Chanesh Mahadeo
Kura Labs
9/26/2022

**Deployment 2**

# Intro:

Welcome to deployment 2 where we go further than the last deployment and actually automate another part of our deployment. In our last deployment we set up our jenkins environment then manually zipped our application and uploaded it to elastic beanstalk. In this deployment we will be automating this process and actually have Jenkins handle our deployment step after a few configurations.

# Setup:

For this section we will be configuring a jenkins server and installing dependencies on our AWS EC2 before configuring our AWS settings. To further consolidate this part and speed up our setup process I have created a script designed to automatically do this.

**NOTE THAT SECURITY GROUP FOR EC2 MUST HAVE 22 80 AND 8080 OPEN. OPTIONALLY ALSO 5000 TO ALLOW HOSTING OF THE FLASK SERVER**

```
#!/bin/bash

# This script is meant to be used as userdata for launching a jenkins ec2
for deployment 2

if [ $UID != 0 ]; then
   echo "Run again with admin permissions"
   exit 1
fi
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | gpg
--batch --yes --dearmor -o /usr/share/keyrings/jenkins.gpg
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' && echo "Jenkins Repo Added"
apt-get update
apt-get install default-jre -y && apt-get install jenkins -y && apt-get
install python3-pip -y && apt-get install python3.10-venv -y
apt-get install unzip -y
systemctl start jenkins && sleep 5s
```

```
usermod -aG sudo jenkins
sudo su jenkins
cd ~
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
"awscliv2.zip"
unzip awscliv2.zip && sudo ./aws/install
pip install awsebcli --upgrade --user
echo "export PATH=~/.local/bin/:$PATH" > .bashrc
source ./.bashrc



exit 0
```

This code is responsible for:

- Creating and starting the Jenkins server
- Installing aws and aws cli
- Configuring the path to allow jenkins user to use “eb”

**Configuring Jenkins**

To configure jenkins is a straightforward and simple process. And will allow us to set up our pipeline.

Steps:

- Fork the deployment repo from https://github.com/kura-labs-org/kuralabs_deployment_2
- Go to your jenkins url with {Public IP of your ec2}:8080
- Ssh into your ec2 instance or use connect on aws site
- Run sudo cat with the password location given
- Input the password into the bar to unlock jenkins
- Install recommended plugins
- Create your user
- Select New item and create a multibranch pipeline
- For source select github and add your forked repo link
- For this example specifically I did not make my repo private so I do not need to set up a github access key.
- Once you validate and it says OK, complete the creation of the repo.
- You should see a build happening

**IAM User Configuration**

Next we will follow steps in order to configure an AWS IAM user. This is important because our jenkins agent will use these credentials and permissions in order to automate our deployment process.

Steps:

- Navigate to IAM in the AWS console. Next click on the Users option in the Access management
- Now Select add user. Next, the username can be EB-user then select Programmatic access and then “next”
- Select “Attach existing policies directly” and select administrator access. Now select Next for this page and the next page
- Finally, create the user and then copy and save the “access key ID” and the “secret access key”

These steps should allow you to successfully create and IAM User and you should now have an access key and a secret access key.

**AWS Configuration**

Now we must configure our access keys and allow aws to successfully use our IAM User that we just created.

Steps:

- SSH into your jenkins ec2 instance
- Run `passwd jenkins`
- Input your desired password for the jenkins user
- Now run `su jenkins`
- This should prompt you to log into the jenkins user
- Now we configure aws using **NOTE THESE ARE TWO SEPERATE COMMANDS**

```
cd
aws configure
```

- This command will prompt you to input both your access key and secret key. This will allow jenkins to act as the IAM user we created earlier
- For region input: us-east-1
- For output format: json

**Elastic Beanstalk Configuration**

We must now configure our elastic beanstalk cli to properly be able to automate our Deployment process.

Steps:

- **NOTE THAT THESE STEPS MUST BE PERFORMED AS THE JENKINS USER**

```
cd ~
cd workspace
cd {name of your deployment pipeline}
```

- `eb init`
- Select us-east-1
- Press Enter
- Select Python
- Select that latest version
- Select N for code commit

These steps will have created the application for our application but not the environment. We will now create the environment with the next command.

Steps:

- Run `eb create`
- Take the default for the next 3 questions by hitting enter (remember the environment name)
- Spot Fleet: No

Your environment should now begin creation. Now we move onto our final steps which is configuring the deploy step of our jenkinsfile

**Jenkinsfile Configuration**

We must now configure the deploy step of our jenkinsfile. After our test step we will add this section to the stages function.

```
stage ('Deploy')
{
                    steps {
                      sh '''

                       /var/lib/jenkins/.local/bin/eb deploy appname-dev

                      '''
                           }
                         }
```

This section will allow Jenkins to now automatically deploy any new builds that we create directly to the staging phase if it passes testing. **NOTE YOU MUST CHANGE APPNAME-DEV WITH YOUR ENVIRONMENT NAME YOU RECEIVED WHEN RUNNING THE EB CREATE COMMAND.** Building your repository should now automatically deploy the new version to EBS

Congrats your deployment is now complete.

## Issues:

Obviously multiple issues are bound to arise in a process such as this. My first issue in following the original documentation was that it lacked specific guidelines to various issues. I fixed these within my own steps but examples of this would include installing the dependencies. Unzip is not a dependency that comes with an aws ec2 so you must apt install this. Also the jenkins user must first be added to the sudoers group. The jenkins user must also have its bin added to the path in order to run the eb command. This is all solved within my userdata script when creating the ec2. My script will do all of this and have the environment set up properly before you even load into the ec2. The workspace folder was also not in the location that it was assumed to be in the original documentation.

## Changes:

Some changes I made to the original documentation was the order of the steps in which things were executed. First, my userdata script covers a lot of background setup steps which just take up time and dont need to be manually done by the engineer. Next I removed the github access token step because it is not needed as this is a public repository. I also removed the need to include the shell everytime you su into the jenkins user because jenkins automatically assigns the bash shell to the jenkins user. I confirmed this by running cat /etc/passwd upon initial startup of my ec2 to verify that it had a shell assigned to it. I corrected the location of the workspace folder. Another change I made to this deployment was I added a test to check if the json storing the “go” variable was working correctly. In the future I will be adding a Cypress test, adding more to the html of the site, and adding more phases to the jenkins including a linter.