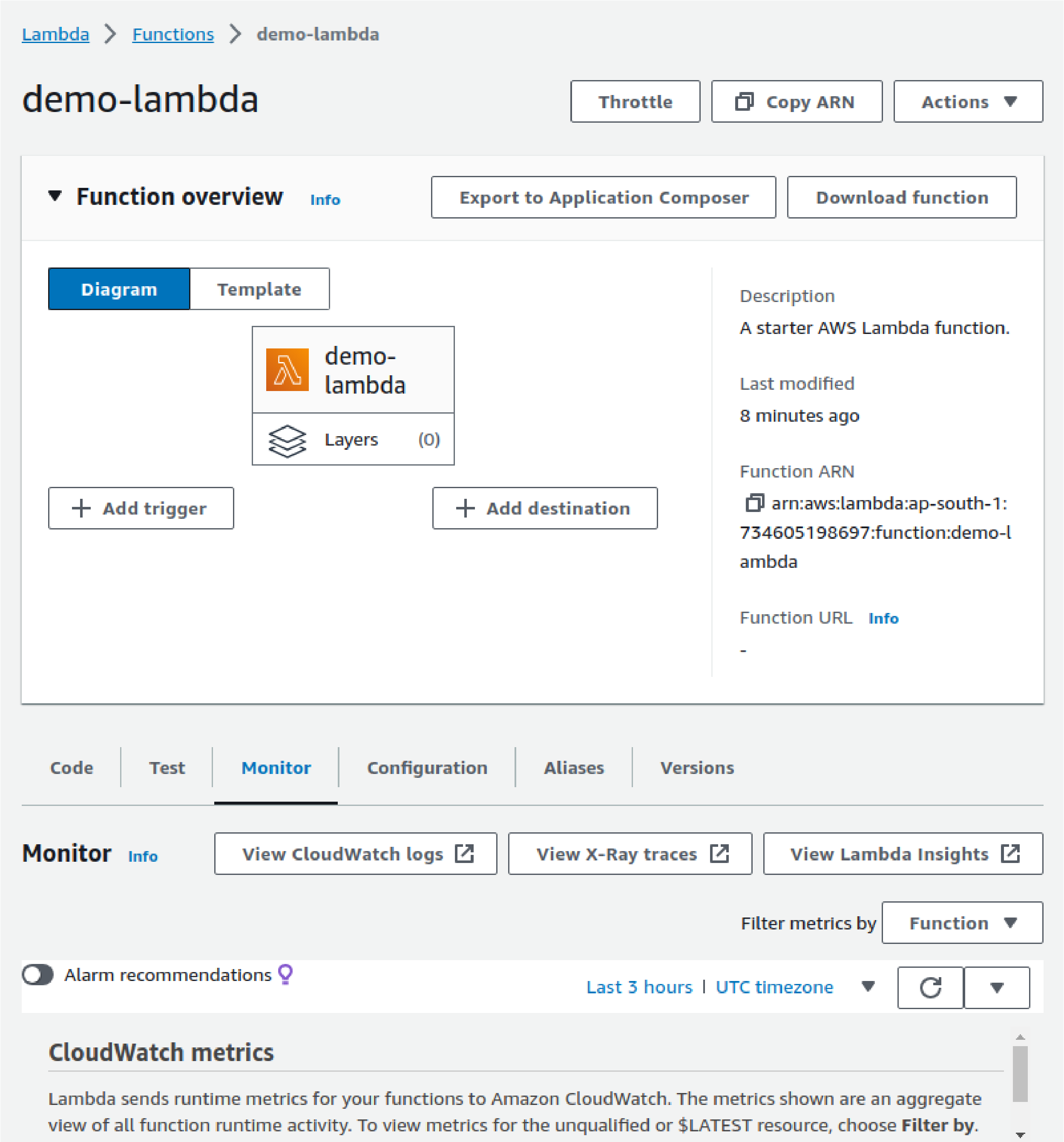
Existing role

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.

service-role/demo-lambda-role-fhwyco8c

View the demo-lambda-role-fhwyco8c role on the IAM console.

- You can also easily monitor your Lambda usage using CloudWatch



Description
A starter AWS Lambda function.

