

## **Tutorial 4 (Part 1) - AWS Lambda and Elastic Container Service**

---

### **Objectives**

In this tutorial, you are expected to learn to

- Create a HelloWorld serverless function with AWS Lambda Console using Python 3.7.
  - Create Lambda functions in Python 3.8 using AWS CLI
  - Deploy a Docker-enabled application to Amazon ECS
- 

### **Task 1: Create a HelloWorld serverless function with AWS Lambda**

#### **Introduction**

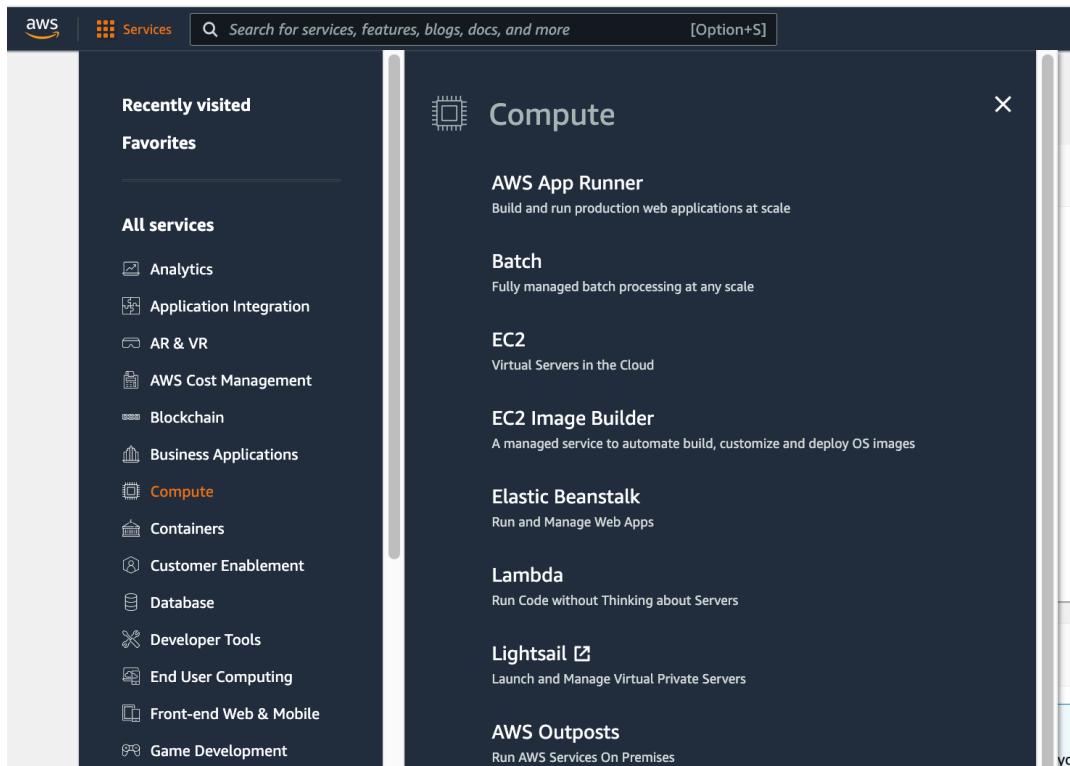
In this tutorial, you will learn the basics of running code on AWS Lambda without provisioning or managing servers. We will walk through how to create a Hello World Lambda function using the AWS Lambda console. We will then show you how to manually invoke the Lambda function using sample event data and review your output metrics.

Everything done in this tutorial is **free tier** eligible.

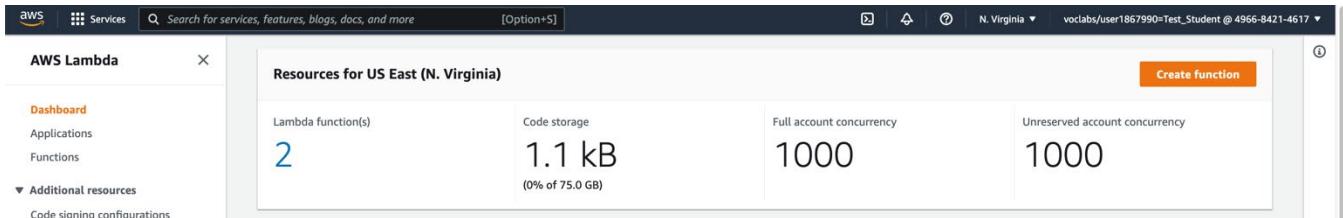
---

#### **Step 1: Enter the Lambda Console.**

- a. Go to the AWS Academy, start a new lab session. Then, browse the AWS Management Console attached to your lab session. We learnt to do this in **Week-2 Lectorial**.
- b. Find **AWS Lambda** under the *Compute* category either by using the *search bar* available in the AWS Management Console or by expanding the *Services* drop-downbox. Click to go to AWS Lambda Console or via other suitable means.



## Step 2: From AWS Lambda dashboard, select **Create function**.



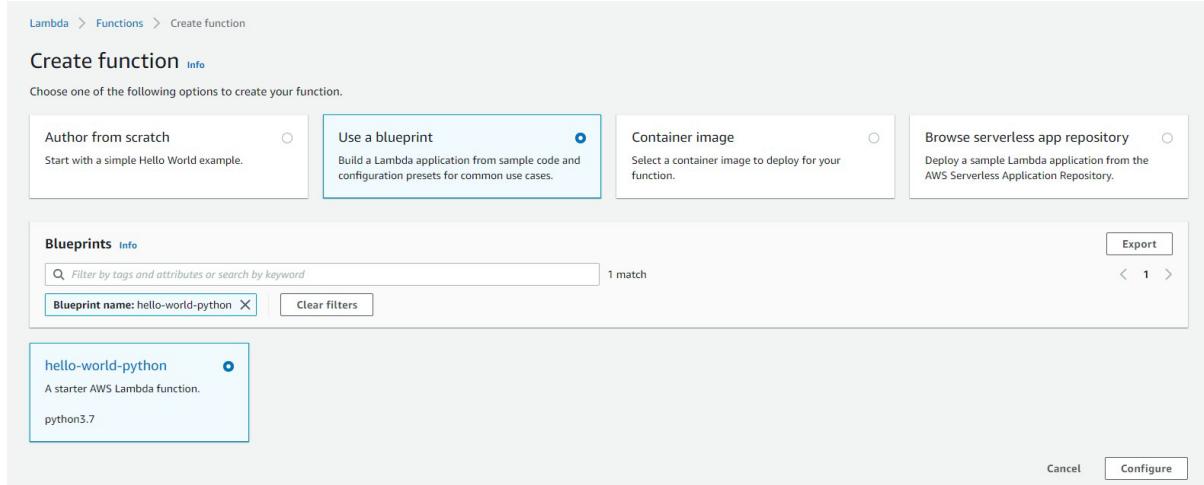
## Step 3: In the “Create function” wizard, select **Use a blueprint** to run a sample Lambda function provided by AWS.

