[Deployment 1 Documentation]

Author: Suborna Nath

Date: September 1, 2022

Description: In this deployment, we are setting up a pipeline to deploy a flask application "url-shortner" from a GitHub repository using EC2, Elastic Beanstalk, and Jenkins.

There are 6 steps involved with this deployment.

1. Create an EC2 on AWS
2. SSH into the EC2
3. Download all the dependencies
4. Configure Jenkins
5. Configure the pipeline, build & test
6. Deploy the application on Elastic Beanstalk

Step 1 >> Create an EC2 on AWS

The purpose of creating an EC2 is to containerize the application and avoid configuration drift. We will be logging into our console at https://aws.amazon.com/. Then, we will go to EC2 > Instances > Launch Instance. We will fill in the configurations as shown below.

Name: Kura_Deployment_1 | **Application & OS Images:** Ubuntu | **Instance Type:** t2.micro
Amazon Machine Image (AMI): Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
Key pair (login): select an existing pair or create a new one with pair type RSA and file format .pem. Make sure to download the keypair which you will need to ssh into the EC2.
Network Settings: For inbound traffic, we need to create a security group with port type **ssh, http, custom TCP** (the web server), and port numbers **22, 80, 8080**, respectively. Source type for outbound traffic can be anywhere (any IP can access my instance) or my IP (only I can access my instance).

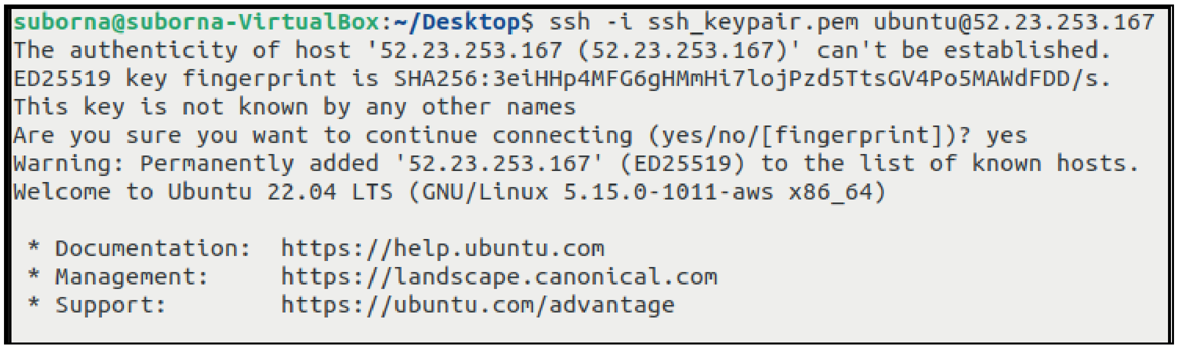
We need to open port 22 so that we can ssh into our EC2, port 80 to have access to http (needed to communicate with Jenkins and the application) and port 8080 to access Jenkins server. If we do not have the ports open, we might run into problems with accessing our EC2 from the terminal and accessing Jenkins.

Step 2 >> SSH into the EC2 from the terminal

To SSH into the EC2, we will *cd* into the directory where you saved your .pem file for the key-pair and run the following command. We also need to change the permission of the .pem file to give permissions to only the users. Otherwise, the permissions are too open causing a security risk and we will not be able to access the EC2 from the terminal.

```
$chmod 400 keypair.pem
$ssh -i ssh_keypair.pem ubuntu@public_IP_of_the_EC2
```

