CLOUD TRAIN

ACCELERATE YOUR GROWTH

MODULE 9

AWS LAMBDA, BEANSTALK & CLI

AWS Workshop

Contact us

TO ACCELERATE YOUR CAREER GROWTH

For questions and more details:

please call @ +91 98712 72900, or

visit https://www.thecloudtrain.com/, or

email at support@thecloudtrain.com, or

WhatsApp us @ +91 98712 72900

How to run an app using AWS Elastic Beanstalk

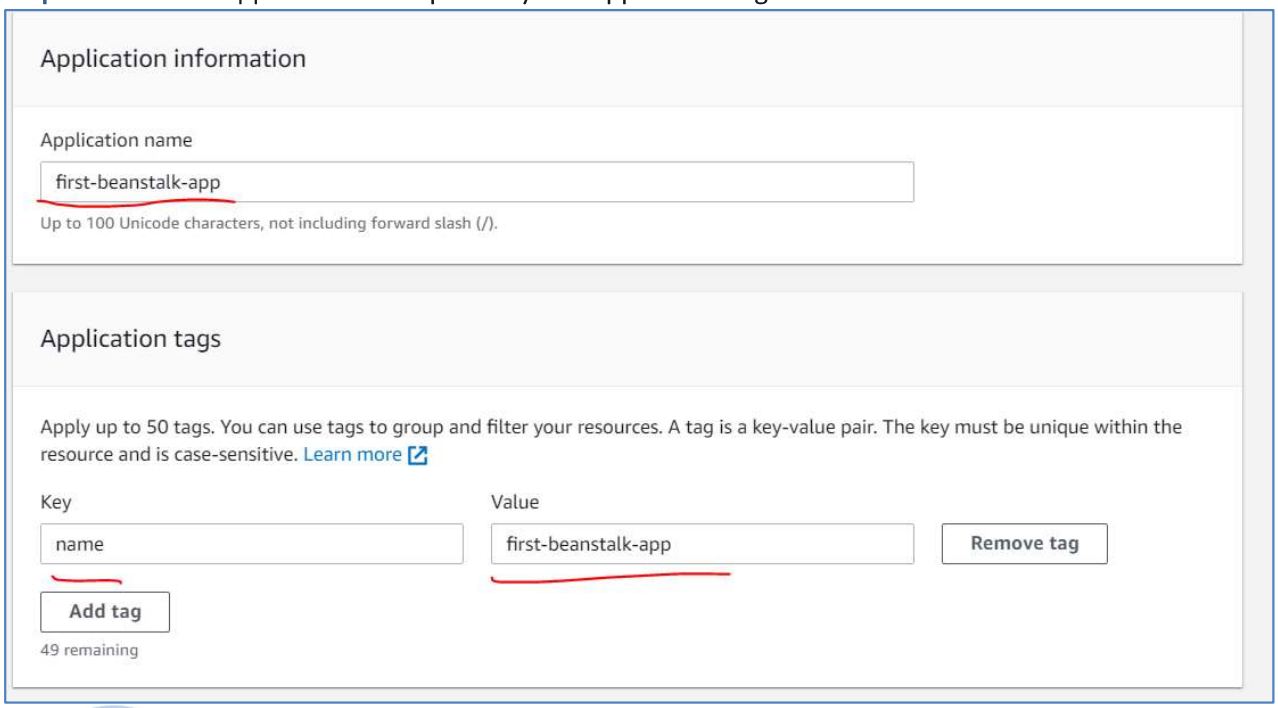
Create an application and an environment

To create your example application, you'll use the **Create a web app** console wizard. It creates an Elastic Beanstalk application and launches an environment within it. An environment is the collection of AWS resources required to run your application code.

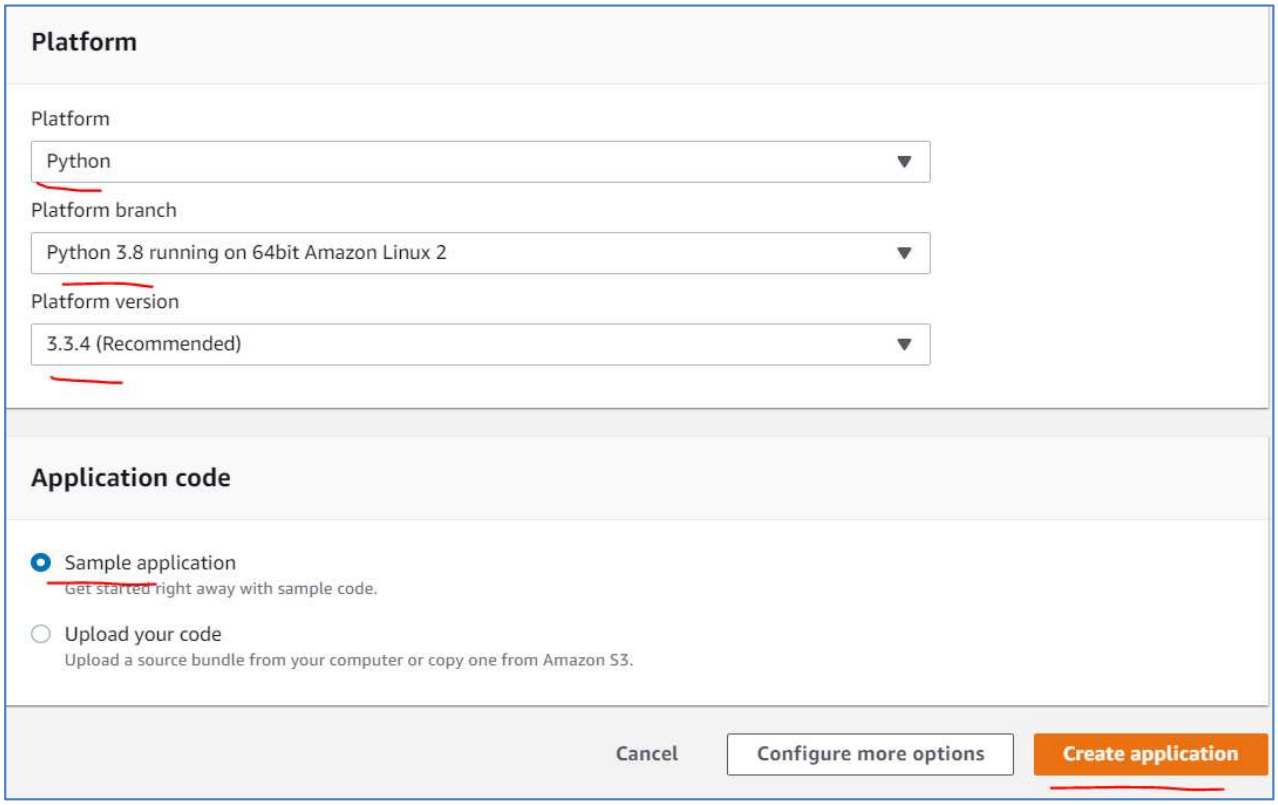
To create an application

Step 1. Open the Elastic Beanstalk console using this link: https://console.aws.amazon.com/elasticbeanstalk/home#/gettingStarted?applicationName=first-beanstalk-app

Step 2. Name the application and optionally add application tags.



Step 3. For **Platform**, choose a platform, and then choose **Create application**. If you have your own application code, then you can choose to deploy your own code too.

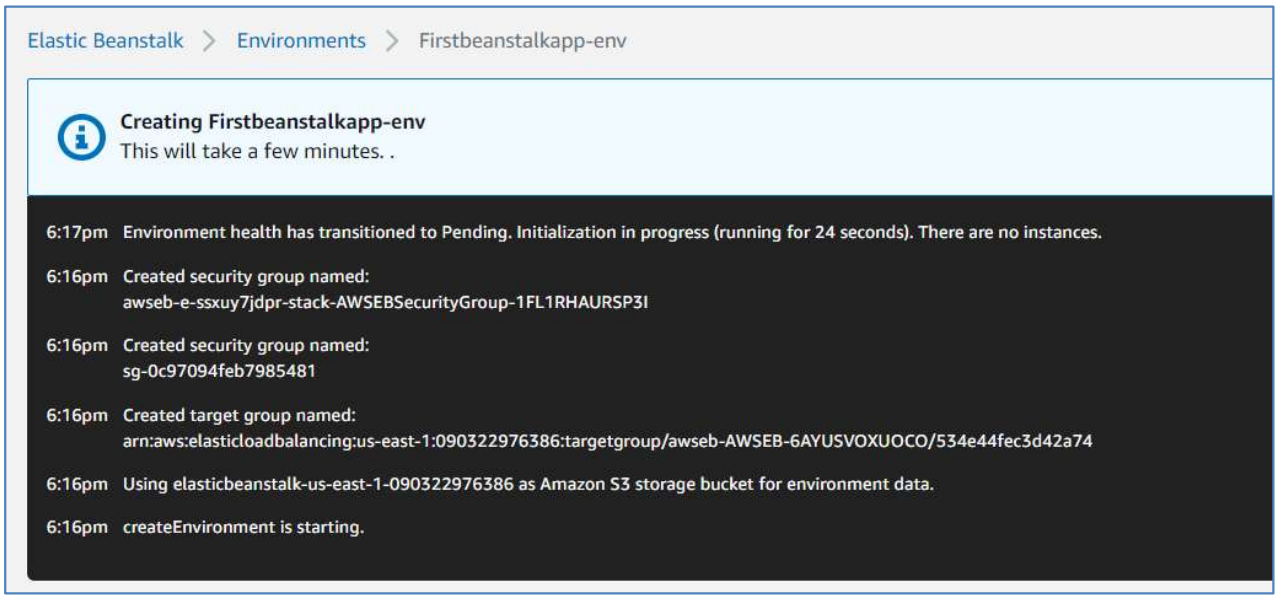


Note: To run the example application on AWS resources, Elastic Beanstalk takes the following actions. They take about five minutes to complete.