Blueprints provide example code to do some minimal processing. Most blueprints process events from specific event sources, such as Amazon S3, DynamoDB, or a custom application.

- In the AWS Lambda console, select Create a Function.

**Note:** The console shows this page only if you do not have any Lambda functions created. If you have created functions already, you will see the *Lambda > Functions* page. On the list page, choose *Create a function* to go to the Create function page.

- b. Select *Use a blueprint* out of the options available to create a function.
- c. In the *Blueprints* search box, type in *hello-world-python* and select the *hello-world-python* blueprint.
- d. Then click *Configure*.



### Step 3: Configure and Create Your Lambda Function.

A Lambda function consists of code you provide, associated dependencies, and configuration. The configuration information you provide includes the compute resources you want to allocate (for example, memory), execution timeout, and an IAM role that AWS Lambda can assume to execute your Lambda function on your behalf.

- a. You will now enter *Basic Information* about your Lambda function, such as

- Name: You can name your Lambda function here. For this tutorial, enter *hello-world-python*.
- Role: You can either create an IAM role (referred as the execution role) with the necessary permissions that AWS Lambda can assume to invoke your Lambda function on your behalf using *Create new role from template(s)* or select an existing Role using *Use an existing role* option.

**Note:** AWS Academy provides restricted access to students on creating roles as well as several other Identity and Access Management related tasks. Therefore, in this exercise, we will use **Use an existing role** option and select a role that AWS Academy already created for us.

- Role name: Select **LabRole** from the drop-down list.

- Lambda Function Code: In this section, you can review the example code authored in Python. We will leave the existing implementation as is.
- Handler: You can specify a handler (a method/function in your code) where AWS Lambda can begin executing your code. AWS Lambda provides event data as input to this handler, which processes the event.

In this example, Lambda identifies this from the code sample, and this should be pre-populated with `lambda_function.lambda_handler`.

**Basic information** [Info](#)

Function name

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.

Role name  
Enter a name for your new role

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

Policy templates - *optional* [Info](#)  
Choose one or more policy templates.

**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.

Runtime  
Python 3.7

```
1 import json
2
```

- b. Go to the bottom of the page and select **Create function**.
- c. Click on the **Configure** tab from the AWS Lambda options menu to configure the memory, timeout, and VPC settings, etc of your lambda function. For this tutorial, leave the default Lambda function configuration values as is.

The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for Code, Test, Monitor, Configuration (which is highlighted with a red box), Aliases, and Versions. On the left, a sidebar lists various configuration sections: General configuration, Triggers, Permissions, Destinations, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous invocation, Code signing, Database proxies, File systems, and State machines. The main content area is titled 'General configuration' and contains the following details:

General configuration	
Description	A starter AWS Lambda function.
Memory (MB)	128
Timeout	0 min 3 sec

#### Step 4: Invoke Lambda Function and Verify Results.

Now, the AWS Lambda console shows the **hello-world-python** Lambda function - you can now test the function, verify results, and review the logs.

- Select **Test** tab from the AWS Lambda options menu and select **Create new event** as the **Test event action** to create a new test event.
- When you are in the test event creation wizard,
  - Type in an **Event name** like *HelloWorldEvent*.
  - Leave event sharing settings as **Private**, which is its default value.
  - Choose **hello-world** from the sample event **Template** list.
  - You can change the values in the sample JSON, but don't change the event structure. For this tutorial, replace *value1* with the phrase *hello, world!*.

Select **Save changes**.

**Test event**

Test event action

Create new event  Edit saved event

Event name

MyHelloWorldEvent

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

MyHelloWorldEvent

Event JSON

```
1- {
2 "key1": "hello world!",
3 "key2": "value2",
4 "key3": "value3"
5 }
```

Format JSON

c. Click *Test*.

d. Upon successful execution, view the results in the console:

- The Execution results section verifies that the execution succeeded.
- The Summary section shows the key information reported in the Log output.
- The Log output section will show the logs generated by the Lambda function execution.

Execution result: succeeded (logs)

Details

The area below shows the last 4 KB of the execution log.

```
"hello world!"
```

Summary

Code SHA-256	Request ID
c84X7t0/dgIQ2qOodoTIGxJt/sEx2afmpyKaQ1twkuU=	77b0bc3a-27f3-4aba-b386-66e5eaf23df6
Init duration	Duration
170.24 ms	1.61 ms
Billed duration	Resources configured
2 ms	128 MB
Max memory used	
36 MB	

