Kevin Sanaycela

Deployment 2

**Step 1: Install Jenkins on the Amazon EC2 Server**

To install Jenkins, I ran the commands listed below. I stored this in a file I called install\_jenkins.sh, making sure appropriate permissions were granted using chmod. This is done as the ubuntu user, not the Jenkins user (although the final line creates a Jenkins user in its own bash shell).

**install\_jenkins.sh**

#!/bin/bash

sudo apt update && sudo apt install default-jre

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo gpg --dearmor -o /usr/share/keyrings>

sudo sh -c 'echo deb [signed-by=/usr/share/keyrings/jenkins.gpg] [http://pkg.jenkins.io/debian-stable binary/](http://pkg.jenkins.io/debian-stable%20binary/) > /etc/apt/sources.list.d/jenkins.list'

sudo apt update && sudo apt install jenkins -y

sudo systemctl start jenkins

sudo systemctl status jenkins

sudo passwd jenkins

sudo su - jenkins -s /bin/bash

**Step 2: Creating a Jenkins user in the AWS Account**

To create the Jenkins user, use the following steps:

IAM > Users > Add User > Set username as eb-user > Access Key: programmatic access > Attach exiting policies directly, add “AdministratorAccess” > skip Tags. Should see the following image at this point.

After, you will get sent to a page with the Access Key and Secret Access Key. Download the .csv file and store in safe place. Below are examples of my user and file.

**Graphical user interface, text, application, email

Description automatically generated**

**new\_user\_credentials.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User name** | **Password** | **Access key ID** | **Secret access key** | **Console login link** |
| **eb-user** |  | AKIAYDMJPUTTHQJ7NHVB | 2xFrf3AvciN6gQjewiTn5YQJObbkyan2i4NU/YOn | <https://557023470822.signin.aws.amazon.com/console> |

**Step 3: Install Amazon CLI on Jenkins EC2 / Configuration**

To download the command line interface, I followed the steps in the documentation, by creating a script and running it.

**install\_awscli.sh**

#!/bin/bash

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

sudo apt install unzip

unzip awscliv2.zip

sudo apt install python3-pip -y

sudo ./aws/install

aws --version

aws configure

Following the configure step, set the access key ID, secret key, region (us-east-1), and data output format (JSON).

**Step 4: Creating Jenkins multibranch build**

I then created a multibranch build as we did in Deployment 1, ensuring that a new access token was created since my old one was going to expire soon. Below, I get the following screen:

Graphical user interface, application

Description automatically generated

Graphical user interface, table

Description automatically generated

Adding webhook: Although not necessary, I added a webhook so that any commits to Github would automatically trigger a new build in Jenkins.

Graphical user interface, text, application, email

Description automatically generated

**Step 5: Deploy Application for Elastic Beanstalk CLI**

Run the following commands:

sudo su - jenkins -s /bin/bash

cd /var/workspace/url-shortener\_main/

eb init

eb create

I then noticed a message in my console telling me that the eb package was downloaded in a different place than the $PATH at the time. To fix this, I noticed that the eb package was located in the /.local/bin folder (again, within the Jenkins user, NOT the ubuntu user). We can change the path so that this we are in this directory by running:

export PATH="/var/lib/jenkins/.local/bin:$PATH"

The above command’s syntax is: export PATH=”<path\_to\_folder\_with\_pkg>:$PATH”

The last two steps will require certain configurations. I followed the configuration according to the deployment instructions provided by Tyrone.

**Step 6: Adding Deployment Stage to Jenkinsfile & slacksend( )**

pipeline {

agent any

stages {

stage ('Build') {

steps {

sh '''#!/bin/bash

python3 -m venv test3

source test3/bin/activate

pip install pip --upgrade

pip install -r requirements.txt

export FLASK\_APP=application

flask run &

'''

}

}

stage ('test') {

steps {

sh '''#!/bin/bash

source test3/bin/activate

python3 -m py.test --verbose --junit-xml test-reports/results.xml

'''

}

post {

always {

junit 'test-reports/results.xml'

}

}

}

stage ('deploy') {

steps {

sh '/var/lib/jenkins/.local/bin/eb deploy url-shortner-dev'

}

post{

success {

slackSend(message:

"""

DEPLOYMENT SUCCESSFUL${custom\_msg()}

""")

}

failure {

slackSend(message: """

DEPLOYMENT FAILED ${custom\_msg()}

""")

}

}

}

}

}

def custom\_msg()

{

def JENKINS\_LOG=

"""

Job: [${env.JOB\_NAME}]

Path to log of each step: ${env.BUILD\_URL}consoleText

"""

return JENKINS\_LOG

}

As shown above, I made edits to the deployment stage of the CI/CD pipeline. Firstly, I made sure to make a note of the name of my environment, which in this case, was “url-shortner-dev.”