1. Creates an Elastic Beanstalk application named **first-beanstalk-app**.
2. Launches an environment named **GettingStartedApp-env** with these AWS resources:
    - An Amazon Elastic Compute Cloud (Amazon EC2) instance (virtual machine)
    - An Amazon EC2 security group
    - An Amazon Simple Storage Service (Amazon S3) bucket
    - Amazon CloudWatch alarms
    - An AWS CloudFormation stack
    - A domain name
3. Creates a new application version named **Sample Application**. This is the default Elastic Beanstalk example application file.

4. Deploys the code for the example application to the **GettingStartedApp-env** environment.

During the environment creation process, the console tracks progress and displays events.



When all of the resources are launched and the EC2 instances running the application pass health checks, the environment's health changes to Ok. You can now use your web application's website.

AWS resources created for the example application

When you create the example application, Elastic Beanstalk creates the following AWS resources:

- **EC2 instance** – An Amazon EC2 virtual machine configured to run web apps on the platform you choose.

  Each platform runs a different set of software, configuration files, and scripts to support a specific language version, framework, web container, or combination thereof. Most platforms use either Apache or nginx as a reverse proxy that processes web traffic in front of your web app, forwards requests to it, serves static assets, and generates access and error logs.
- **Instance security group** – An Amazon EC2 security group configured to allow incoming traffic on port 80. This resource lets HTTP traffic from the load balancer reach the EC2 instance running your web app. By default, traffic is not allowed on other ports.

- **Amazon S3 bucket** – A storage location for your source code, logs, and other artifacts that are created when you use Elastic Beanstalk.
- **Amazon CloudWatch alarms** – Two CloudWatch alarms that monitor the load on the instances in your environment and are triggered if the load is too high or too low. When an alarm is triggered, your Auto Scaling group scales up or down in response.
- **AWS CloudFormation stack** – Elastic Beanstalk uses AWS CloudFormation to launch the resources in your environment and propagate configuration changes. The resources are defined in a template that you can view in the AWS CloudFormation console.
- **Domain name** – A domain name that routes to your web app in the form *subdomain.region.elasticbeanstalk.com*.

Explore your environment

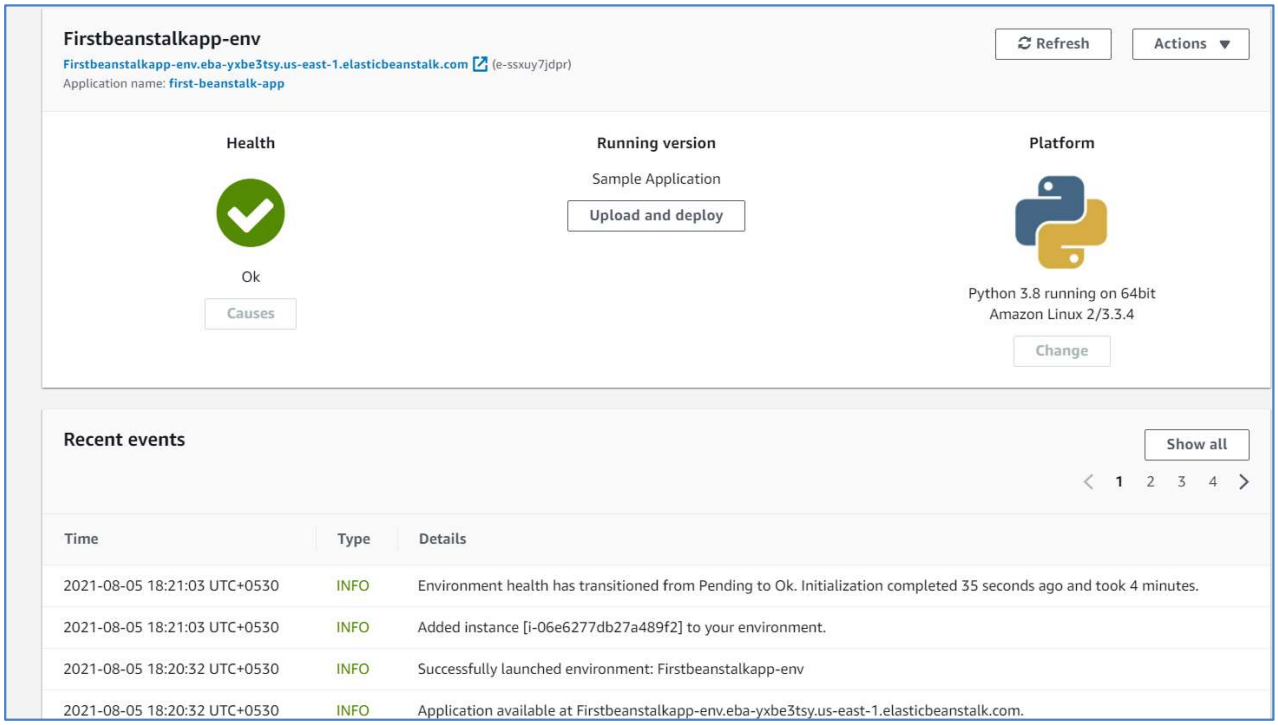
To see an overview of your Elastic Beanstalk application's environment, use the environment page in the Elastic Beanstalk console.

To view the environment overview

Step 1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

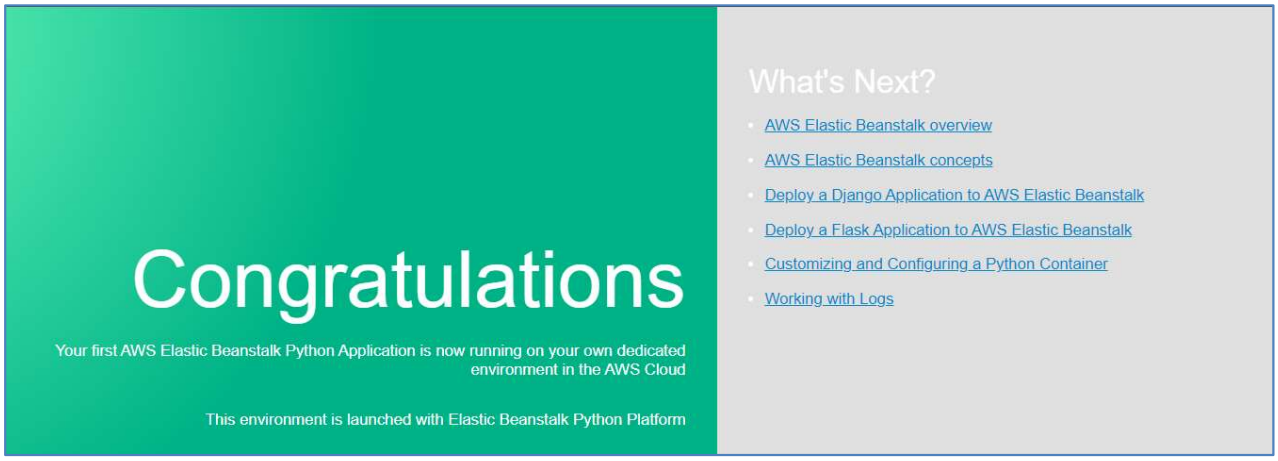
Step 2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

> **Note:** If you have many environments, use the search bar to filter the environment list. The environment overview pane shows top level information about your environment. This includes its name, its URL, its current health status, the name of the currently deployed application version, and the platform version that the application is running on. Below the overview pane you can see the five most recent environment events.