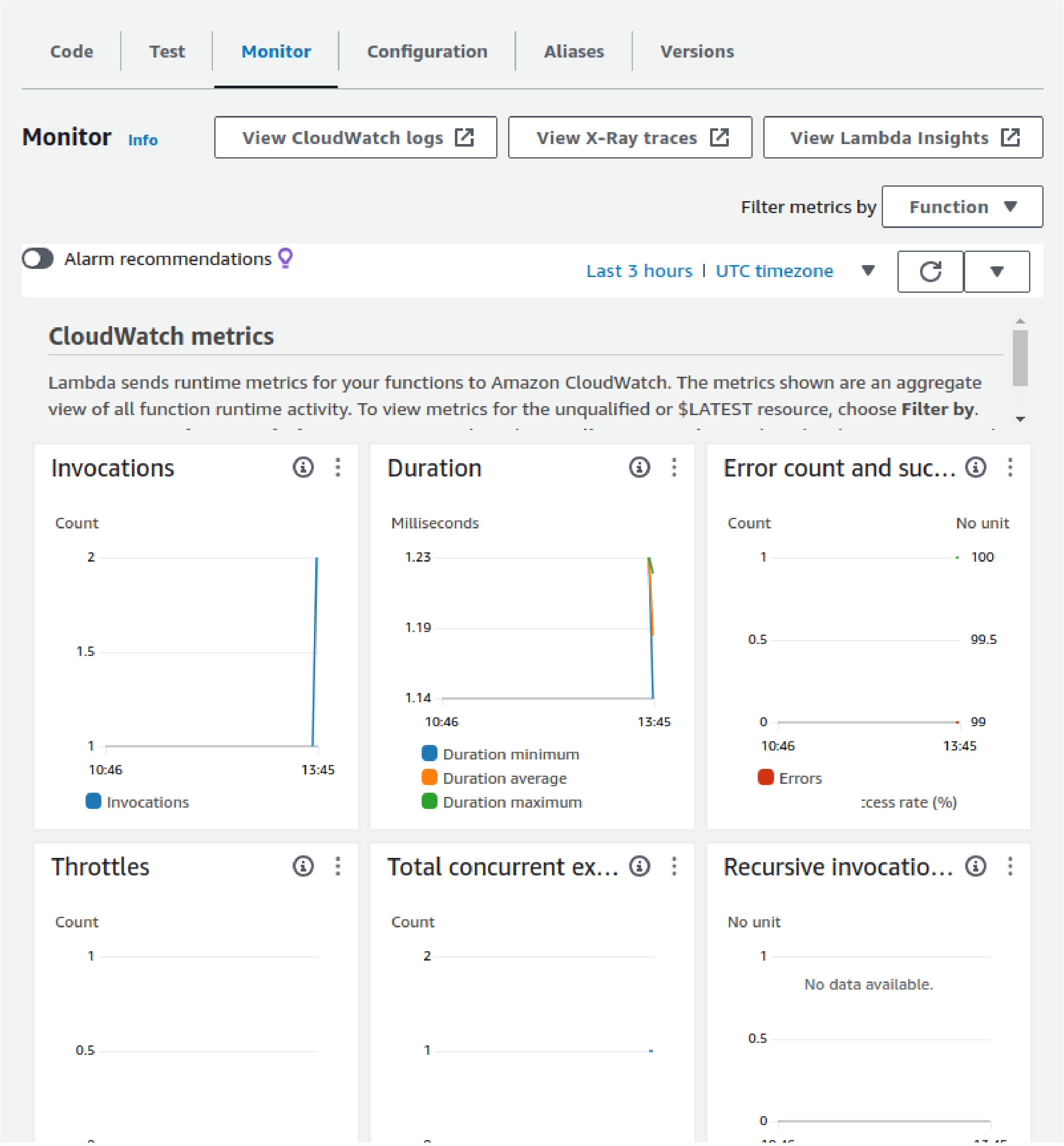
Last modified
8 minutes ago

Function ARN
arn:aws:lambda:ap-south-1:734605198697:function:demo-lambda

Function URL Info
-

Filter metrics by Function

Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose **Filter by**.