When prompted, type *yes* and press *Enter*. We should have a welcome screen similar to the screenshot. The prompt in the terminal will change to *ubuntu@your_private_IP:~$*



```
suborna@suborna-VirtualBox:~/Desktop$ ssh -i ssh_keypair.pem ubuntu@52.23.253.167
The authenticity of host '52.23.253.167 (52.23.253.167)' can't be established.
ED25519 key fingerprint is SHA256:3eiHHp4MFG6gHMmHi7lojPzd5TtsGV4Po5MAWdFDD/s.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.23.253.167' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-1011-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

Step 3 >> Download all the dependencies on the EC2

For this step, we need to install the tools needed to build and test the application. To build the application, we will use JRE and Jenkins. To test, we will use Python and Python virtual environment (venv). Since Jenkins runs on Java, we need an environment to execute Jenkins. JRE (Java Runtime Environment) provides a Java environment to run Jenkins. We will run the below commands to install JRE and Jenkins. Python environment will provide all the dependencies (Flask, pytest) to test the application. If we do not have the environment, our test will fail. Also, if we do not have Jenkins installed, we will not be able to access its server and build the pipeline.

1. Install the Python environment
$sudo apt install python3-pip
$sudo apt install python3-venv

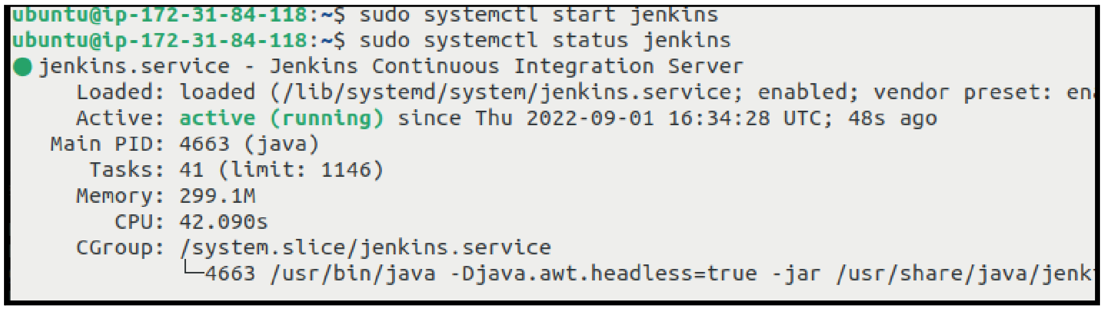
2. Update the repository and install JRE
$sudo apt update && sudo apt install default-jre -y

3. Get all the files for installation of Jenkins
$wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo gpg --dearmor -o/usr/share/keyrings/jenkins.gpg
$sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

4. Update the repository and install Jenkins
$sudo apt update && sudo apt install jenkins -y

5. Start Jenkins and check the status to make sure it is running
$sudo systemctl start jenkins
$sudo systemctl status jenkins

The terminal should have an output with status active and running as shown in the screenshot. We need to make sure that Jenkins is up and running in order to access the server from a browser.

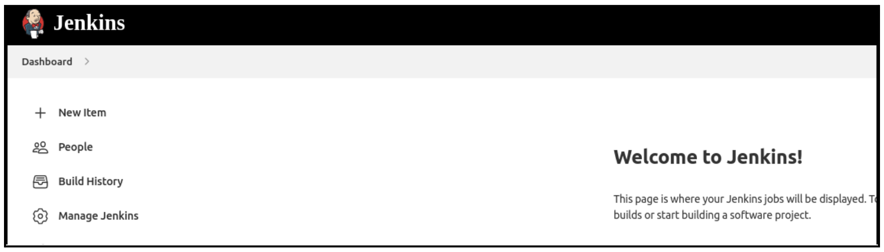