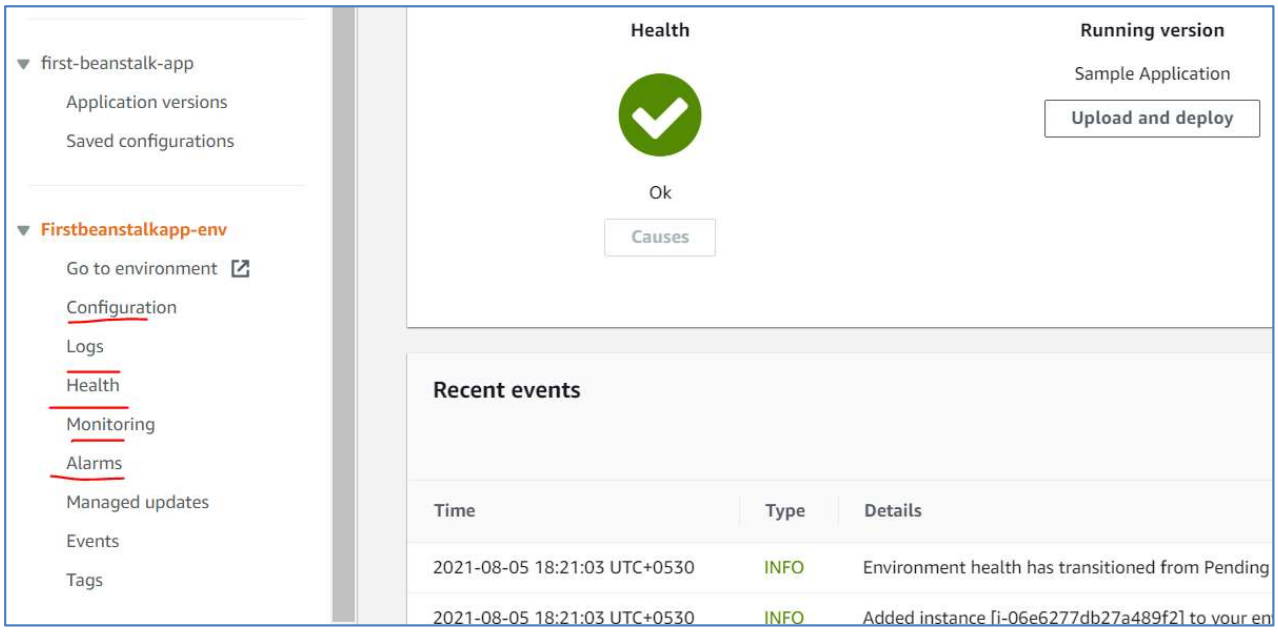


While Elastic Beanstalk creates your AWS resources and launches your application, the environment is in a Pending state. Status messages about launch events are continuously added to the overview.

The environment's **URL** is located at the top of the overview, below the environment name. This is the URL of the web application that the environment is running. Choose this URL to get to the example application's *Congratulations* page.



The navigation page on the left side of the console links to other pages that contain more detailed information about your environment and provide access to additional features:



- **Configuration** – Shows the resources provisioned for this environment, such as the Amazon Elastic Compute Cloud (Amazon EC2) instances that host your application. You can configure some of the provisioned resources on this page.
- **Health** – Shows the status of and detailed health information about the Amazon EC2 instances running your application.

- **Monitoring** – Shows statistics for the environment, such as average latency and CPU utilization. You can use this page to create alarms for the metrics that you are monitoring.
- **Events** – Shows information or error messages from the Elastic Beanstalk service and from other services whose resources this environment uses.
- **Tags** – Shows environment tags and allows you to manage them. Tags are key-value pairs that are applied to your environment.

Deploy a new version of your application

Periodically, you might need to deploy a new version of your application. You can deploy a new version at any time, as long as no other update operations are in progress on your environment.

The application version that you started this tutorial with is called **Sample Application**.

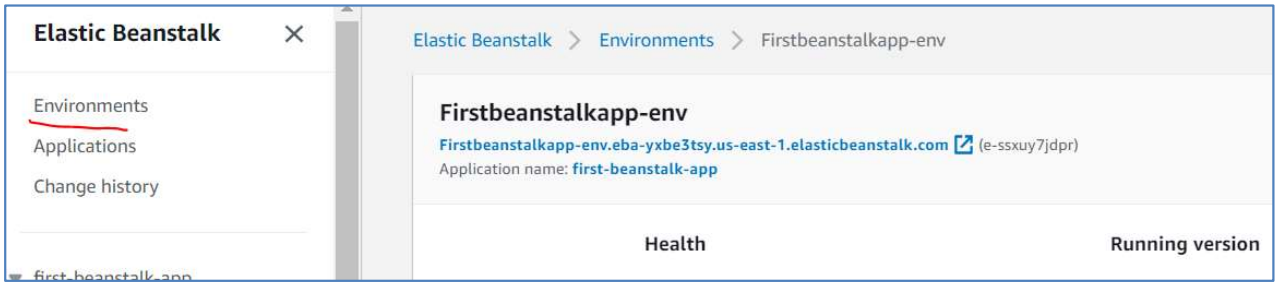
To update your application version

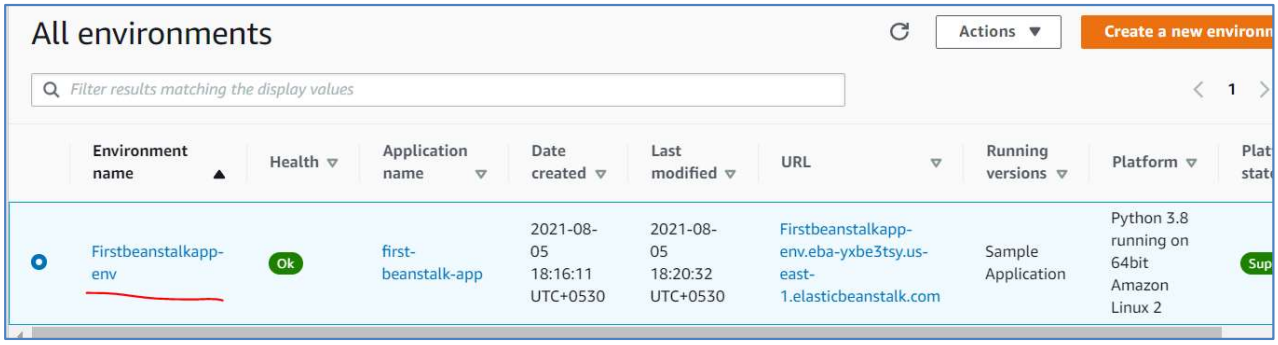
Step 1. Download the sample application that matches your environment's platform. Use one of the following applications. **We will go with python.zip here, because previous version of app we deployed is same.**

- **Docker** – docker.zip
- **Multicontainer Docker** – docker-multicontainer-v2.zip
- **Preconfigured Docker (Glassfish)** – docker-glassfish-v1.zip
- **Go** – go.zip
- **Corretto** – corretto.zip
- **Tomcat** – tomcat.zip
- **.NET Core on Linux** – dotnet-core-linux.zip
- **.NET** – dotnet-asp-v1.zip
- **Node.js** – nodejs.zip
- **PHP** – php.zip
- **Python** – python.zip (Download URL: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/samples/python.zip)
- **Ruby** – ruby.zip

Step 2. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

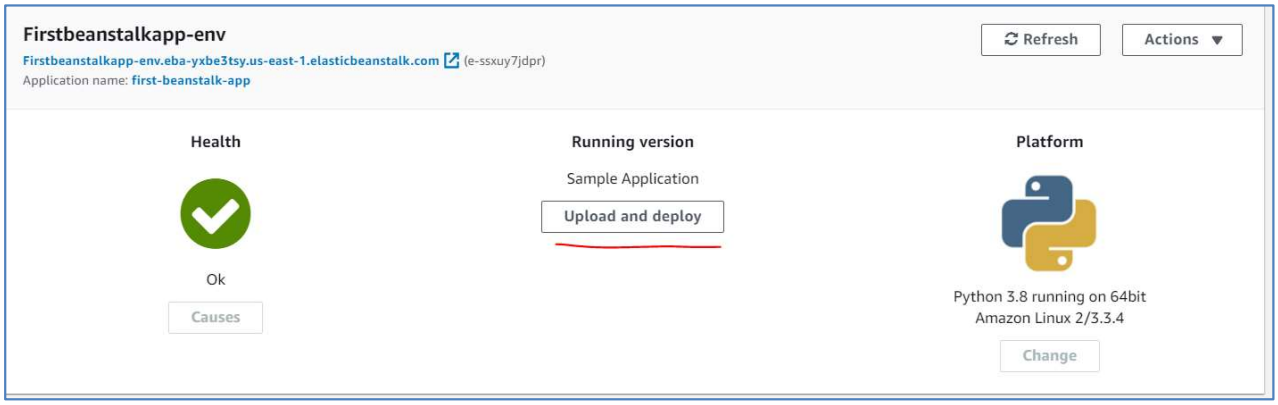
Step 3. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.



