Ishtaar Desravines
Deployment 5 Documentation

Objective:

Deploying a python application to Gunicorn through the CI/CD pipeline using Jenkins, Github, AWS services (VPC, IAM, ECS), and Terraform. A Jenkins manager and two Jenkins agents were created on 3 ECS2 in the default VPC of AWS. The Jenkins agent with Docker installed was responsible for creating an image of the application and pushing it to Dockerhub and the Jenkins agent with Terraform installed was responsible for creating the AWS infrastructure that the application was deployed onto.

Recap:

- 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.
- A Jenkins agent, created with an EC2 with port 22 open and Terraform installed to create VPC, ALB, and ECS resources.
- A Jenkins agent, created with an EC2 with port 22 open and Docker installed to create an image of the application and push the image to Docker hub.
- Once Jenkins is configured, with AWS, Docker hub, Github credentials, an SCM checkout from Github occurs and the Build, Test, Docker Build, Docker Push, Terraform ECS Deploy, and Terraform ECS Destroy stages are run on the source code.
- When Terraform creates the resources for the VPC, ALB, and ECS, the image of the application is pulled onto the infrastructure and the container is run.
- The user can access the application through the url that is assigned to the ALB created by Terraform.

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
```

Create a Docker Jenkins Agent:

- Launch an EC2 with port 22 (ssh) open and the following user data script to install Docker and Java Runtime Environment.
- The Docker Jenkins agent will be used to build the image of the url shortener application and push the image to Dockerhub.

```
#!/bin/bash

curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo apt update
sudo apt -y install default-jre
```

Create a Terraform Jenkins Agent:

- Launch an EC2 with port 22 (ssh) open and the following user data script to install Terraform and Java Runtime Environment.
- The Terraform Jenkins agent will be used to create the infrastructure and deploy the application to ECS on AWS.

```
#!/bin/bash

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
sudo apt -y install default-jre
```

Configure and Connect Jenkins Agent to Jenkins Server:

- On Jenkins GUI on Jenkins Manager server, select “Build Executor Status” and then “+ New Node”
- Give the Node a name, ex “dockerAgent” and select permanent agent
- Enter the following configurations:
  - Name: dockerAgent
  - Description: Docker Deployment server
  - Number of executors: 1
  - Remote root directory: /home/ubuntu/agent
  - Labels: dockerAgent
  - Usage: only build jobs with label
  - Launch methods: launch agents via ssh
  - Host: IP address of Docker Jenkins agent EC2
  - Credentials: ubuntu (SSH-CALI)

        - Add → Jenkins → SSH username with private key
        - Scope: Global, Description: deployment agent servers
        - Username: ubuntu, private key: paste EC2 private key
    - Host Key verification strategy: non verifying verification strategy
    - Availability: keep this agent online as much as possible.
- Select Apply then Save and wait for Jenkins manager to launch the jenkins agent.
- Repeat process for the Terraform Jenkins Agent

Install Docker Pipeline Plugin on Jenkins Manager:

- Select “Dashboard” → “Manage Jenkins” → “Manage Plugins”
- Select “Available”, then search “Docker pipeline” in the search bar.
- Install the plugin

Configure Credentials on Jenkins Manager:

AWS:

- 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

Docker Hub:

- First generate a personal access token in Docker Hub account.
- Select username → “Account Settings” → “Security”
- Under “Access Token”, select “New Access Token”
- Enter a description and permissions to the repository.
- Next, navigate to Jenkins Dashboard and select “Manage Credentials”
- Then select “System” → “Global Credentials” → “ + Add Credentials”
    - Kind: Username with password
    - Scope: Global

    - Username: docker hub username
    - Password: paste access token
    - Select Create

Dockerfile On Github Repository:

- Include a Dockerfile on your github repository for the Jenkinsfile to read when building the image it will push to Docker hub.
- Dockerfile for the url shortener application:

```
FROM python:latest

RUN apt update

RUN apt -y install git

RUN git clone https://github.com/IshtaarD/kuralabs_deployment_5

WORKDIR /kuralabs_deployment_5

RUN pip install -r requirements.txt

RUN pip install gunicorn

EXPOSE 8000

ENTRYPOINT python3 -m gunicorn -w 4 application:app -b 0.0.0.0:8000
```

Creating a Jenkinsfile:

- Create a Jenkinsfile to specify stages of the pipeline.
- A Build, Test, Docker Build, Docker Push, Terraform ECS Deploy, and Terraform ECS Destroy were included in the Jenkinsfile below:

```
pipeline {
  agent any
  environment {
    DOCKERHUB_CREDENTIALS = credentials('dockerhub')
  }
   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
        py.test --verbose --junit-xml test-reports/results.xml
        '''
        }

      post{
        always {
          junit 'test-reports/results.xml'
        }

      }
    }
    stage ('Docker Build'){
        agent { label 'dockerAgent' }
        steps {
            sh 'sudo docker build -t ishtaard/gunicorn-flask:latest .'
        }
    }
    stage ('Docker Push'){
        agent { label 'dockerAgent' }
        steps {
            sh '''#!/bin/bash
            echo $DOCKERHUB_CREDENTIALS_PSW | sudo docker login -u
$DOCKERHUB_CREDENTIALS_USR --password-stdin
            sudo docker push ishtaard/gunicorn-flask:latest
            docker logout
            '''