```
ubuntu@ip-172-31-84-118:~$ sudo systemctl start jenkins
ubuntu@ip-172-31-84-118:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
     Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: en
     Active: active (running) since Thu 2022-09-01 16:34:28 UTC; 48s ago
   Main PID: 4663 (java)
      Tasks: 41 (limit: 1146)
     Memory: 299.1M
        CPU: 42.090s
     CGroup: /system.slice/jenkins.service
             └─4663 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenk
```

Step 4 >> Configure Jenkins

Since Jenkins is running in the terminal, we can open up the server in a browser. The server address is *http://public_IP_of_EC2:8080.* We will need the admin password created by Jenkins in the EC2 to proceed with the configuration. The password can be found with the below command. The output will be the password and we can copy/paste it inside the box under *Administrator password* and *continue.*

$sudo cat /var/lib/jenkins/secrets/initialAdminPassword



This page is where your Jenkins jobs will be displayed. T
builds or start building a software project.

Once we install the suggested plugins, create the user, and finish the setup, we should have a welcome screen similar to the screenshot.

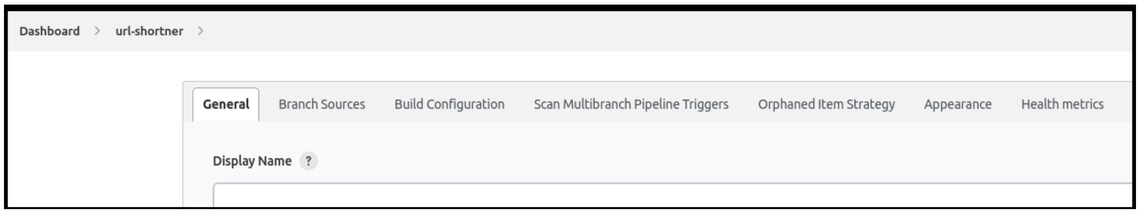
Since the server is inside the EC2, we will need to have the EC2 running to access Jenkins server. Once we stop/terminate our EC2, we can no longer access the server. If we exit the EC2 from the terminal, we can still access the server because the EC2 is still running on AWS.

Step 5 >> Configure the Pipeline on Jenkins | Build & Test

First, we will head over to Github to create a token. The token is needed to connect Github with Jenkins.

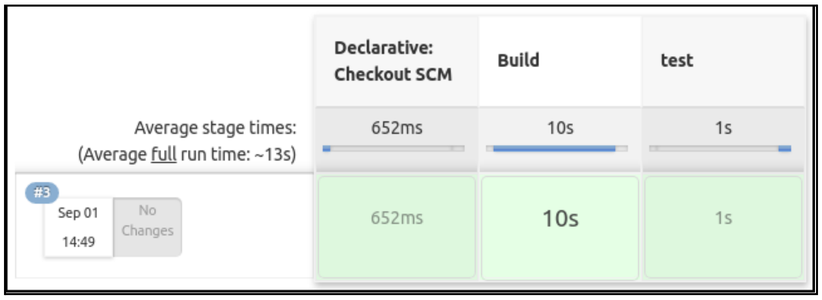
Github > *Settings* > *Developers Settings* > *Personal Access Tokens* > *Generate new token*. Select repo, admin:repo_hook and generate the token. Make sure to copy the token.

Back in **Jenkins**, we will create a new pipeline with name “url-shortner”. Click on *New Item*, add item name, select *Multibranch Pipeline* and hit *OK.* The page shoul look something like this screenshot.



We will use the below configurations to set up the pipeline.

- General → **Display Name:** Build Flask App | **Description:** CI/CD Deployment 1
- Branch Source → **Add Source:** GitHub
    - **Github Credentials:** Add > Jenkins (Jenkins will use Github API to validate the credentials)
        - Enter the proper GitHub username and password (access token created in Github)
    - **Repository HTTPS URL**: link to the GitHub repository > Validate
- Build Configuration → **Mode:** by Jenkinsfile | **Script Path:** Jenkinsfile