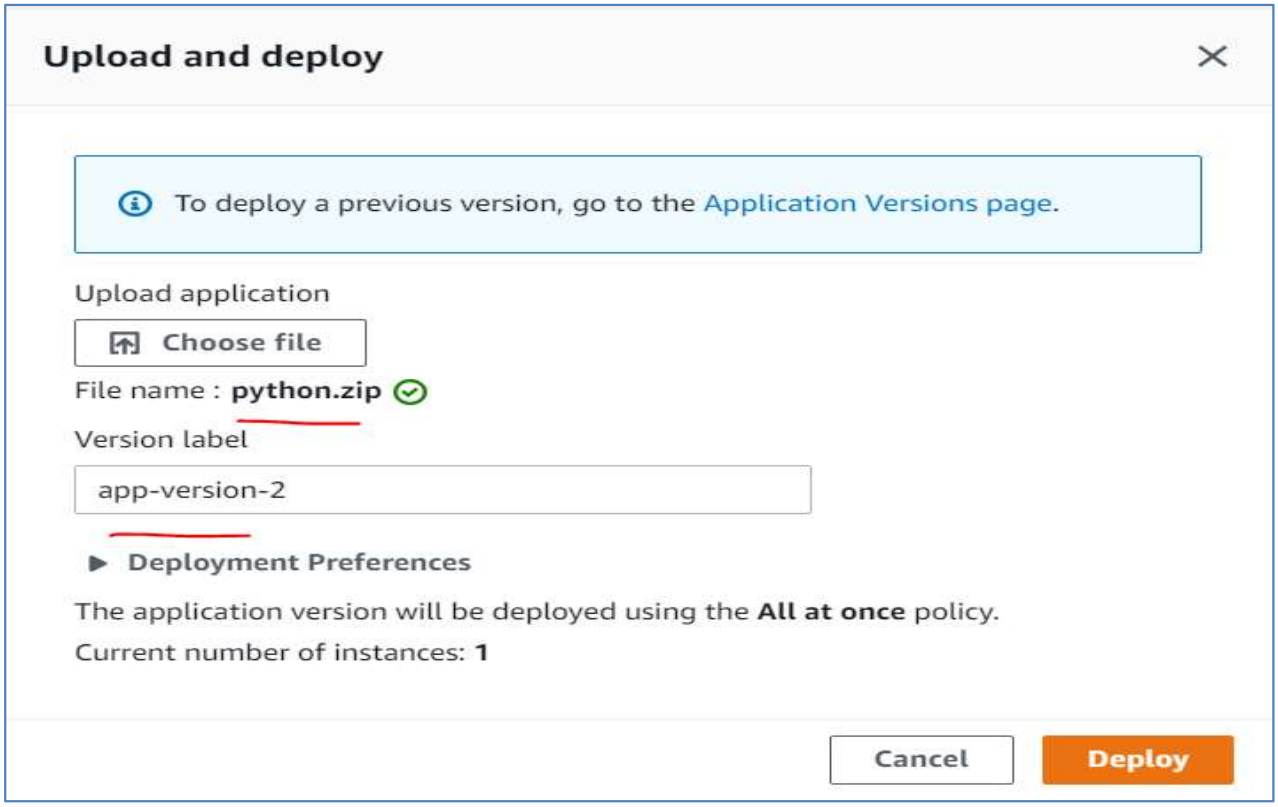


Note: If you have many environments, use the search bar to filter the environment list.

Step 4. On the environment overview page, choose **Upload and deploy**.



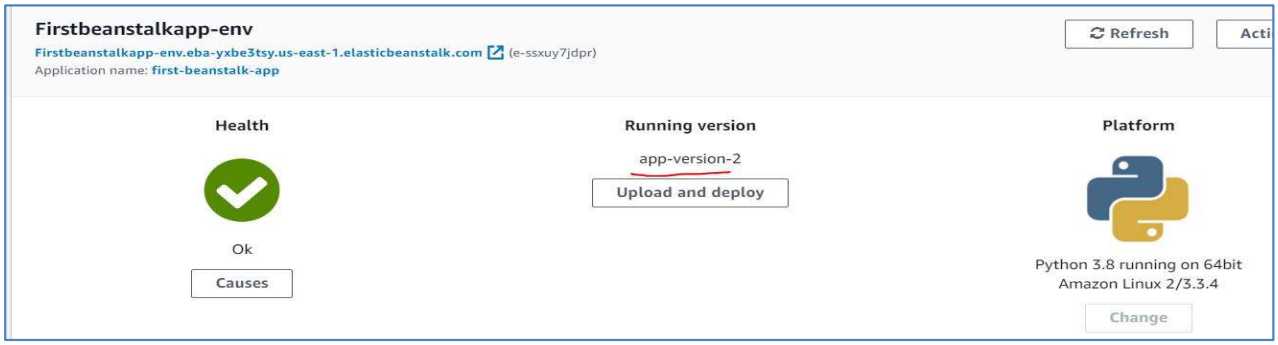
Step 5. Choose **Choose file**, and then upload the sample application source bundle that you downloaded.



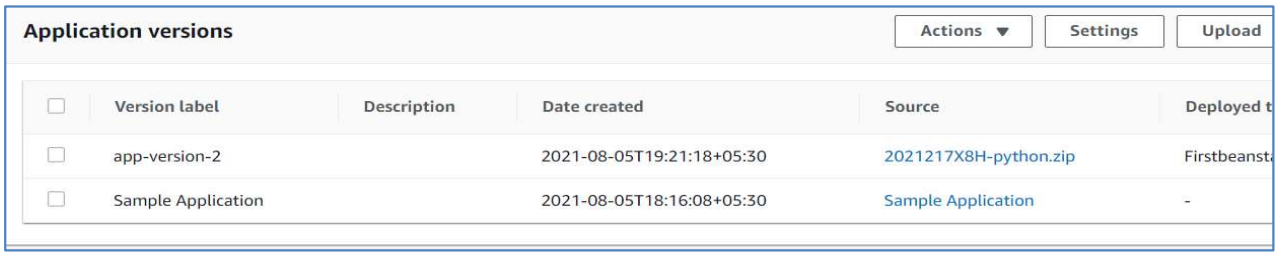
The console automatically fills in the **Version label** with a new unique label. If you type in your own version label, ensure that it's unique.

Step 6. Choose **Deploy**.

While Elastic Beanstalk deploys your file to your Amazon EC2 instances, you can view the deployment status on the environment's overview. While the application version is updated, the **Environment Health** status is gray. When the deployment is complete, Elastic Beanstalk performs an application health check. When the application responds to the health check, it's considered healthy and the status returns to green. The environment overview shows the new **Running Version**—the name you provided as the **Version label**.



Elastic Beanstalk also uploads your new application version and adds it to the table of application versions. To view the table, choose **Application versions** under **first-beanstalk-app** on the navigation pane.



Configure your environment

You can configure your environment to better suit your application. For example, if you have a compute-intensive application, you can change the type of Amazon Elastic Compute Cloud (Amazon EC2) instance that is running your application. To apply configuration changes, Elastic Beanstalk performs an environment update.

Some configuration changes are simple and happen quickly. Some changes require deleting and recreating AWS resources, which can take several minutes. When you change configuration settings, Elastic Beanstalk warns you about potential application downtime.

Make a configuration change

In this example of a configuration change, you edit your environment's capacity settings. You configure a load-balanced, scalable environment that has between two and four Amazon EC2 instances in its Auto Scaling group, and then you verify that the change occurred. Elastic Beanstalk creates an additional Amazon EC2 instance, adding to the single instance that it created initially. Then, Elastic Beanstalk associates both instances with the environment's load balancer. As a result, your application's responsiveness is improved and its availability is increased.

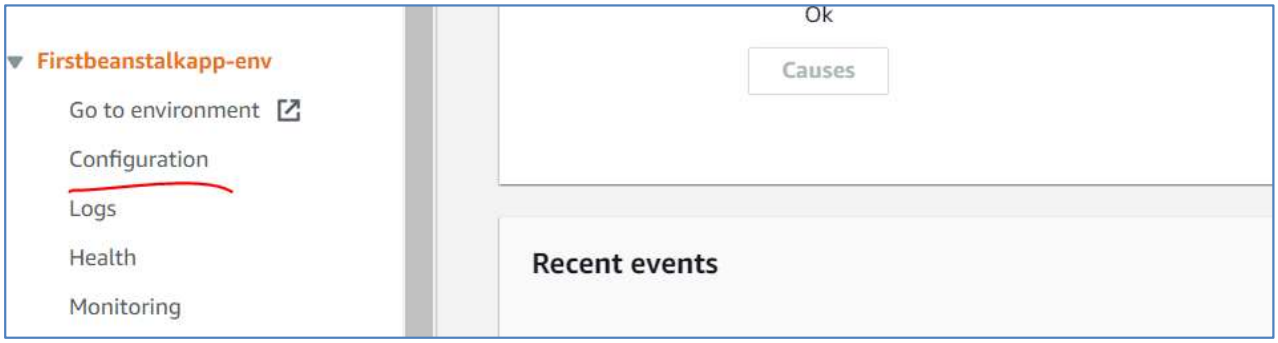
To change your environment's capacity

Step 1. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

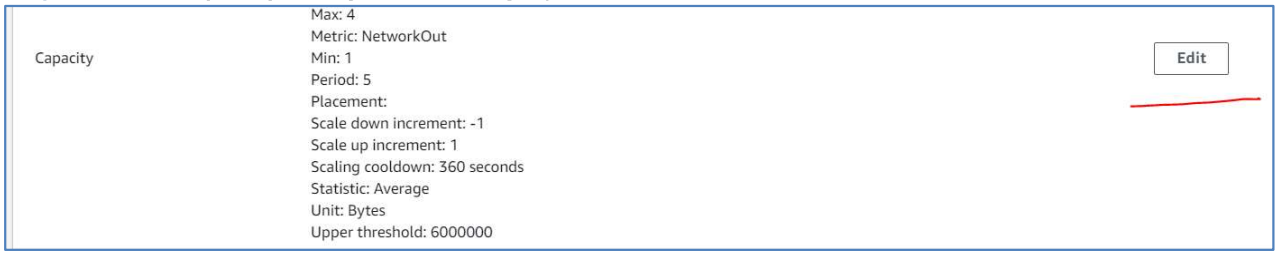
Step 2. In the navigation pane, choose **Environments**, and then choose the name of your environment from the list.