Log output

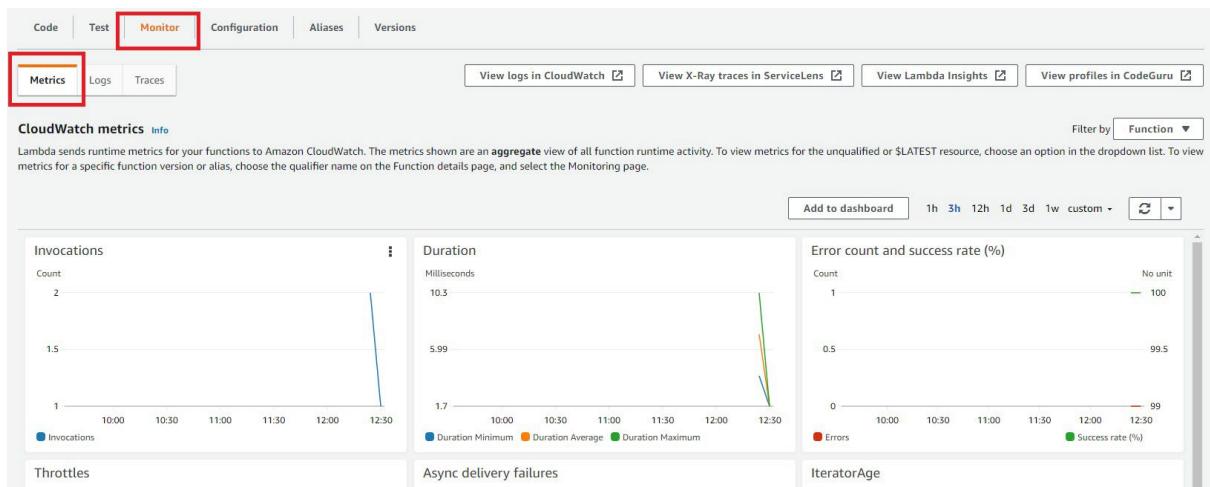
The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: 77b0bc3a-27f3-4aba-b386-66e5eaf23df6 Version: $LATEST
Loading function
value1 = hello world!
value2 = value2
value3 = value3
END RequestId: 77b0bc3a-27f3-4aba-b386-66e5eaf23df6
REPORT RequestId: 77b0bc3a-27f3-4aba-b386-66e5eaf23df6 Duration: 1.61 ms Billed Duration: 2 ms Memory Size: 128 MB Max Memory Used: 36 MB Init Duration: 170.24 ms
```

## Step 5: Monitor Your Metrics

AWS Lambda automatically monitors Lambda functions and reports metrics through Amazon CloudWatch. To help you monitor your code as it executes, Lambda automatically tracks the number of requests, the latency per request, and the number of requests resulting in an error and publishes the associated metrics.

- a. Invoke the Lambda function a few more times by repeatedly clicking the **Test** button. This will generate the metrics that can be viewed in the next step.
- b. Select *Monitor* tab from the AWS Lambda options menu to view the results.
- c. Click the *Metrics* option to view metrics of your Lambda function. Lambda metrics are reported through Amazon CloudWatch. You can leverage these metrics to set custom alarms. For more information about CloudWatch, see the [Amazon CloudWatch Developer Guide](#).



The *Metrics* tab will show seven CloudWatch metrics: *Invocations*, *Duration*, *Error count and success rate (%)*, *Throttles*, *Async delivery failures*, *IteratorAge* and *Concurrent executions*.

With AWS Lambda, you pay for what you use. After you hit your AWS Lambda free tier limit, you are charged based on the number of requests for your functions (invocation count) and the time your code executes (invocation duration). For more information, see [AWS Lambda Pricing](#).

## Step 6: Delete the Lambda Function

While you will not get charged for keeping your Lambda function, you can easily delete it from the AWS Lambda console.

- a. Select the **Actions** button and click **Delete**.
- b. You will be asked to confirm the deletion of your Lambda function - select **Delete**.

---

## Task 2: Create a Lambda function in Python 3.8 using AWS CLI

### Prerequisites

This section describes the tools and resources required to complete the steps in the tutorial.

#### 1. Install the AWS CLI

The AWS CLI is an open-source tool that enables you to interact with AWS services using commands in your command line shell. To complete the steps in this section, you must have the following:

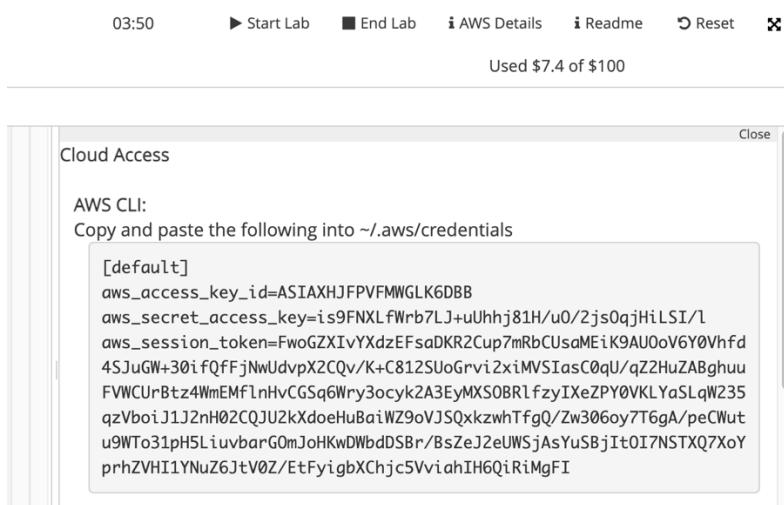
- [AWS CLI – Install version 2](#)

In addition, you also need to make sure that the AWS access credentials are properly configured in your localcomputer/development environment using the steps below.

To configure the access credentials,

- a. Log in to your AWS Academy account, start a lab session if you do not have one running already.
- b. Click *AWS Details* menu item from the “Learner Lab – Associate services” section, and copy the access credentials generated (I.e. `aws_access_key_id`, `aws_secrete_key`, `aws_session_token`) for your AWS Academy lab session.

**Note:** Make sure the credentials you are using are not expired and are associated with your current lab session in AWS Academy portal.



- c. Save the credentials into the `~/.aws/credentials` file that can be located in the logged in user's home directory on **MacOS/Linux** systems. On **Windows**, you will find this file in `%UserProfile%\aws\credentials`. If the file named `credentials` does not exist, create it.

If you are using your personal AWS accounts,

- Run [AWS CLI – Quick configuration](#) with `aws configure` command, as below.

```
C02ZD019LVDF-AmaniSoysa:bqlab3 amani.soysa$ aws configure
AWS Access Key ID [*****AUSN]: xxxxxxxxxxxx
AWS Secret Access Key [*****1p8n]: xxxxxxxxxxxx
Default region name [us-east-1]:
Default output format [None]:
C02ZD019LVDF-AmaniSoysa:bqlab3 amani.soysa$
```

**\*\*\* Run the following step ONLY if you are using your personal AWS accounts \*\*\***

## 2. Create an execution role

Your Lambda function's [execution role](#) is an AWS Identity and Access Management (IAM) role that grants your function permission to access AWS services and resources. Create an execution role in IAM with [Invoke](#) permission.

### To create the execution role

1. Create a `policy.json` file with the following content within your preferred location in the local file-system.

```
{"Version": "2012-10-17", "Statement": [{"Effect": "Allow", "Principal": "*",
"Service": "lambda.amazonaws.com"}, {"Action": "sts:AssumeRole"}]}
```

2. While you are in the same directory, open a command prompt and use the [create-role](#) command to create an execution role named `LabRole`.

```
aws iam create-role --role-name LabRole --assume-role-policy-document
file://policy.json
```

This command produces the following output. Save the value returned in Arn.

```
{
    "Role": {
        "Path": "/",
        "RoleName": "LabRole",
        "RoleId": "AROAWNZPPVHULXRJXQJD5",
        "Arn": "arn:aws:iam::your-account-id:role/LabRole",
        "CreateDate": "2021-01-05T18:00:30Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}
```

Copy `arn:aws:iam::your-account-id:role/LabRole` into a .txt file to use later.

3. Use the [attach-role-policy](#) command to add AWSLambdaBasicExecutionRole permissions to the role.