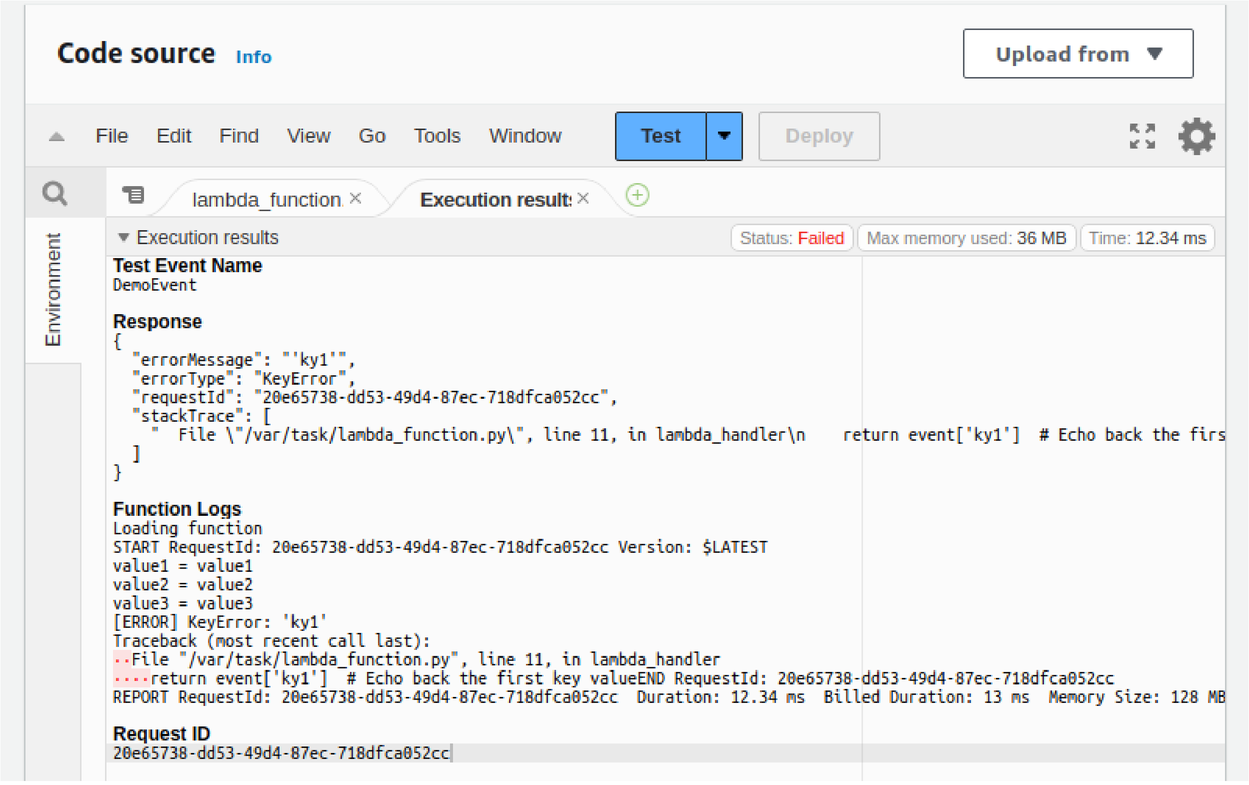


CloudWatch metrics

Lambda sends runtime metrics for your functions to Amazon CloudWatch. The metrics shown are an aggregate view of all function runtime activity. To view metrics for the unqualified or $LATEST resource, choose **Filter by**.

No data available.

- Also, if you make any changes to your code, you will need to first deploy those changes then you can test it again. You can also monitor any errors.



Status: Failed | Max memory used: 36 MB | Time: 12.34 ms

Test Event Name

```
DemoEvent
```

Response

```
{
  "errorMessage": "'ky1'",
  "errorType": "KeyError",
  "requestId": "20e65738-dd53-49d4-87ec-718dfca052cc",
  "stackTrace": [
    "  File \"/var/task/lambda_function.py\", line 11, in lambda_handler\n    return event['ky1']  # Echo back the firs
  ]
}
```

Function Logs

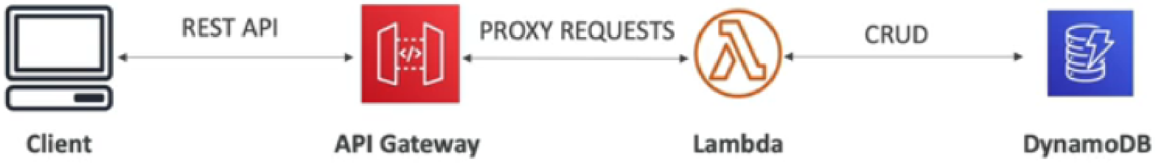
```
Loading function
START RequestId: 20e65738-dd53-49d4-87ec-718dfca052cc Version: $LATEST
value1 = value1
value2 = value2
value3 = value3
[ERROR] KeyError: 'ky1'
Traceback (most recent call last):
  File "/var/task/lambda_function.py", line 11, in lambda_handler
    return event['ky1']  # Echo back the first key valueEND RequestId: 20e65738-dd53-49d4-87ec-718dfca052cc
REPORT RequestId: 20e65738-dd53-49d4-87ec-718dfca052cc  Duration: 12.34 ms  Billed Duration: 13 ms  Memory Size: 128 MB
```

Request ID

```
20e65738-dd53-49d4-87ec-718dfca052cc
```

API Gateway

- It is a fully managed serverless service for developers to easily create, publish, maintain and monitor APIs.
- Suppose you want to create a serverless HTTP API using Lambda functions and DynamoDB as your database.
- Now, you want your clients to be able to access that API. But, lambda functions are not directly exposed to the client. Which is why we need an API Gateway.
- The API Gateway is going to provide the client with the rest HTTP API to connect directly to your website.
- It supports RESTful APIs and WebSocket APIs.