Next, I wanted to add an additional feature, namely one that notifies the admin user via Slack whenever the deployment stage is complete. Specifically, it will notify the admin user whether the deployment stage of the pipeline was a success or not. Either way, the Jenkins bot will also provide the admin with a log containing a summary of the success of each step in the pipeline.

**Sub-steps:**

1. Slack App Directory > search for Jenkins CI > install to the channel you’d like to receive notifications > edit Configuration > make note of the Token and the workspace name

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. Manage Jenkins > Manage Credentials > Add New Credentials

In the Secret tab, add the Token. You can give this token a name by changing the ID tab.

Graphical user interface, application, email

Description automatically generated

1. Manage Jenkins > Configure System

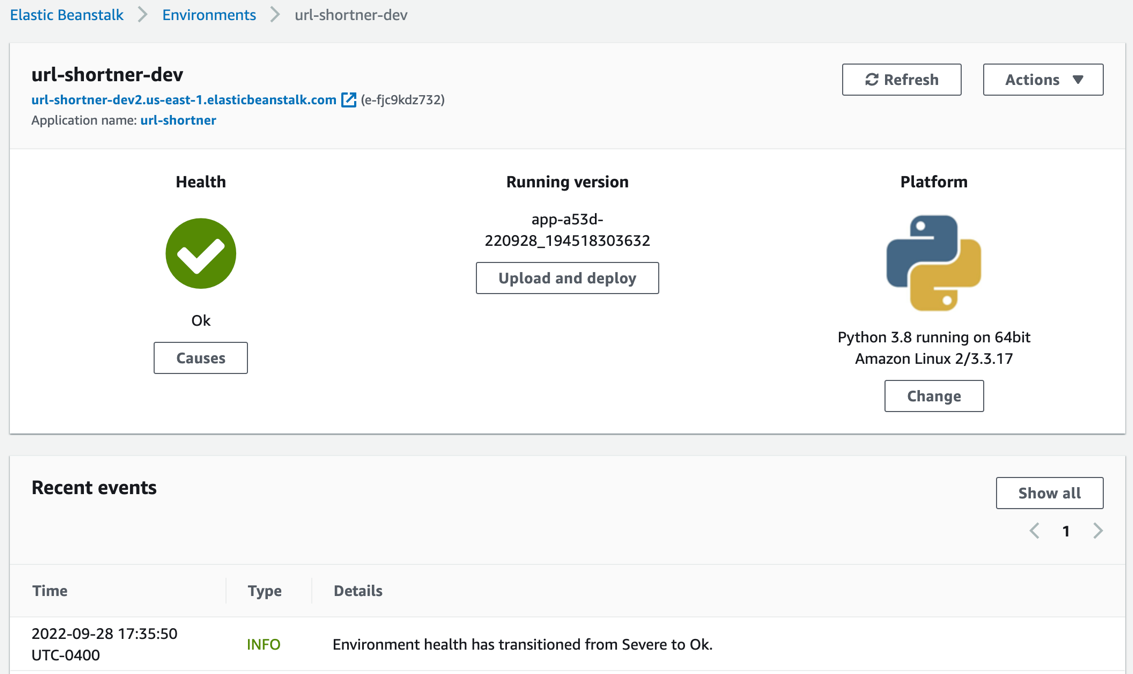
Enter correct values for Workspace and the appropriate credentials.

Graphical user interface, application

Description automatically generated

Make sure that this works by clicking on the “Test Connection” button. You can test the notification system by minutely editing the Jenkinsfile, especially now that we already have a webhook linked to the forked repository.

**Final check on Amazon Elastic Beanstalk:**



Graphical user interface, application

Description automatically generated