```
$ aws iam attach-role-policy --role-name LabRole --policy-arn
arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

This command produces no output.

---

### 3. Creating a function without runtime dependencies

A dependency can be any package, module or other assembly dependency that is not included with the [Lambda runtime](#) environment for your function's code. For more information, see [What is a runtime dependency?](#).

For more information, see [What is a runtime dependency?](#)

This section describes how to create a Lambda function without runtime dependencies.

#### Overview

In this tutorial, you will use a [sample Lambda function implementation](#) available in AWS SDK examples Github repository, to create a Lambda function using the AWS CLI. You will learn how to:

- Set up the directory structure of a deployment package (.zip file).
- Create a deployment package for a Lambda function without any runtime dependencies.
- Use the AWS CLI to upload the deployment package and create the Lambda function.
- Invoke the Lambda function to return a mathematical calculation.

The sample code contains standard math and logging Python libraries, which are used to return a calculation based on user input. Standard Python libraries are included with the `python3.8 runtime`. Although the function's code doesn't depend on any other Python libraries and has no additional application dependencies, AWS Lambda still requires a deployment package to create and deploy a Lambda function.

#### Create the deployment package

Your AWS Lambda function's code consists of scripts or compiled programs and their dependencies. You use a *deployment package* to deploy your function code to Lambda. Lambda supports two types of deployment packages: container images and .zip file archives.

Create the .zip file that Lambda uses as your deployment package.

##### To create the deployment package

1. Open a command prompt and create a `my-math-function` project directory. For example, on macOS:

```
mkdir my-math-function
```

2. Navigate to the `my-math-function` project directory.

```
cd my-math-function
```

3. Copy the contents of this [sample Lambda function implementation](#) and save it in a new filenamed `lambda_function.py`. Your directory structure should look like this.

```
my-math-function$  
| lambda_function.py
```

4. Create a `my-deployment-package.zip` file adding the `lambda_function.py` to the root of the zip file.

This generates a `my-deployment-package.zip` file in your project directory.

## Create the Lambda function

Lambda needs to know the [runtime](#) environment to use for your function's code, the [handler](#) in your function code, and the [execution role](#) that it can use to invoke your function.

Create the Lambda function using the execution role and deployment package that you created in the previous steps.

### To create the function

1. Navigate to the `my-math-function` project directory.

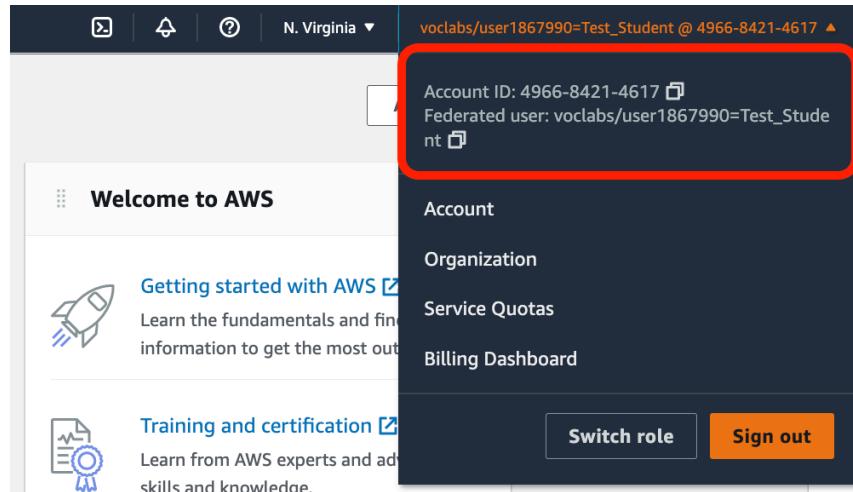
```
cd my-math-function
```

2. Create a function named `my-math-function`.

If you are using your own AWS account for this exercise, substitute the value for `--role` with the Arn you copied in Step 2.

If you are using an AWS Academy account, follow the steps below to generate the Arn of the **LabRole**, which is to be used with the --role directive.

- a. Find the **Account ID** associated with your AWS Academy account by clicking on the user info drop-down box to the top right hand side corner of the AWS Management Console, as shown below.



- b. Now, substitute this Account ID for the your-account-id in the role ARN  
**arn:aws:iam::your-account-id:role/LabRole**

E.g. **arn:aws:iam::496684214617/LabRole**

Now, run the following command.

```
aws lambda create-function --function-name my-math-function --zip-file
fileb://my-deployment-package.zip --handler lambda_function.lambda_handler
--runtime python3.8 --role arn:aws:iam::your-account-id:role/LabRole
```

This command produces an output similar to what is shown below.

```
{
  "FunctionName": "my-math-function",
  "FunctionArn": "arn:aws:lambda:us-east-1:496684214617:function:my-math-function",
  "Runtime": "python3.8",
  "Role": "arn:aws:iam::496684214617:role/LabRole",
  "Handler": "lambda_function.lambda_handler",
  "CodeSize": 795,
  "Description": "",
  "Timeout": 3,
  "MemorySize": 128,
  "LastModified": "2022-03-29T19:49:04.632+0000",
  "CodeSha256": "xbLyNtTM63CN0RUSMoB8UwJHJuFw86Tq/K4d4w1Kp/0=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "e0959a8d-b1ed-41c1-b094-5923faf11faa",
  "State": "Pending",
  "StateReason": "The function is being created.",
  "StateReasonCode": "Creating",
  "PackageType": "Zip"
}
```

The Lambda function in this step uses a function handler named `lambda_function.lambda_handler`. For more information about function handler naming conventions, see [Naming in AWS Lambda function handler in Python](#).

## Invoke the Lambda function

Invoke the Lambda function [synchronously](#) using the event input defined for the sample code. For more information, see [How it works](#) in [AWS Lambda function handler in Python](#).

### To invoke the function

- Create `input.json` file containing the following content.

```
{"action": "square", "number": 3}
```

- Use the [invoke](#) command.

**Note:** If you are using an AWS CLI version prior to v2, remove the following directive from your command.

```
--cli-binary-format raw-in-base64-out
```

```
aws lambda invoke --function-name my-math-function --payload
file://input.json output.txt --cli-binary-format raw-in-base64-out
```

This command produces the following output.

```
{
  "StatusCode": 200,
  "ExecutedVersion": "$LATEST"
}
```

For the RequestResponse invocation type, the status code is 200. For more information, see the [Invoke](#) API reference.