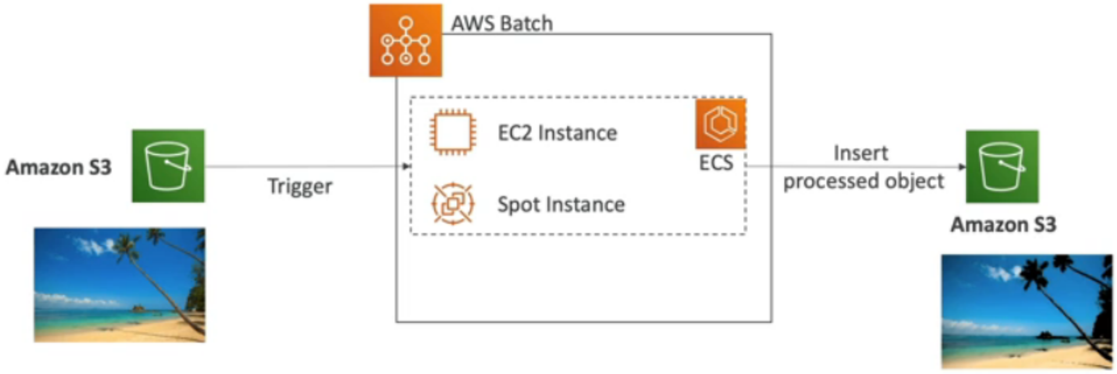
- All in all, serverless API = API Gateway and lambda

AWS Batch

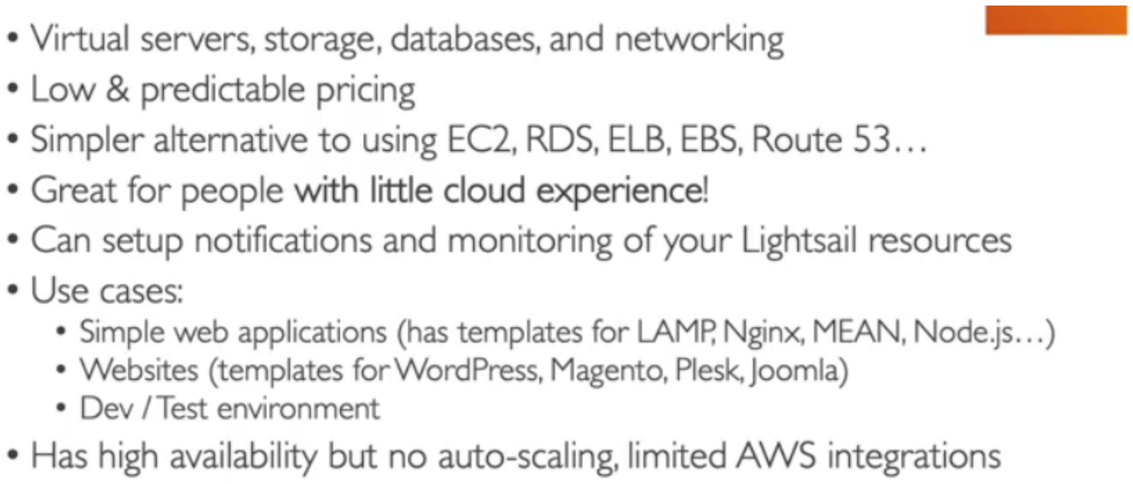
- Note: batch is not a serverless service but a fully managed one which relies on ec2 instances, so, we don't need to provision any instances, aws does.
- A batch job is a job with a starting and ending point (unlike continuous jobs which are ever running). An example of a batch job could be something scheduled from 1pm to 3pm.
- So, AWS Batch is a fully managed batch processing service which can efficiently run 100000s of computing batch jobs on AWS.
- It will dynamically launch EC2 instances or spot instances with the right amount of compute or memory depending on the workload.
- Basically, you are supposed to schedule batch jobs and AWS Batch handles the rest. This leads to less focus on infrastructure and even optimized costs.
- Batch jobs can be defined as docker images and be run on ECS.

- **For example:**

  Let's say we want to handle lots of images that users upload to Amazon S3 all at once. First, we upload an image to Amazon S3, and that kicks off a batch job. This batch job sets up an ECS cluster with the right number of instances (like virtual machines) to handle the load of all the jobs waiting in line. These instances then run special containers (Docker images) to do the job we need. In our case, the job might be something like adding a filter to an image, and then putting the processed image into another Amazon S3 bucket. It's like a smooth process where Docker images and ECS clusters team up to get batch jobs done quickly and efficiently.