Note: If you have many environments, use the search bar to filter the environment list.

Step 3. In the navigation pane, choose **Configuration**.

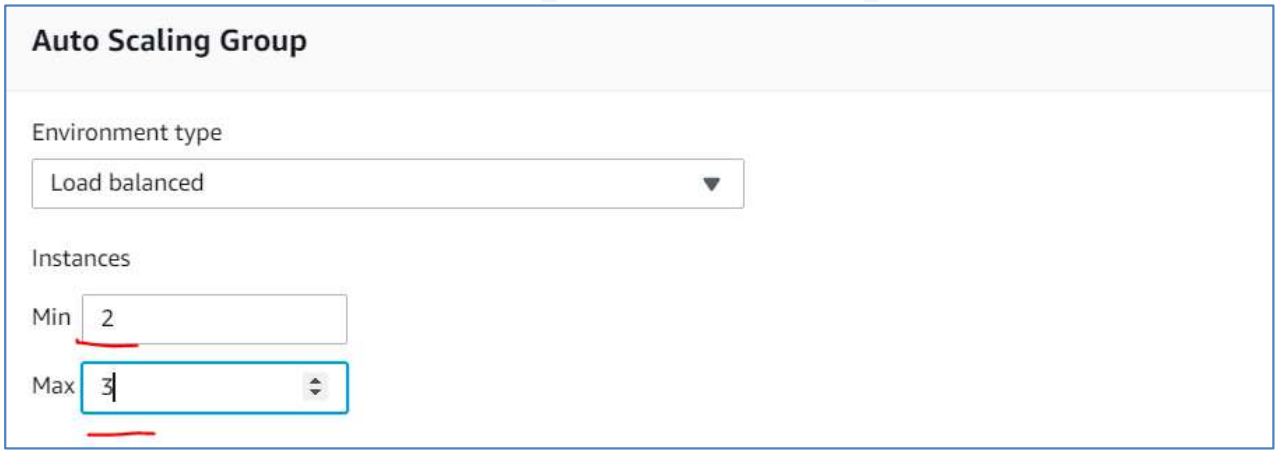


Step 4. In the **Capacity** configuration category, choose **Edit**.



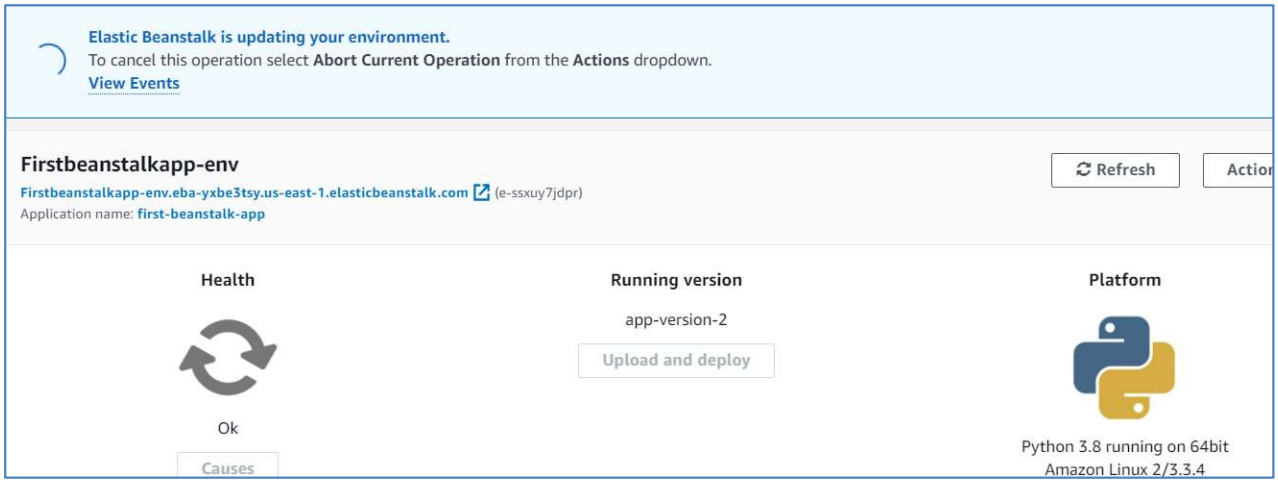
Step 5. In the **Auto Scaling group** section, change **Environment type** to **Load balanced**.

Step 6. In the **Instances** row, change **Max** to **3**, and then change **Min** to **2**.



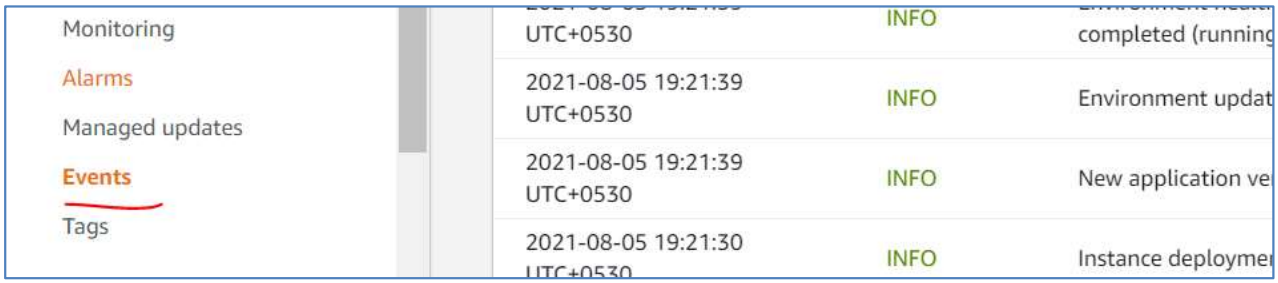
Step 7. Choose **Apply**.

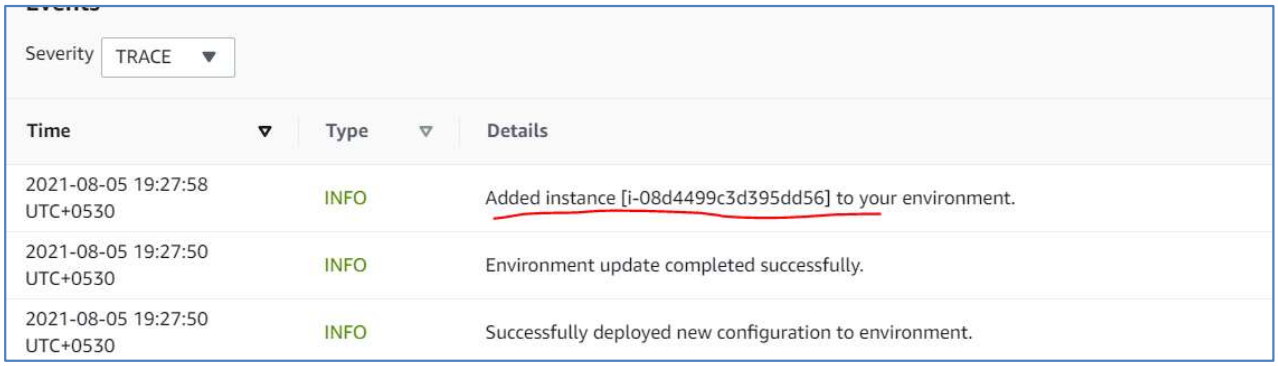
Step 8. If a warning tells you that this update replaces all of your current instances. Choose **Confirm**.



Step 9. In the navigation pane, choose **Events**.

The environment update can take a few minutes. To find out that it's complete, look for the event **Successfully deployed new configuration to environment** in the event list. This confirms that the Auto Scaling minimum instance count has been set to 2. Elastic Beanstalk automatically launches the second instance.