You should see the following mathematical calculation in `output.txt`:

```
{"result": 9}
```

## Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them for further experimentation. By deleting AWS resources that you are no longer using, you prevent unnecessary charges to your AWS account.

### To delete the Lambda function

- Use the [delete-function](#) command.

```
aws lambda delete-function --function-name my-math-function
```

This command produces no output.

## Creating a function with runtime dependencies

A dependency can be any package, module or other assembly dependency that is not included with the [Lambda runtime](#) environment for your function's code. For more information, see [what is a runtime dependency?](#).

For more information, see [what is a runtime dependency?](#)

This section describes how to create a Lambda function with runtime dependencies.

### Overview

In this tutorial, you use sample code to create a Lambda function using the AWS CLI. The sample code uses the **requests** python package/library to get the HTML code associated with the web page <https://docs.aws.amazon.com/>. The aforementioned **requests** package/library is one developed by a third-party and therefore, not included with the core python3.8 [runtime](#). So, you need to install it and package it into your deployment package.

In this exercise, you will learn to

- Set up the directory structure of a deployment package (.zip file) with runtime dependencies.
- Create a deployment package for a Lambda function with runtime dependencies.
- Use the AWS CLI to upload the deployment package and create the Lambda function.
- Invoke the Lambda function to return the source code.

## Create the deployment package

Your AWS Lambda function's code consists of scripts or compiled programs and their dependencies. You use a *deployment package* to deploy your function code to Lambda. Lambda supports two types of deployment packages: container images and .zip file archives. In this exercise, we will use the latter.

Create the .zip file that Lambda uses as your deployment package.

### To create the deployment package

1. Open a command prompt and create a `my-advanced-function` project directory.  
Forexample, on macOS:

```
mkdir my-advanced-function
```

2. Navigate to the `my-advanced-function` project directory.

```
cd my-advanced-function
```

3. Copy the contents of the following sample Python code and save it in a new filenamed `lambda_function.py`:

```
import requests
def lambda_handler(event, context):
    response = requests.get("https://docs.aws.amazon.com")
    print(response.text)
    return response.text
```

Your directory structure should look like this:

```
my-advanced-function
| lambda_function.py
```

4. Install the requests library to a new package directory, which resides in the `my-advanced-function` directory.

```
pip install --target ./package requests
```

5. Create a deployment package `my-deployment-package.zip` with the `lambda_function.py` and all the installed libraries at the root within the zip file.

This generates a `my-deployment-package.zip` file in your project directory.

**Note:** Do not bundle the package directory. You only need to copy the directories inside the package directory into the root directory within the zip file.

## Create the Lambda function

Lambda needs to know the [runtime](#) environment to use for your function's code, the [handler](#) in your function code, and the [execution role](#) that it can use to invoke your function.

Create the Lambda function using the execution role and deployment package that you created in the previous steps.

### To create the function

1. Navigate to the `my-advanced-function` project directory.

```
cd my-advanced-function
```

2. Create a function named `my-advanced-function`. Substitute the value for `role` with the `Arn` you copied in previous steps.

```
aws lambda create-function --function-name my-advanced-function --zip-file fileb://my-deployment-package.zip --handler lambda_function.lambda_handler --runtime python3.8 --role arn:aws:iam::your-account-id:role/LabRole
```

This command produces an output similar to what is shown below.

```
{
  "FunctionName": "my-advanced-function",
  "FunctionArn": "arn:aws:lambda:us-east-1:496684214617:function:my-advanced-function",
  "Runtime": "python3.8",
  "Role": "arn:aws:iam::496684214617:role/LabRole",
  "Handler": "lambda_function.lambda_handler",
  "CodeSize": 788251,
  "Description": "",
  "Timeout": 3,
  "MemorySize": 128,
  "LastModified": "2022-03-29T20:22:25.225+0000",
  "CodeSha256": "OB1mpS6aQFEGDs098G11Bi65JNqpd3L0iNY6Z/DTlWE=",
  "Version": "$LATEST",
  "TracingConfig": {
    "Mode": "PassThrough"
  },
  "RevisionId": "e9298995-8d59-42c3-9273-61f648fefafc",
  "State": "Pending",
  "StateReason": "The function is being created.",
  "StateReasonCode": "Creating",
  "PackageType": "Zip"
}
```

The Lambda function in this step uses a function handler of `lambda_function.main`. For more information about function handler naming conventions, see [Naming in AWS Lambda function handler in Python](#).

## Invoke the Lambda function

Invoke the Lambda function [synchronously](#) using the event input defined for the sample code. For more information, see [How it works](#) in [AWS Lambda function handler in Python](#).

### To invoke the function

- Create `input.json` file with the following content

```
{"dummyKey": "dummyValue"}
```

- Use the [invoke](#) command.

```
aws lambda invoke --function-name my-advanced-function --cli-binary-format raw-in-base64-out --payload file://input.json output.txt
```

**Note:** If you are using an AWS CLI version prior to v2, remove the following directive from the command:

```
--cli-binary-format raw-in-base64-out
```

This command produces the following output:

```
{  
  "StatusCode": 200,  
  "ExecutedVersion": "$LATEST"  
}
```

For the `RequestResponse` invocation type, the status code is `200`. For more information, see the [Invoke](#) API reference.

You should see the HTML code of the web page <https://docs.aws.amazon.com> in `output.txt`.

## Updating a Lambda function in Python 3.8

### To update a Python function

1. Add updated function code files to the root of your deployment package.
2. Use the `fileb://` prefix to upload the binary `.zip` file to Lambda and update the function code.

```
~/my-function$ aws lambda update-function-code --function-name my-  
advanced-function --zip-file fileb://my-deployment-package.zip
```

## Clean up your resources

You can now delete the resources that you created for this tutorial, unless you want to retain them. By deleting AWS resources that you're no longer using, you prevent unnecessary charges to your AWS account.

### To delete the Lambda functions

- Use the [delete-function](#) command.

```
aws lambda delete-function --function-name my-advanced-function
```

This command produces no output.