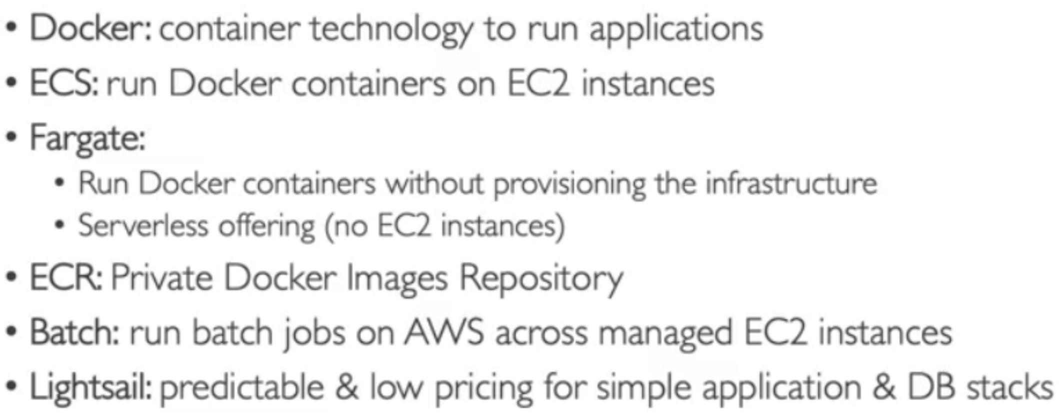


Lightsail

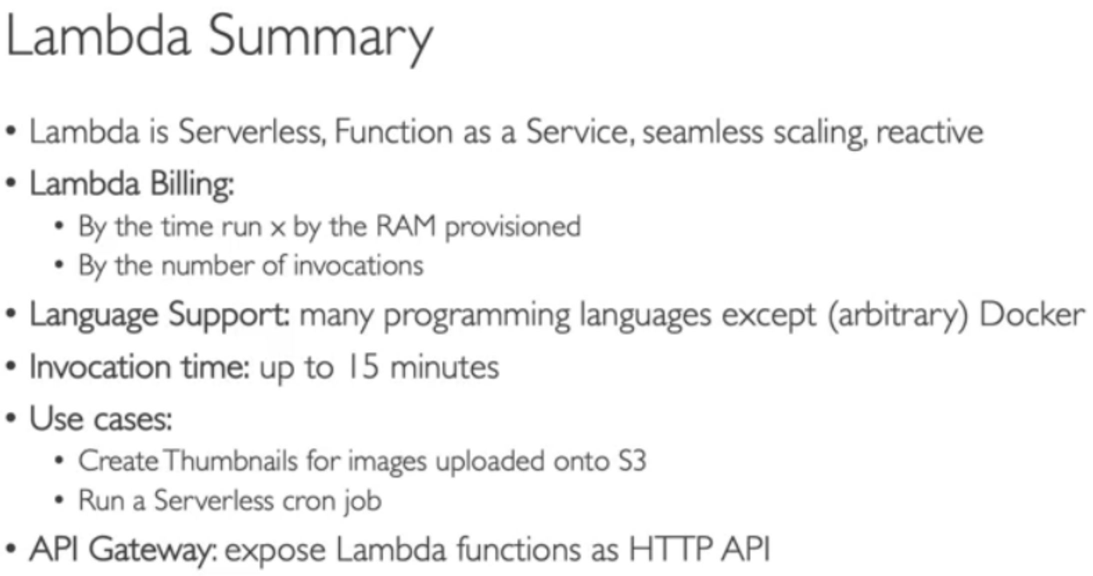


- Virtual servers, storage, databases, and networking
- Low & predictable pricing
- Simpler alternative to using EC2, RDS, ELB, EBS, Route 53…
- Great for people **with little cloud experience!**
- Can setup notifications and monitoring of your Lightsail resources
- Use cases:
  - Simple web applications (has templates for LAMP, Nginx, MEAN, Node.js…)
  - Websites (templates for WordPress, Magento, Plesk, Joomla)
  - Dev / Test environment
- Has high availability but no auto-scaling, limited AWS integrations