Verify the configuration change

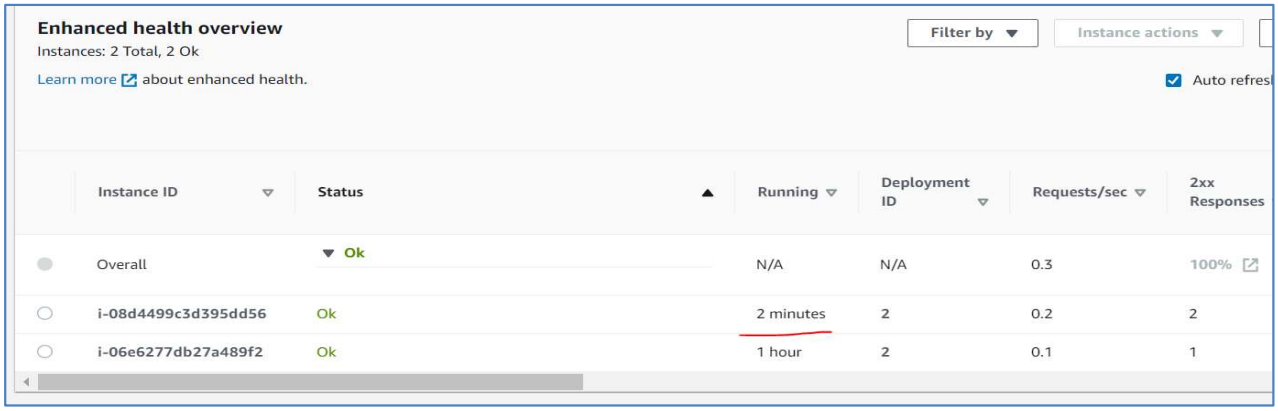
When the environment update is complete and the environment is ready, verify your change.

To verify the increased capacity

Step 1. In the navigation pane, choose **Health**.

Step 2. Look at the **Enhanced health overview** page.

You can see that two Amazon EC2 instances are listed following the **Overall** line. Your environment capacity has increased to two instances.



To delete the application and all associated resources

Step 1. Delete all application versions.

a. Open the Elastic Beanstalk console, and in the **Regions** list, select your AWS Region.

b. In the navigation pane, choose **Applications**, and then choose **first-beanstalk-app**.

c. In the navigation pane, find your application's name and choose **Application versions**.

d. On the **Application versions** page, select all application versions that you want to delete.

e. Choose **Actions**, and then choose **Delete**.

f. Turn on **Delete versions from Amazon S3**.

g. Choose **Delete**, and then choose **Done**.

Step 2. Terminate the environment.

a. In the navigation pane, choose **first-beanstalk-app**, and then choose **GettingStartedApp-env** in the environment list.

b. Choose **Environment actions**, and then choose **Terminate Environment**.

c. Confirm that you want to terminate **GettingStartedApp-env** by typing the environment name, and then choose **Terminate**.

Step 3. Delete the **first-beanstalk-app** application.

a. In the navigation pane, choose the **first-beanstalk-app**.

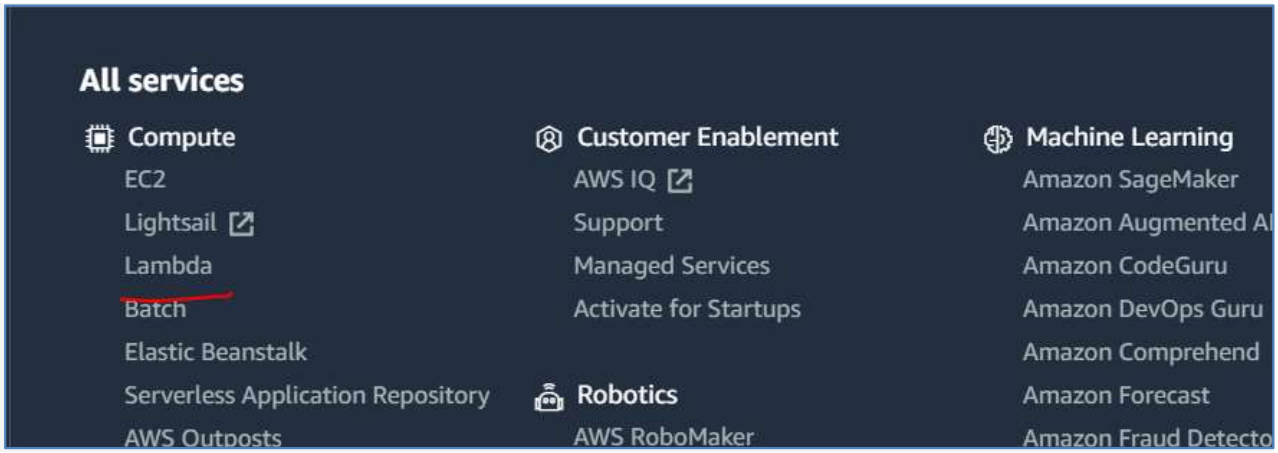
b. Choose **Actions**, and then choose **Delete application**.

c. Confirm that you want to delete **first-beanstalk-app** by typing the application name, and then choose **Delete**.

Using AWS Lambda to create serverless functions

Enter the Lambda Console

Step 1. Click https://console.aws.amazon.com/console/home to open AWS Management Console.
Find Lambda under Compute and click to open the AWS Lambda Console.

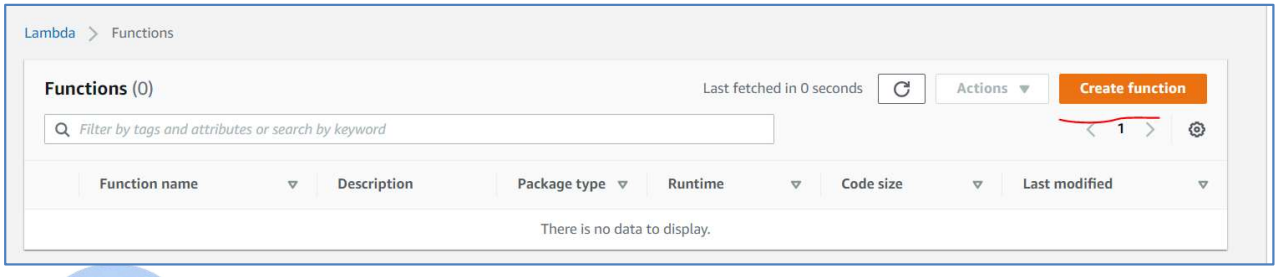


Select a Lambda Blueprint

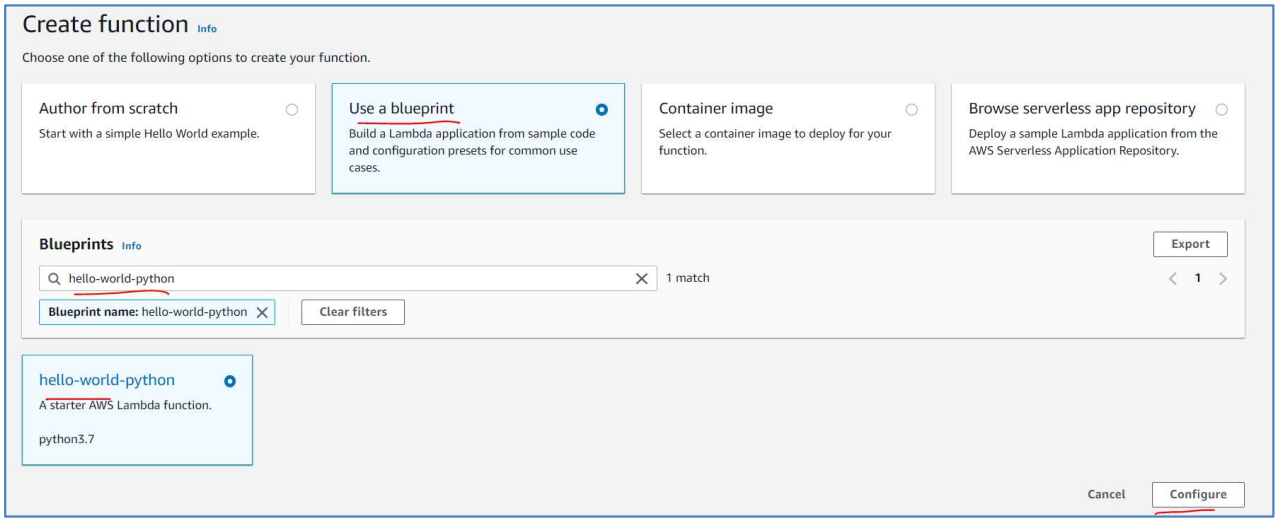
Blueprints provide example code to do some minimal processing. Most blueprints process events from specific event sources, such as Amazon S3, DynamoDB, or a custom application.

Step 1. 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.



Step 2. Select Blueprints.
Step 3. In the Filter box, type in hello-world-python and select the hello-world-python blueprint
Step 4. Then click Configure.