**Note:** Also delete the execution role (if you used your personal AWS account for this exercise).

### Task 3: Deploy Docker Containers on Amazon Elastic Container Service (Amazon ECS)

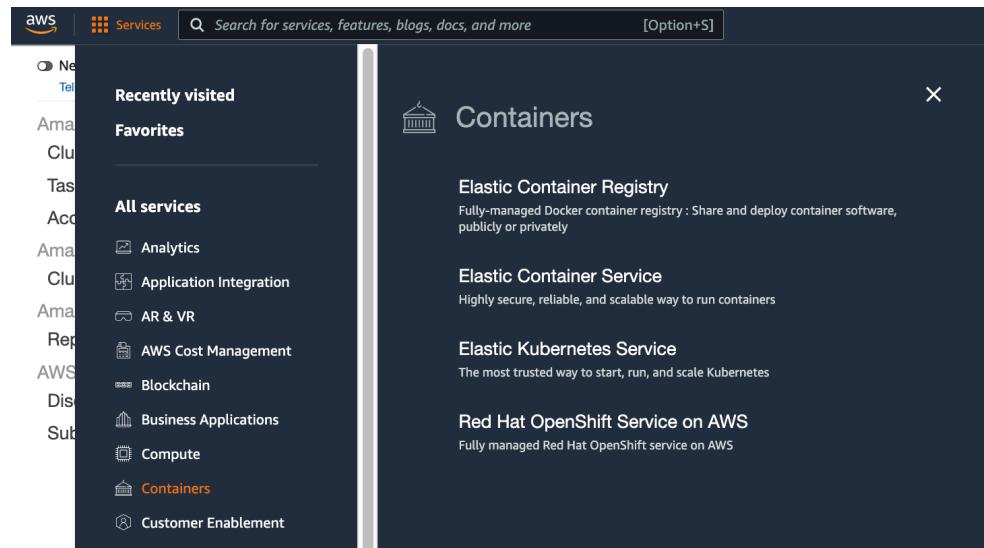
Amazon Elastic Container Service (Amazon ECS) is the Amazon Web Service you use to run Docker applications on a scalable cluster of compute resources. In this tutorial, you will learn how to run a Docker-enabled sample application on an Amazon ECS cluster behind a load balancer, test the sample application, and delete your resources to avoid charges.

Everything done in this tutorial is [free tier](#) eligible.

---

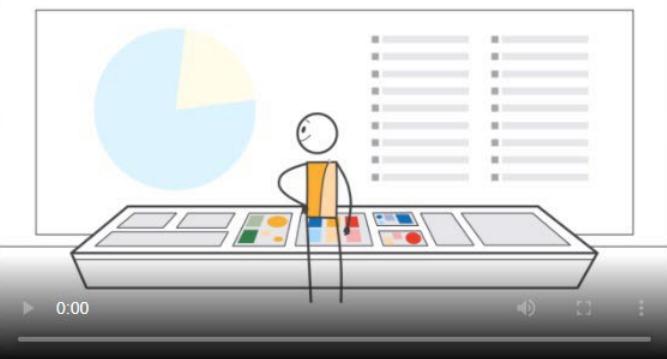
#### Step 1: Set up your first ECS cluster and run it

Select AWS Elastic Container Service from the list of services available in your AWS Management Console under the **Containers** category.



The Amazon ECS *Get started* wizard will guide you through the process of creating a cluster and launching a sample web application. In this step, you will enter the Amazon ECS console and launch the wizard by clicking on the *Get started* button.

## Amazon Elastic Container Service (ECS)



containers running applications, services, and batch processes. Amazon ECS places containers across familiar features like Elastic Load Balancing, EC2 security groups, EBS volumes and IAM roles.

[Get started](#)

[Learn more about Amazon ECS](#)

### Step 2: Create a container and task definitions

A *container definition* is used in task definitions to describe the different containers that are launched as part of a task.

A *task definition*, meanwhile, is like a blueprint for your application. In this step, you will select a container definition and task definition, so Amazon ECS knows which Docker image to use for containers, how many containers to use in the task, and the resource allocation for each container.

The task definition comes pre-loaded with default configuration values.

- Review the default values and select *Next Step*.

If you prefer to modify the configurations or would like to learn more, see [Task Definition Parameters](#).

**Important note:** If you are using an AWS Academy account for this exercise, remember to choose **LabRole** as your **Task execution role**.

Getting Started with Amazon Elastic Container Service (Amazon ECS) using the AWS Management Console

**Step 1: Container and Task**

Step 2: Service  
Step 3: Cluster  
Step 4: Review

Diagram of ECS objects and how they relate

Container definition Edit

Choose an image for your container below to get started quickly or define the container image to use.

<b>sample-app</b> image : httpd:2.4 memory : 0.5GB (512) cpu : 0.25 vCPU (256)	<b>nginx</b> image : nginx:latest memory : 0.5GB (512) cpu : 0.25 vCPU (256)
<b>tomcat-webserver</b> image : tomcat memory : 2GB (2048) cpu : 1 vCPU (1024)	<b>custom</b> <span style="float: right;">Configure</span> image : -- memory : -- cpu : --

Task definition Edit

A task definition is a blueprint for your application, and describes one or more containers through attributes. Some attributes are configured at the task level but the majority of attributes are configured per container.

Task definition name	first-run-task-definition	<span style="float: right;">i</span>
Network mode	awsvpc	<span style="float: right;">i</span>
Task execution role	Create new	<span style="float: right;">i</span>
Compatibilities	FARGATE	<span style="float: right;">i</span>
Task memory	0.5GB (512)	
Task CPU	0.25 vCPU (256)	

## Step 3: Configure your service

Now that you have created a container and task definition, you will configure the Amazon ECS service. A service launches and maintains copies of the task definition in your cluster. For example, by running an application as a service, Amazon ECS will auto-recover any stopped tasks and maintain the number of copies you specify.

### a. Configure service options:

- **Service Name:** The default *sample-app-service* is a web-based "Hello World" application provided by AWS. It is meant to run indefinitely, so by running it as a service, it will restart if the task becomes unhealthy or unexpectedly stops.
- **Desired number of tasks:** To stay within the [AWS free tier](#), leave the default value of 1. This will create 1 copy of your task.

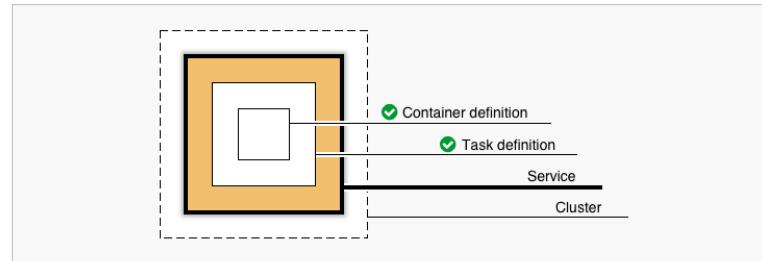
Step 1: Container and Task

**Step 2: Service**

Step 3: Cluster

Step 4: Review

## Diagram of ECS objects and how they relate



## Define your service

**Edit**

A service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster.