SUMMARY

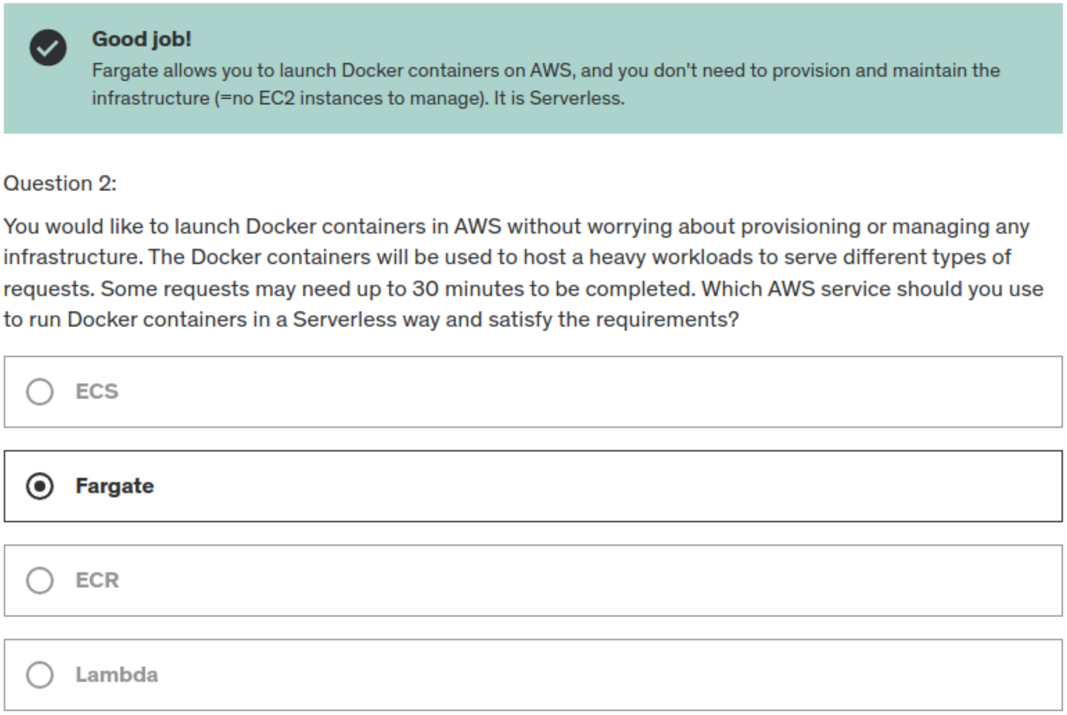


- **Docker:** container technology to run applications
- **ECS:** run Docker containers on EC2 instances
- **Fargate:**
  - Run Docker containers without provisioning the infrastructure
  - Serverless offering (no EC2 instances)
- **ECR:** Private Docker Images Repository
- **Batch:** run batch jobs on AWS across managed EC2 instances
- **Lightsail:** predictable & low pricing for simple application & DB stacks



Lambda Summary

- Lambda is Serverless, Function as a Service, seamless scaling, reactive
- **Lambda Billing:**
  - By the time run x by the RAM provisioned
  - By the number of invocations
- **Language Support:** many programming languages except (arbitrary) Docker
- **Invocation time:** up to 15 minutes
- **Use cases:**
  - Create Thumbnails for images uploaded onto S3
  - Run a Serverless cron job
- **API Gateway:** expose Lambda functions as HTTP API

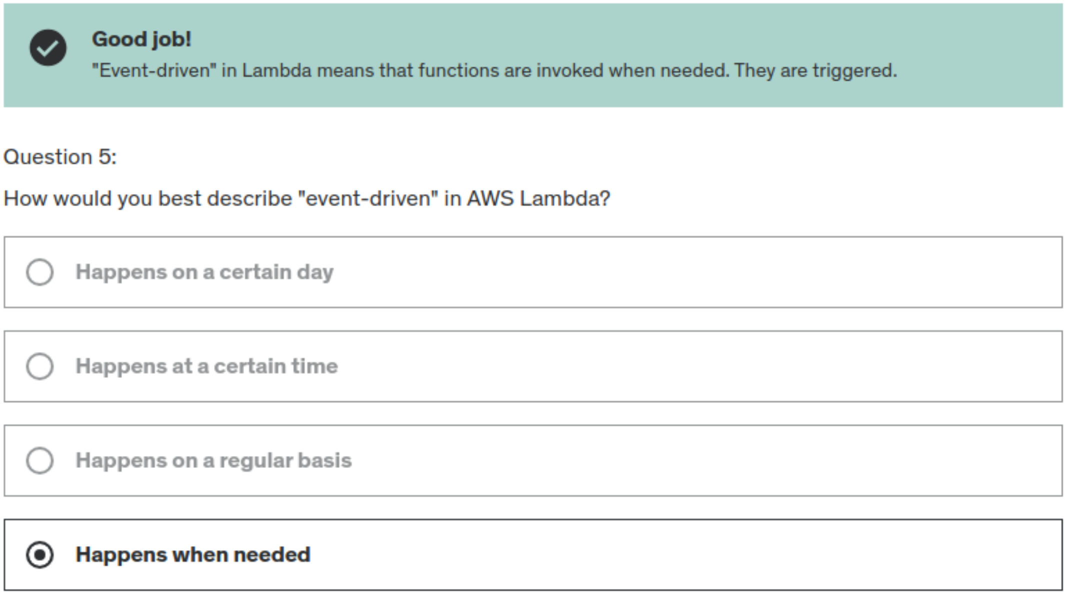


Good job!
Fargate allows you to launch Docker containers on AWS, and you don't need to provision and maintain the infrastructure (=no EC2 instances to manage). It is Serverless.

Question 2:

You would like to launch Docker containers in AWS without worrying about provisioning or managing any infrastructure. The Docker containers will be used to host a heavy workloads to serve different types of requests. Some requests may need up to 30 minutes to be completed. Which AWS service should you use to run Docker containers in a Serverless way and satisfy the requirements?

- ○ ECS
- ● **Fargate**
- ○ ECR
- ○ Lambda

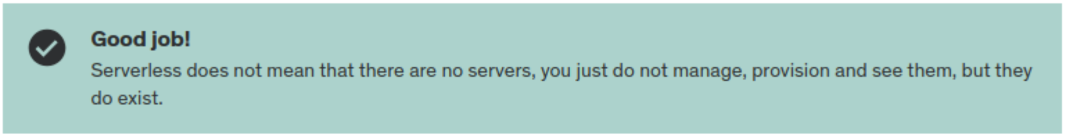


Good job!
"Event-driven" in Lambda means that functions are invoked when needed. They are triggered.

