Ishtaar Desravines
10/28/2022
Deployment 4 Documentation

Objective: Deploying a python application to Gunicorn through the CI/CD pipeline using Jenkins, Github, AWS services, and Terraform. The AWS default VPC was used as a Jenkins server and Terraform was used to create another infrastructure that the application was deployed to.

Recap of the Deployment:

- A source code for a URL shortener flask application is added to Github.
- A Jenkins Server, created with an EC2 with ports 22, and 8080 open. Port 8080 is open to access the Jenkins GUI.
- Terraform is installed on the Jenkins server in order to run terraform stages in the Jenkins pipeline.
- Once Jenkins is configured, with AWS and Github credentials, an SCM checkout from Github occurs and the Build, Test, Init, Plan, Apply, stages are run on the source code on the Jenkins server.
- The Terraform apply stage creates the infrastructure the url shortener application will be deployed to.
- User data is included in the Terraform configuration files in order to spin up two instances that install git and gunicorn when it is launched. The EC2s have ports 22 and 8000 open.
- The application files are downloaded from another Github repo and then deployed using gunicorn. The url application website can be accessed with the IP of the EC2 instance and specifying port 8000 (ex. 10.0.0.17:8000)

Creating a Jenkins Server on an EC2

- Create a Jenkins manager on an EC2 on AWS
- Launch an instance and open ports 22 (ssh), 80 (HTTP), and 8080 (TCP) in the security groups setting.
- Under “Advanced details”, add jenkins server script below to the “User data” section to automatically install Jenkins on the EC2 and create an active agent.
- Launch instance and ssh into the Jenkins EC2.
- Verify jenkins is running on the EC2 with: `$ systemctl status jenkins`
- Open a web browser and enter http://{EC2publicipaddress}:8080 to access the jenkins webpage.

- Retrieve the access key and create a Jenkins account.

```
#!/bin/bash

sudo apt update
sudo apt -y install openjdk-11-jre
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee
/usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc]
https://pkg.jenkins.io/debian-stable binary/ | sudo tee
/etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get -y install jenkins
sudo systemctl start jenkins
systemctl status jenkins >> ~/file.txt
```

Installing Virtual Environment on Jenkins Server:

- Python must be installed since the application includes python files.
- Install virtual environments on Jenkins EC2 so that the application can run without version conflicts from their dependencies.
- Follow the commands below:

```
$ sudo apt install python3-pip
$ sudo apt install python3.10-venv
```

Install Terraform on Jenkins Server:

- Terraform needs to be installed on the Jenkins Server in order to run the Terraform stages in the Jenkins file (Init, Plan, Apply).

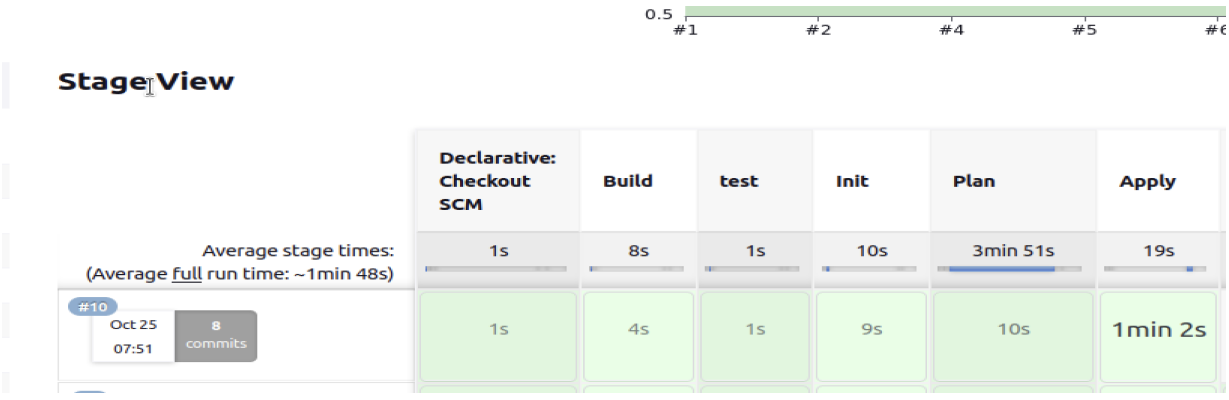
```
$ wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | sudo
tee /usr/share/keyrings/hashicorp-archive-keyring.gpg
$ echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
$ sudo apt update && sudo apt install terraform
```