Service name sample-app-service

Number of desired tasks 1

Security group Automatically create new

A security group is created to allow all public traffic to your service only on the container port specified.  
You can further configure security groups and network access outside of this wizard.

Load balancer type  None  
 Application Load Balancer

**\*Required****Cancel****Previous****Next**

- b. Elastic load balancing: You have the option to use a load balancer with your service. Amazon ECS can create an Application Load Balancer (ALB) to distribute the traffic across the container instances your task is launched on.

- The default values for ALB *listener protocol* and *listener port* are set up for the sample application. For more information on load balancing configuration, see [Service Load Balancing](#).

Step 1: Container and Task

**Step 2: Service**

Step 3: Cluster

Step 4: Review

Diagram of ECS objects and how they relate

Define your service

A service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster.

Service name: sample-app-service

Number of desired tasks: 1

Security group: Automatically create new  
Two security groups are created to secure your service: An Application Load Balancer security group that allows all traffic on the Application Load Balancer port and an Amazon ECS security group that allows all traffic ONLY from the Application Load Balancer security group. You can further configure security groups and network access outside of this wizard.

Load balancer type:  Application Load Balancer

Load balancer listener port: 80

Load balancer listener protocol: HTTP

\*Required      Cancel      Previous      Next

d. Review your settings and select Next Step.

## Step 4: Configure your cluster

Your Amazon ECS tasks run on a *cluster*, which is the set of container instances (i.e EC2 instances) running the Amazon ECS container agent. In this step, we create a *Fargate cluster*, which is a fully managed service offered by AWS. In *Fargate*, your containers run without you managing and configuring individual Amazon EC2 instances.

- Leave the cluster name as *default*.

- Click *Next*.

## Getting Started with Amazon Elastic Container Service (Amazon ECS) using Fargate

Step 1: Container and Task  
Step 2: Service  
**Step 3: Cluster**  
Step 4: Review

Diagram of ECS objects and how they relate

Configure your cluster

The infrastructure in a Fargate cluster is fully managed by AWS. Your containers run without you managing and configuring individual Amazon EC2 instances.

To see key differences between Fargate and standard ECS clusters, see the [Amazon ECS documentation](#).

Cluster name  Cluster names are unique per account per region. Up to 255 letters (uppercase and lowercase), numbers, and hyphens are allowed.

VPC ID  i

Subnets  i

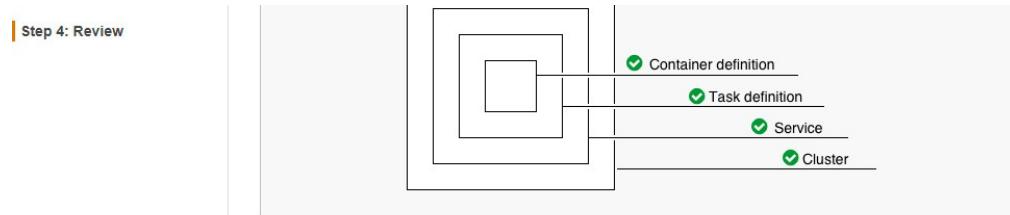
\*Required Cancel Previous **Next**

In previous steps, you have configured your container and task definition (which is like an application blueprint), the Amazon ECS service (which launches and maintains copies of your task definitions), and a Fargate cluster (which is the set of container instances running the container agent). In this step, you will review, launch, and view the resources you create.

- You have a final chance to review your task definition, task configuration, and cluster configurations before launching.

- Select *Create* to launch your resources.

This may take a few minutes.



## Review

Review the configuration you've set up before creating your task definition, service, and cluster.

### Task definition

[Edit](#)

Task definition name: first-run-task-definition

Network mode: awsvpc

Task execution role: Create new

Container name: sample-app

Image: httpd:2.4

Memory: 512

Port: 80

Protocol: HTTP

### Service

[Edit](#)

Service name: sample-app-service

Number of desired tasks: 1

Load balancer listener port: 80

Load balancer listener protocol: HTTP

### Cluster

[Edit](#)

Cluster name: default

VPC ID: Automatically create new

Subnets: Automatically create new

\*Required

[Cancel](#)

[Previous](#)

[Create](#)

- After the launch is complete, select *View service*.

**Note:** While your ECS cluster is getting provisioned, examine the types of resources it creates for you automatically.

Creating resources for your service. This may take up to 10 minutes. When we're complete, you can view your service.

### Launch Status

We are creating resources for your service. This may take up to 10 minutes. When we're complete, you can view your service.

[Back](#)

[View service](#)

Additional features that you can add to your service after creation

#### Scale based on metrics

You can configure scaling rules based on CloudWatch metrics

Preparing service : 10 of 10 complete

#### ECS resource creation

Cluster default

Task definition first-run-task-definition:1

Service sample-app-service

#### Additional AWS service integrations

Log group /ecs/first-run-task-definition

CloudFormation stack EC2ContainerService-default

VPC vpc-07630b4550c6c29b4

Subnet 1 subnet-0c5ad545428c223b1

Subnet 2 subnet-06d1a0bc72c3c4884

Security group sg-09da85ed4f2efad88

Load balancer am:aws.elasticloadbalancing.us-east-1:494911531456:loadbalancer/app/EC2Co-EcsEl-1801ZT23DS0A9/8a9ff9523a0e72880

## Step 6: Open the Sample Application

In this step, you will verify that the sample application is up and running by pointing your browser to the load balancer DNS name.

The screenshot shows the AWS ECS Service details page for 'sample-app-service'. The service is active and uses the 'first-run-task-definition:1' task definition. It is a Replica service type, Fargate launch type, and has the AWS Service Role for ECS as its service role. It was created by 'arn:aws:iam::494911531456:role/vocstartsoft'. The 'Details' tab is selected, showing the 'Load Balancing' section. In this section, the target group name is 'EC2Co-Defau-LRCKBILV6K12', which is highlighted with a red box. The container name is 'sample-app' and the container port is 80. Below this, the 'Network Access' section shows the health check grace period as 0, allowed VPC as 'vpc-07630b4550c6c29b4', allowed subnets as 'subnet-06d1a0bc72c3c4884, subnet-0c5ad545428c223b1', security groups as 'sg-09da85ed4f2efad88', and auto-assign public IP as 'ENABLED'.

- From the *Target groups* page you were redirected to in the previous step, click the *Load balancer* link.