Question 5:

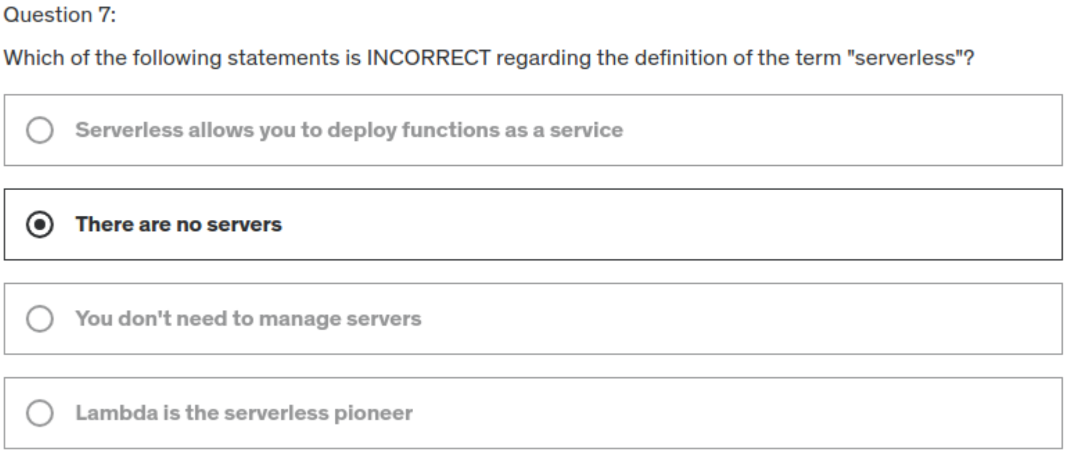
How would you best describe "event-driven" in AWS Lambda?

- ○ Happens on a certain day
- ○ Happens at a certain time
- ○ Happens on a regular basis
- ● **Happens when needed**



Good job!

Serverless does not mean that there are no servers, you just do not manage, provision and see them, but they do exist.



Question 7:

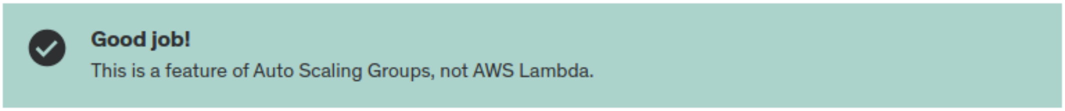
Which of the following statements is INCORRECT regarding the definition of the term "serverless"?

○ Serverless allows you to deploy functions as a service

● **There are no servers**

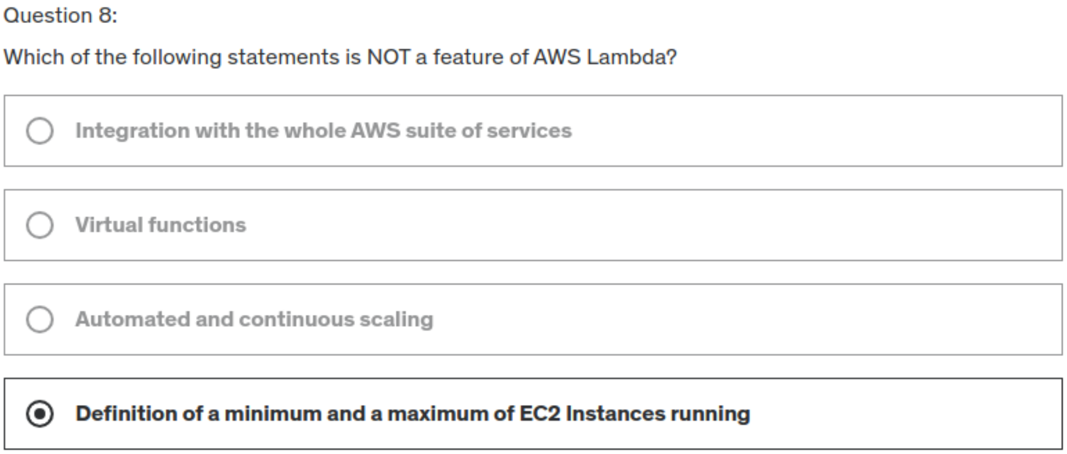
○ You don't need to manage servers

○ Lambda is the serverless pioneer



Good job!

This is a feature of Auto Scaling Groups, not AWS Lambda.



Question 8:

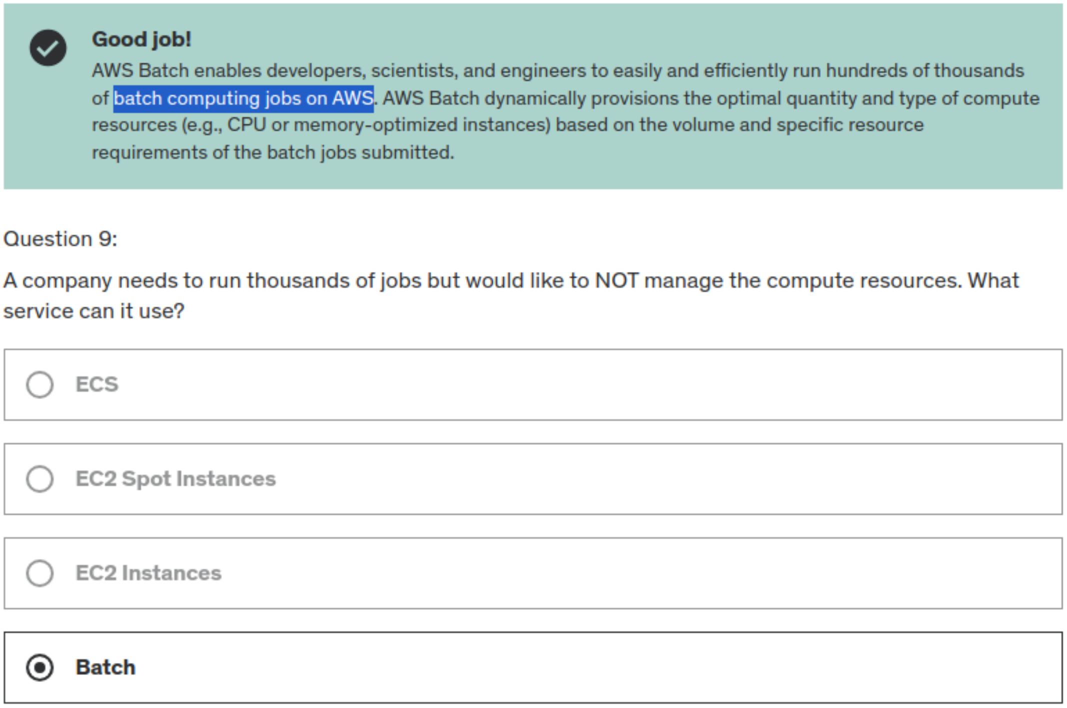
Which of the following statements is NOT a feature of AWS Lambda?

○ Integration with the whole AWS suite of services

○ Virtual functions

○ Automated and continuous scaling

● **Definition of a minimum and a maximum of EC2 Instances running**



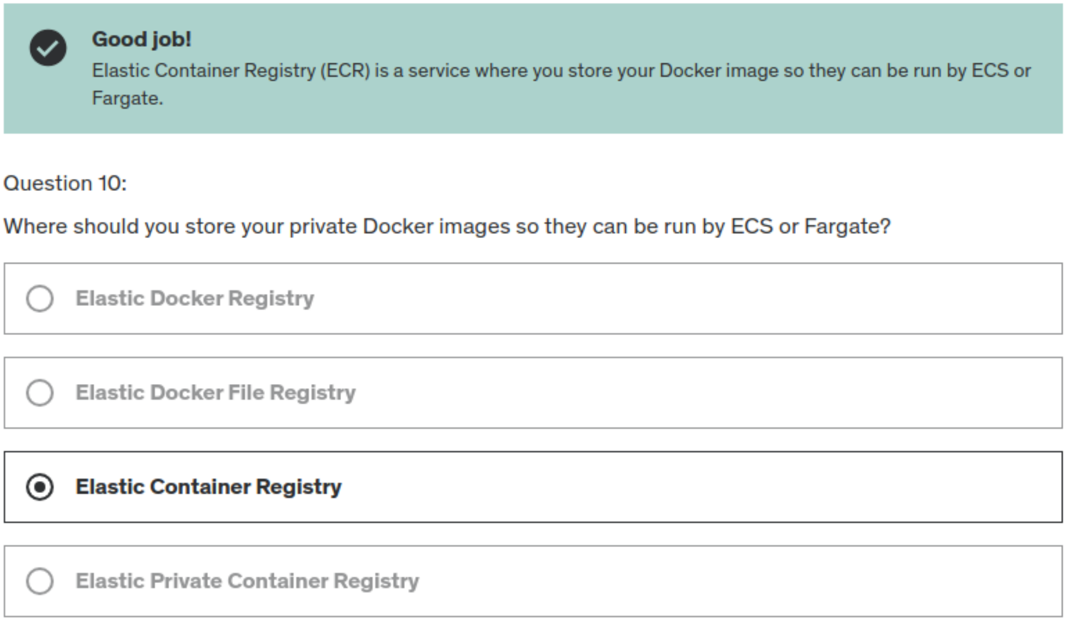
✓ **Good job!**

AWS Batch enables developers, scientists, and engineers to easily and efficiently run hundreds of thousands of batch computing jobs on AWS. AWS Batch dynamically provisions the optimal quantity and type of compute resources (e.g., CPU or memory-optimized instances) based on the volume and specific resource requirements of the batch jobs submitted.

Question 9:

A company needs to run thousands of jobs but would like to NOT manage the compute resources. What service can it use?

- ○ ECS
- ○ EC2 Spot Instances
- ○ EC2 Instances
- ● **Batch**



✓ **Good job!**

Elastic Container Registry (ECR) is a service where you store your Docker image so they can be run by ECS or Fargate.

Question 10:

Where should you store your private Docker images so they can be run by ECS or Fargate?

- ○ Elastic Docker Registry
- ○ Elastic Docker File Registry
- ● **Elastic Container Registry**
- ○ Elastic Private Container Registry