Configure Credentials on Jenkins:

- Navigate to Jenkins Dashboard and select “Manage Credentials”
- Then select “Add Credentials”
- For AWS Access Key Credentials:
    - Kind: Secret text
    - Scope: Global
    - Secret: paste AWS access key
    - ID: AWS_ACCESS_KEY
    - Select Create
- Repeat the steps above for AWS Secret Access Key Credentials

Create a Pipeline Build in Jenkins:

- Create a new pipeline by selecting: “New Item” and enter the name of the application: “url-shortener” → select “Multibranch pipeline” → select “OK”
- Enter Display Name: Build Flask
- Enter Description: CI/CD pipeline deployment 4
- Add Branch source: Github and under Github Credentials: select “Add” → Jenkins
- Under “Username”: Enter Github Username
- Under “Password”: Enter Github personal access token
- Save Github credentials
- Enter the Github repository URL → select “Validate”
- Check to make sure under Build Configuration → Mode , it says “by Jenkinsfile” and under Script it say “Jenkinsfile”
- Select Apply → Save
- Saving this configuration should trigger an automatic SCM checkout, build, then test in Jenkins. All of these stages of the pipeline should pass.



Add Destroy Stage to Delete Infrastructure Created by Terraform

- To the Jenkins file, add the following code to make it the last stage of the pipeline and then start another build on Jenkins.

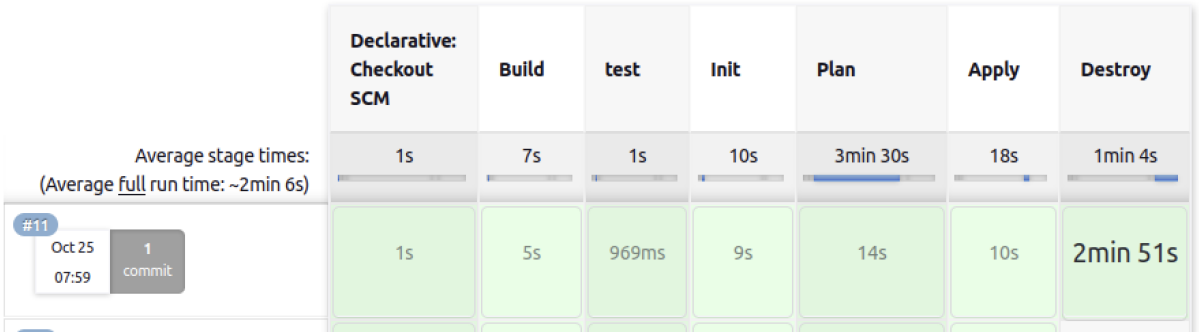
```
stage('Destroy') {
       steps {

       withCredentials([string(credentialsId: 'AWS_ACCESS_KEY', variable:
'aws_access_key'),
                        string(credentialsId: 'AWS_SECRET_KEY', variable:
'aws_secret_key')]) {

                           dir('intTerraform') {
                              sh 'terraform destroy -auto-approve
-var="aws_access_key=$aws_access_key"
-var="aws_secret_key=$aws_secret_key"'
  }
 }
 }
}
```

- All stages should pass and the infrastructure on terraform will be destroyed

Stage View



Improvements:

- Add a webhook to trigger automatic build any time there is a change to the source code.
- Add an application load balancer to the infrastructure created by terraform to distribute traffic to the EC2s in either availability zones.