**EC2Co-Defau-LRCKBILV6K12**

Details

Target type	IP	Protocol	HTTP: 80	Protocol version	HTTP/1
Load balancer		vpc-07630b4550c6c29b4			
Total targets	1	Healthy	1	Unhealthy	0
Unused	0	Initial	0	Draining	0

Targets | Monitoring | Health checks | Attributes | Tags

Registered targets (1)

IP address	Port	Zone	Health status	Health status details
10.0.1.213	80	us-east-1b	healthy	

- From the load balancer's information section, copy the *DNS name* associated with it.
- Paste it into a new browser window.
- You will see the *Amazon ECS Sample App* being rendered into the browser window.

EC2 Co-EcsEl-1801ZT23DS0A9

Name	DNS name	State	VPC ID	Availability Zones	Type	Created At	Monitoring
EC2Co-EcsEl-1801ZT23DS...	EC2Co-EcsEl-1801ZT23DS...	Active	vpc-07630b4550c6c29b4	us-east-1b, us-east-1a	application	July 14, 2021 at 12:17:47 A...	Enabled

**EC2Co-EcsEl-1801ZT23DS0A9 (A Record)**

Name: EC2Co-EcsEl-1801ZT23DS0A9  
ARN: arn:aws:elasticloadbalancing:us-east-1:149491151456:loadbalancer/app/EC2Co-EcsEl-1801ZT23DS0A9/8a9f9523a0e72680  
State: Active  
Type: application  
Scheme: internet-facing  
IP address type: ipv4  
VPC: vpc-07630b4550c6c29b4  
Availability Zones:  
Hosted zone: Z35SXDDOTRQ7X7K  
Creation time: July 14, 2021 at 12:17:47 AM UTC+10  
Security groups: sg-01ef0db20c4dc010, EC2ContainerService-default-AlbSecurityGroup-1S605DEQIKABB  
Edit security groups

**Amazon ECS Sample App**

**Congratulations!**

Your application is now running on a container in Amazon ECS.

## Step 7: Delete Your Resources

- Click on the cluster name (e.g. *default*).

The screenshot shows the AWS ECS Cluster Overview page. On the left, there's a sidebar with links like Amazon ECS, Clusters, Task Definitions, Account Settings, etc. The main area shows the 'default' cluster details. A red box highlights the cluster name 'default' and its monitoring status: 'CloudWatch monitoring' and 'Default Monitoring'. Below this, it shows service counts: 1 Fargate service, 1 running task, and 0 pending tasks. It also shows EC2 and EXTERNAL sections with 0 services, 0 running tasks, and 0 pending tasks each.

- Select the checkbox next to *sample-app-service* and click *Update*.

The screenshot shows the AWS ECS Cluster Details page for the 'default' cluster. At the top, it displays the Cluster ARN (arn:aws:ecs:us-east-1:494911531456:cluster/default), Status (ACTIVE), and Registered container instances (0). Below this, it shows Pending tasks count (0 Fargate, 0 EC2, 0 External), Running tasks count (1 Fargate, 0 EC2, 0 External), Active service count (1 Fargate, 0 EC2, 0 External), and Draining service count (0 Fargate, 0 EC2, 0 External). A table lists services, with the 'sample-app-service' row highlighted by a red box. The table includes columns for Service Name (checkbox checked for sample-app-service), Status (ACTIVE), Service type (REPLICA), and Task Definition (first-run-task-definition:1). Navigation tabs at the bottom include Services, Tasks, ECS Instances, Metrics, Scheduled Tasks, Tags, and Capacity Providers. Action buttons like Create, Update, Delete, and Actions are available.

- To ensure you don't accidentally delete a service with active tasks, you need to stop all tasks before Amazon ECS will delete a service.

- After you update your service, select *Delete*.

**Step 1: Configure service**

Step 2: Configure network  
Step 3: Set Auto Scaling (optional)  
Step 4: Review

### Configure service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Task Definition Family: first-run-task-definition  Revision: 1 (latest)

Launch type: FARGATE

Platform version: LATEST

Force new deployment:

Cluster: default

Service name: sample-app-service

Service type\*: REPLICA

Number of tasks:

Minimum healthy percent: 100

Maximum percent: 200

Deployment circuit breaker: Disabled

\*Required

---

Amazon ECS

**Clusters**

- Task Definitions
- Account Settings
- Amazon EKS
- Clusters
- Amazon ECR
- Repositories
- AWS Marketplace
- Discover software
- Subscriptions

### Cluster : default

Get a detailed view of the resources on your cluster.

Cluster ARN: am:aws:ecs:us-east-1:494911531456:cluster/default  
Status: ACTIVE

Registered container instances: 0

Pending tasks count	0 Fargate, 0 EC2, 0 External
Running tasks count	1 Fargate, 0 EC2, 0 External
Active service count	1 Fargate, 0 EC2, 0 External
Draining service count	0 Fargate, 0 EC2, 0 External

Services    Actions

Filter in this page	Launch type	Service type	1 selected
<input type="checkbox"/> Service Name	Status	Service type	Task Definition
<input checked="" type="checkbox"/> sample-app-service	ACTIVE	REPLICA	first-run-task-definition:1

d. Delete *Load balancer* created for your application:

- [Enter the Amazon EC2 console](#)
- In the left hand panel, select *Load balancers*.
- Select the checkbox next to the created Load balancer (e.g. EC2Co-EcsEl-1801ZT23DS0A9).
- Select Actions > Delete.

The screenshot shows the AWS EC2 Dashboard with the 'Load Balancers' section selected. A specific load balancer, 'EC2Contai-EcsElast-YY3FOKBZ89H9', is highlighted with a red box. A context menu is open over this item, listing several actions: 'Delete', 'Edit health check', 'Edit subnets', 'Edit instances', 'Edit listeners', and 'Edit security groups'. The 'Delete' option is clearly marked with a red box. The main pane displays details about the load balancer, including its DNS name, port configuration, availability zones, instance count, health check, and creation date.

## Congratulations!

Congratulations! You have learned how to configure, deploy, and delete your Docker-enabled application to Amazon Elastic Container Service (Amazon ECS). Amazon ECS is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.

Additional Documents:

- Updating a Lambda function in Python 3.8.  
<https://docs.aws.amazon.com/lambda/latest/dg/python-package-update.html>
- Docker Desktop overview. <https://docs.docker.com/desktop/>
- A Docker Tutorial for Beginners. <https://docker-curriculum.com/>

**Note: Terminate all your AWS resources after this lab to avoid extra costs.**