Configure and Create your Lambda Function

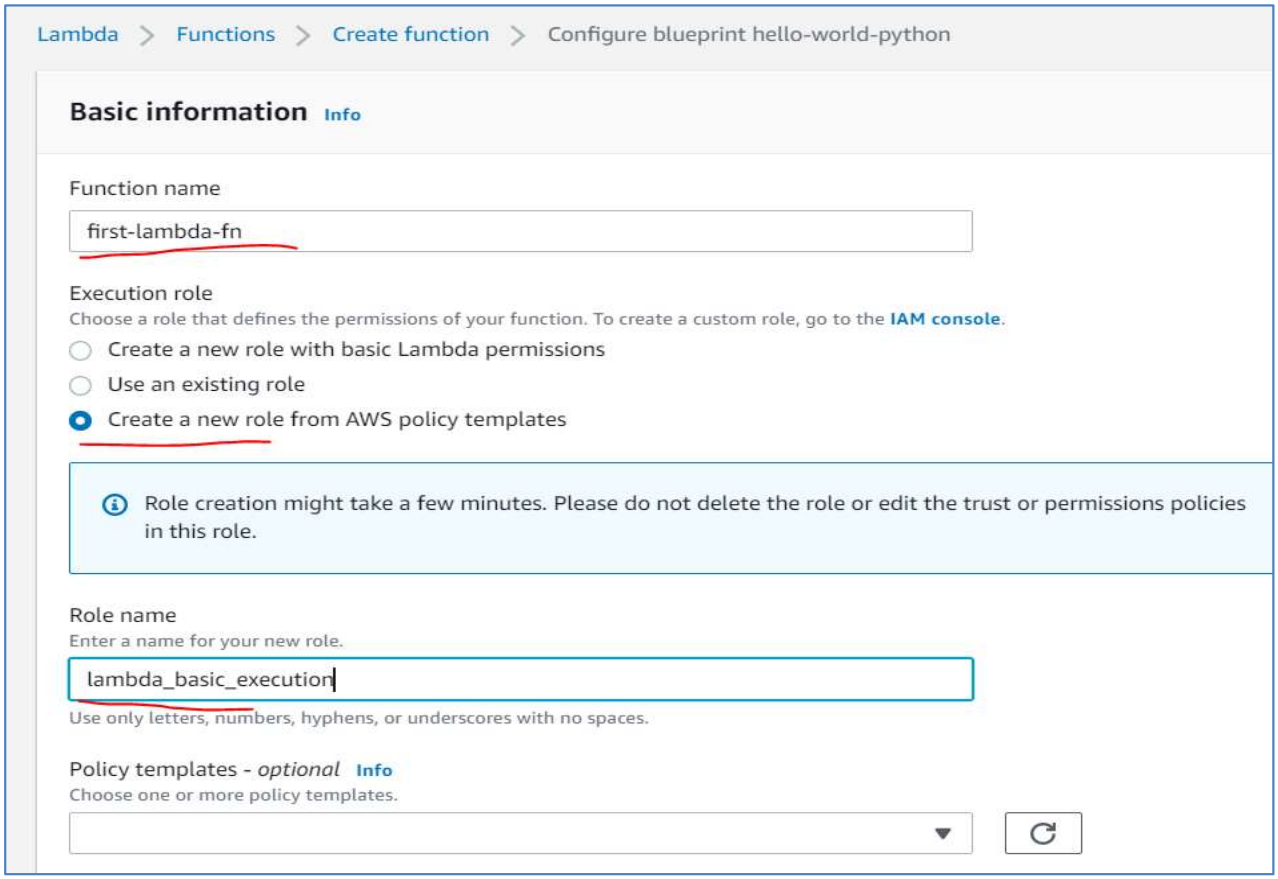
A Lambda function consists of code you provide, associated dependencies, and configuration. The configuration information you provide includes the compute resources you want to allocate (for example, memory), execution timeout, and an IAM role that AWS Lambda can assume to execute your Lambda function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

Step 1. You will now enter Basic Information about your Lambda function.

Basic Information:

- **Name:** You can name your Lambda function here. For this tutorial, enter **first-lambda-fn**.
- **Role:** You will 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. Select Create new role from template(s).
- **Role name: lambda_basic_execution**
- **Lambda Function Code:** In this section, you can review the example code authored in Python.

Step 2. Go to the bottom of the page and select Create Function.

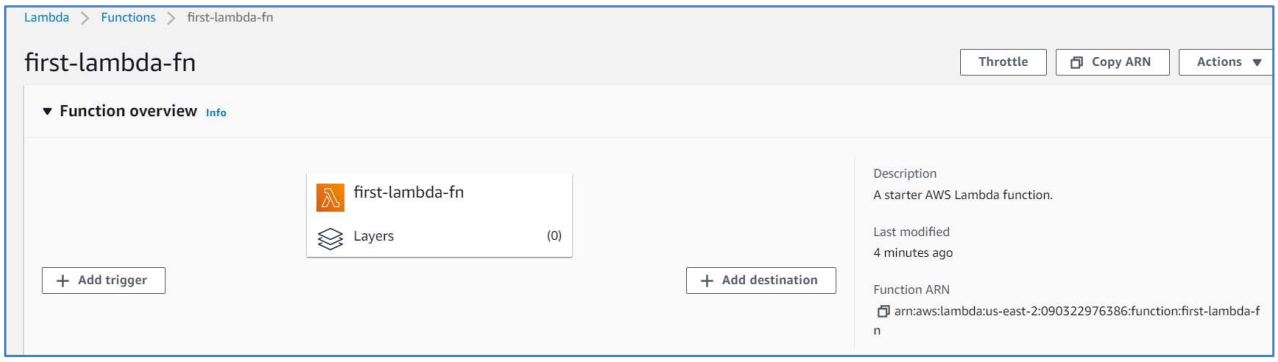

Lambda > Functions > Create function > Configure blueprint hello-world-python
Basic information Info
Function name
first-lambda-fn
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
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.
lambda_basic_execution
Use only letters, numbers, hyphens, or underscores with no spaces.
Policy templates - optional Info
Choose one or more policy templates.

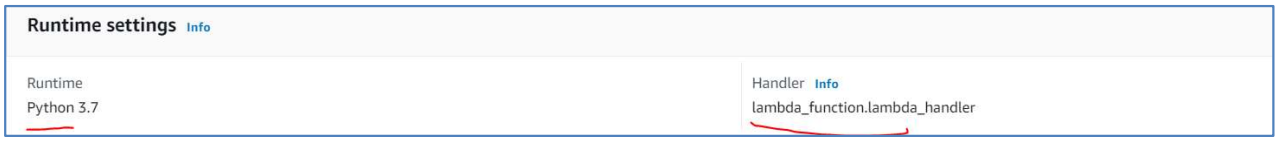

Lambda function code
Code is preconfigured by the chosen blueprint. You can configure it after you create the function. Learn more about deploying Lambda functions.
Runtime
Python 3.7
import json
print('Loading function')
def lambda_handler(event, context):
#print("Received event: " + json.dumps(event, 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')

Step 3. In Runtime settings you can see below parameters :

- **Runtime:** Currently, you can author your Lambda function code in Java, Node.js, C#, Go or Python. For this tutorial, leave this on Python 2.7 as the runtime.
- **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**.





Step 4. Check configuration tab for your memory, timeout, and VPC settings. For this tutorial, leave the default Lambda function configuration values.

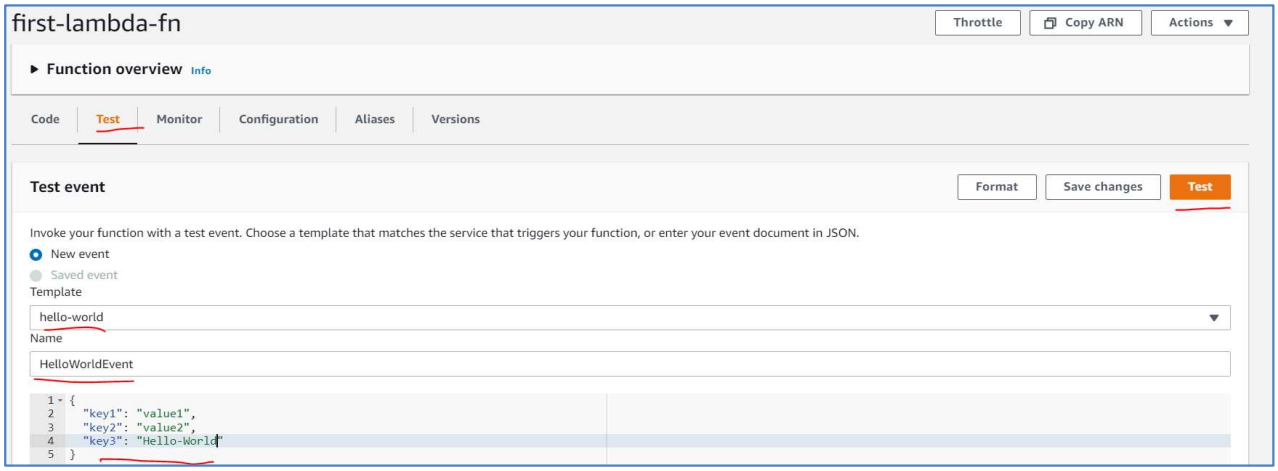
Invoke Lambda Function and Verify Results

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

Step 1. Click on **Test** tab and Choose **Hello World** from the **Sample event template** list from the Input test event page.