	Declarative: Checkout SCM	Build	test
Average stage times: (Average full run time: ~13s)	652ms	10s	1s
#3 Sep 01 14:49 No Changes	652ms	10s	1s

The build will start automatically and should be successful similar to the screenshot (green for both build/test).
If our build or test fails, we can look at the *console logs* for the build to figure out what went wrong. If our test fails, one reason can

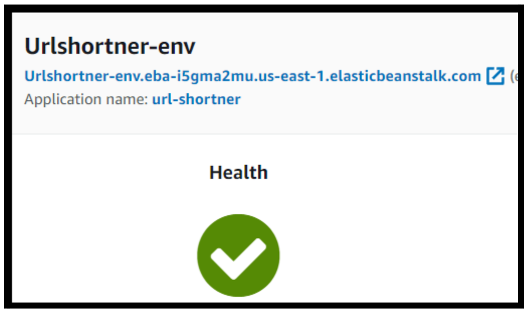
be that the environment for the test was not created.

Step 6 >> Deploy the Flask Application on Elastic Beanstalk

Since our build and test ran successfully, we know that our application is ready for deployment to production. We will clone the GitHub repo, *cd* into it and create an artifact for elastic beanstalk by zipping all the necessary files in the EC2. We will be uploading the zip file on Elastic Beanstalk to deploy the app.

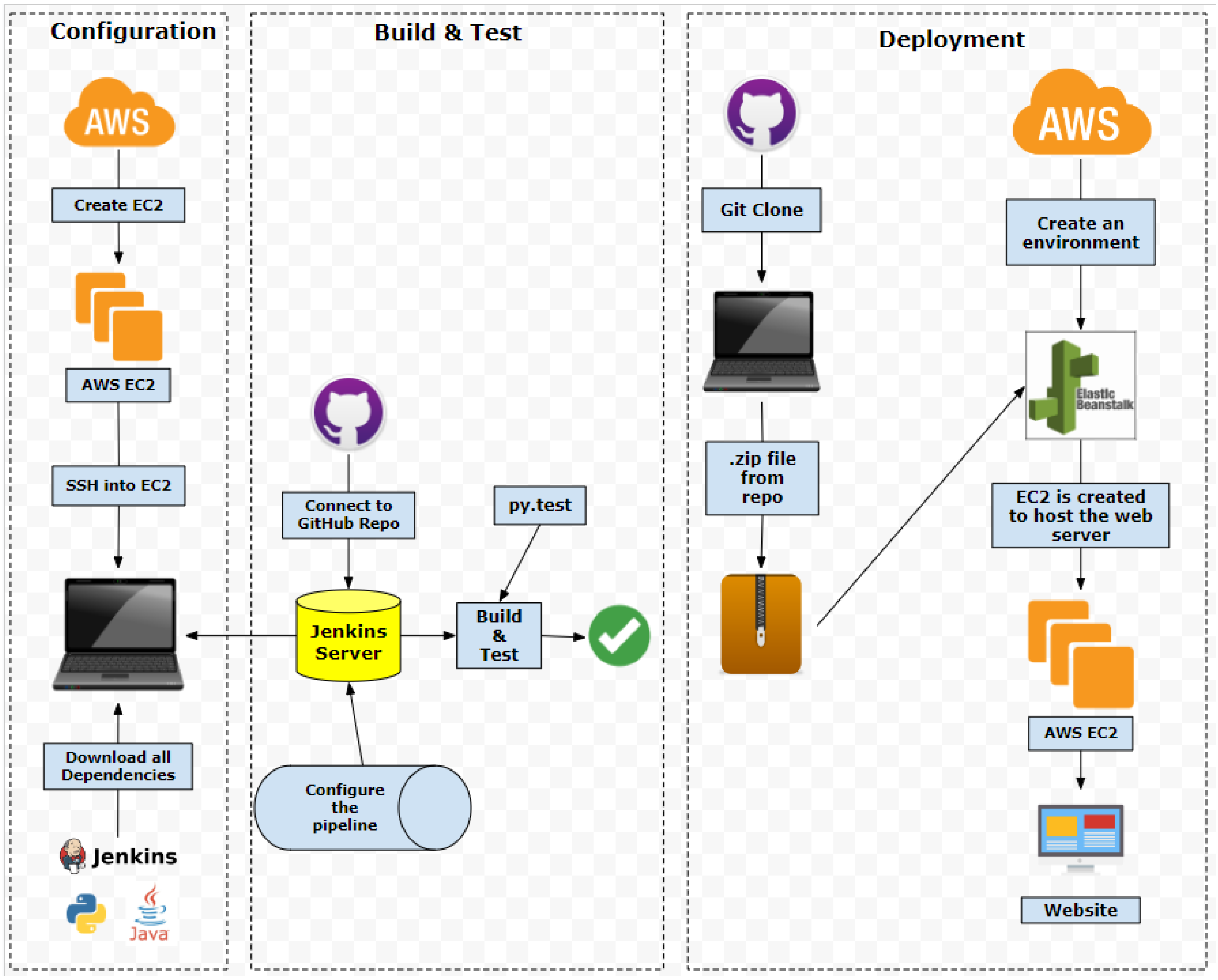
```
$git clone https://github.com/SubornaN/kuralabs_deployment_1/blob/main/application.py
$git archive -v -o myapp.zip --format=zip HEAD
```

Now, we head over to AWS > Elastic Beanstalk > Create a new environment > Web server environment
We will use these configurations listed below.
Application name: url-shortner | **Platform:** Python | **Platform Branch:** Python 3.8
We can now upload the zip file with Upload your code > Local file > Choose file > Create environment


Urlshortner-env
Urlshortner-env.eba-i5gma2mu.us-east-1.elasticbeanstalk.com
Application name: url-shortner

It will take a few minutes to create and launch the environments. We should see something similar to the screenshot (green check), which means the deployment was successful. We can click the link to visit the website.
When we are creating the zip file, we need to make sure it includes all the hidden files. Otherwise, we can run into an error. This is why we are cloning the repository in the terminal before creating the zip file.

Deployment Pipeline Diagram



What could be improved?

Since we connected the Github repository to Jenkins server, the pipeline will start running the build and test automatically every time there is a new commit on the main branch. However, we are using a different platform for the deployment which is not integrated with Jenkins. As a result, we will have to manually check and deploy the new changes. We can improve the deployment portion of the pipeline by integrating Beanstalk with Jenkins so that the deployment happens after the build and test automatically. Also, we need to implement a monitoring system with tools like Splunk and set alerts so that we can get notified if something goes wrong.