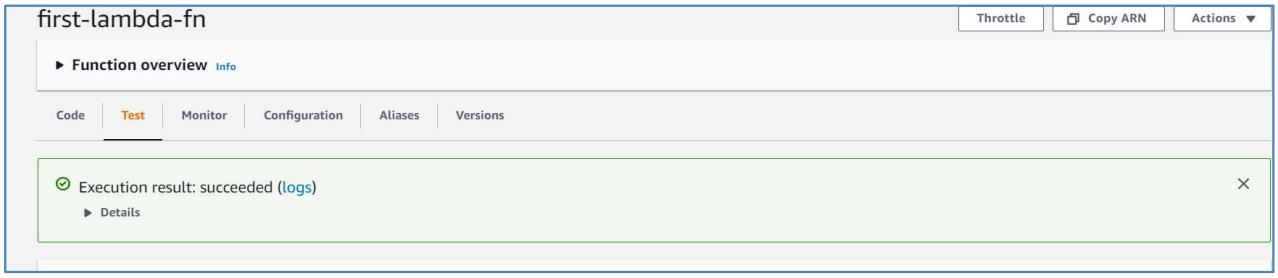
Step 2. Type in an event name like **HelloWorldEvent**.

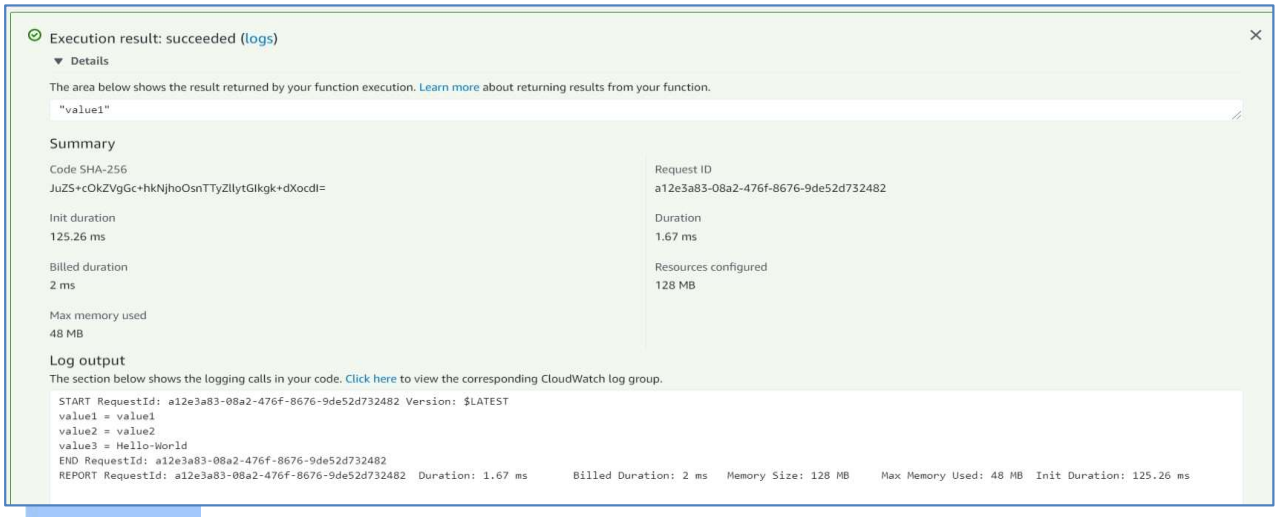
Step 3. You can change the values in the sample JSON, but don't change the event structure. For this tutorial, replace value1 with hello, world!. Select **Save Changes**.



Step 4. Select **Test**.

Step 5. 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 as below:





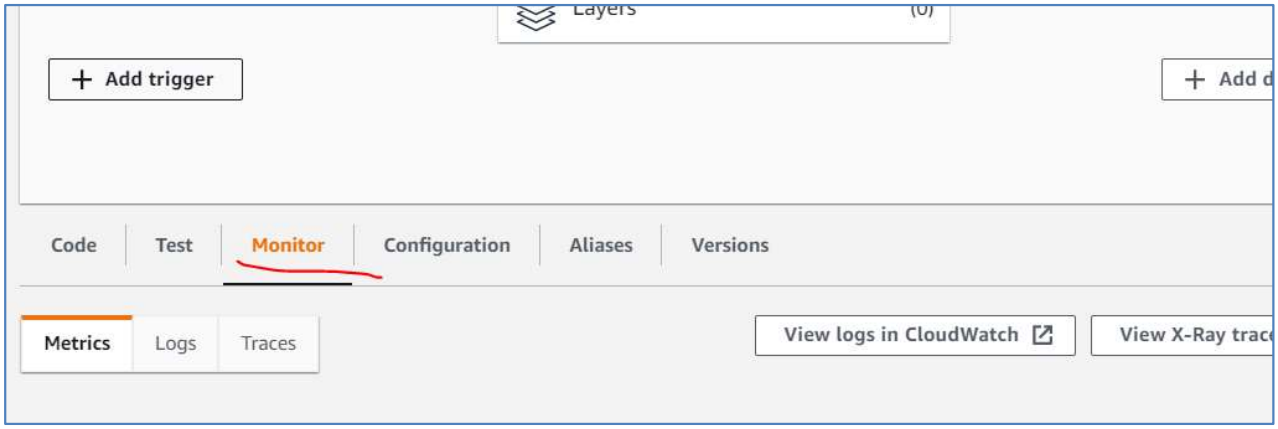
The Log output section will show the logs generated by the Lambda function execution.

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.

Step 1. 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.

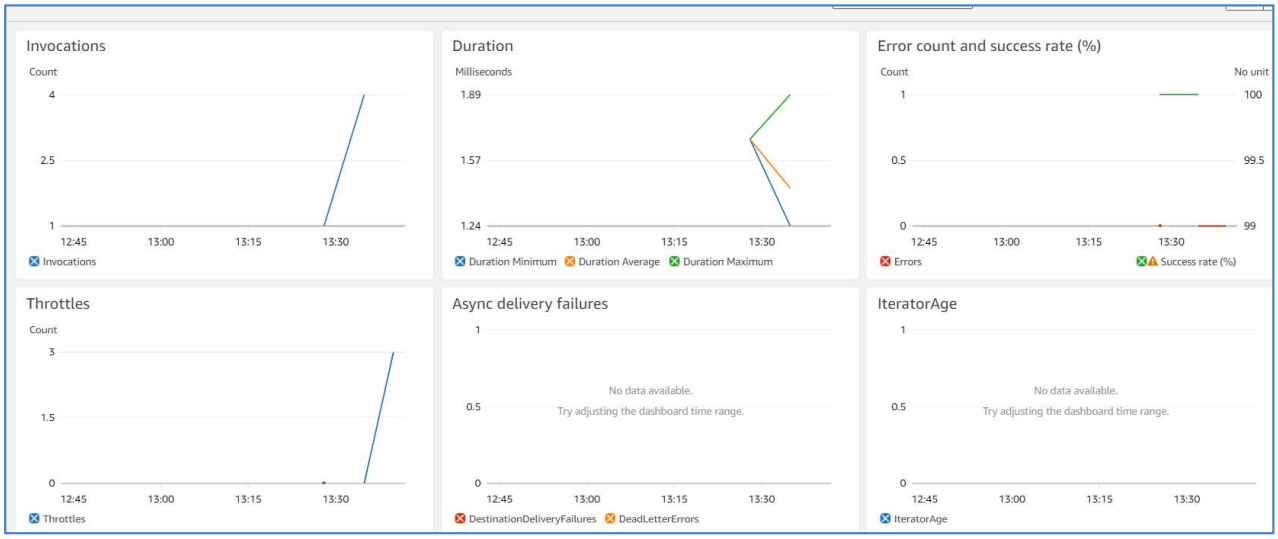
Step 2. Select **Monitor** tab to view the results.



Step 3. Scroll down to view the metrics for your Lambda function. Lambda metrics are reported through Amazon CloudWatch. You can leverage these metrics to set custom alarms

Step 4. The Monitoring tab will show six CloudWatch metrics: Invocation count, Invocation duration, Invocation errors, Throttled invocations, Iterator age, and DLQ errors.

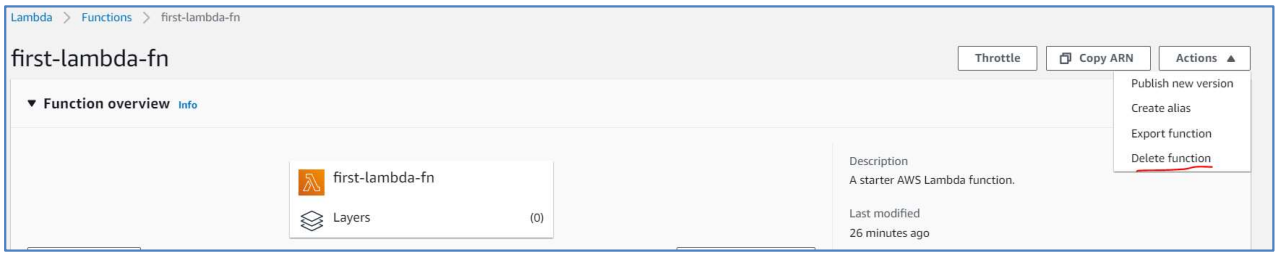
With AWS Lambda, you pay for what you use. After you hit your AWS Lambda free tier limit, you are charged based on the number of requests for your functions (invocation count) and the time your code executes (invocation duration).



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.

Step 1. Select the Actions button and click Delete Function.



Step 2. You will be asked to confirm your termination - select Delete.