        }
    }
    stage ('Terraform ECS Deploy'){
      agent { label 'terraformAgent' }
      steps {
        withCredentials([string(credentialsId: 'AWS_ACCESS_KEY', variable:
'aws_access_key'),
        string(credentialsId: 'AWS_SECRET_KEY', variable: 'aws_secret_key')]) {
            dir('intTerraform') {
              sh ''' #!/bin/bash
              terraform init
              terraform plan -out plan.tfplan -var="aws_access_key=$aws_access_key"
-var="aws_secret_key=$aws_secret_key"
              terraform apply plan.tfplan
              sleep 180
              '''
            }
        }
      }
    }
    stage ('Terraform ECS Destroy') {
      agent { label 'terraformAgent' }
      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"'
            }
        }
      }
   }
 }
}
```

IAM Role for ECS:

- Access IAM service on AWS and select “Roles” under “Access Management”
- Select “ Create Role” then “AWS service” under “Trusted entity type”

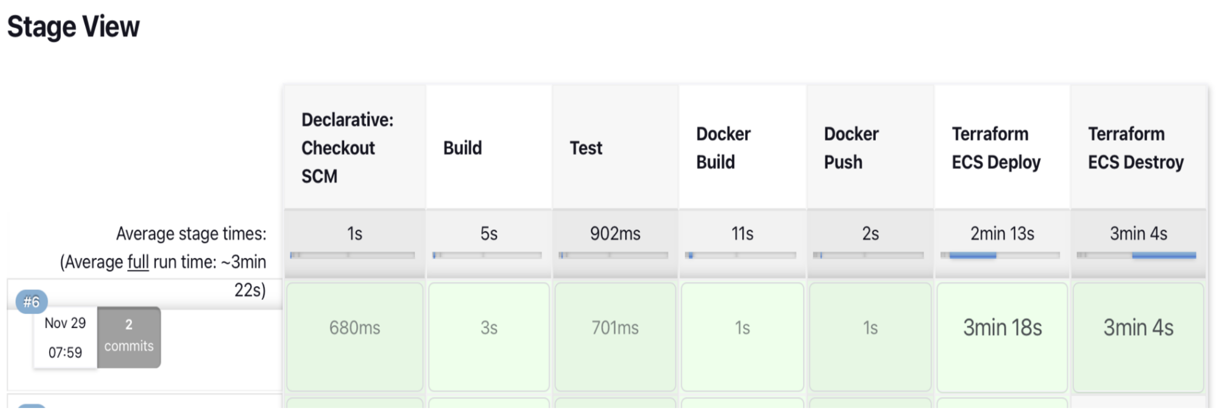
- Under “Use Case”, choose “Elastic Container Service” in the drop down menu and then”Elastic Container Service Task” and “Next”
- In the search bar type, “AmazonECSTaskExecutionRolePolicy”, check the box and press “Next”.
- Name the role “ECSTaskExec” and confirm.
- Copy the ARN to use the terraform configuration files in next step.

Modifications to Terraform Configuration Files (main.tf):

- The provided Terraform configuration files will provision the AWS resources to deploy the application to ECS.
- In the main.tf file, replace with the name of your image on line 33.
- In the main.tf file, on lines 55 and 56, replace the ARN with the ARN you copied when creating a role in IAM.

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 5
- 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 then will go through all of the stages listed in the Jenkinsfile. All of these stages of the pipeline should pass.



Stage View

	Declarative: Checkout SCM	Build	Test	Docker Build	Docker Push	Terraform ECS Deploy	Terraform ECS Destroy
Average stage times: (Average full run time: ~3min 22s)	1s	5s	902ms	11s	2s	2min 13s	3min 4s
#6 Nov 29 07:59 2 commits	680ms	3s	701ms	1s	1s	3min 18s	3min 4s

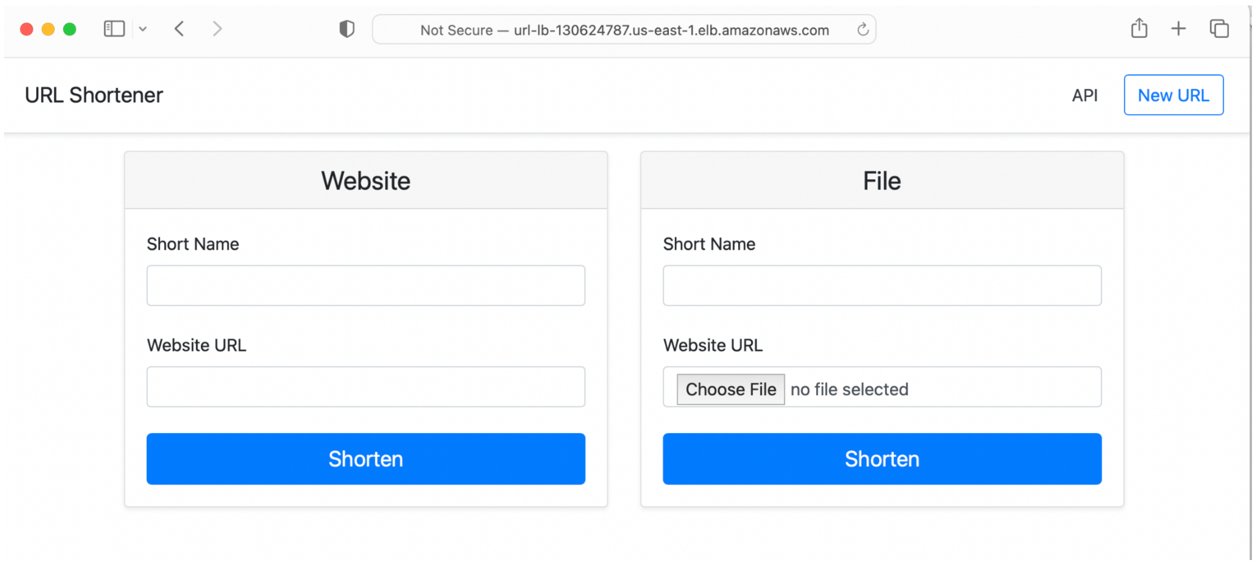
Access the Application:

- The url shortener application can be accessed through the dns name that is assigned through AWS when terraform creates the resources.
- The url is output in the Jenkins logs.

```
Outputs:

⊠[0malb_url = "http://url-lb-130624787.us-east-1.elb.amazonaws.com"
+ sleep 180
```

- The url shortener application:



Issues:

- The terraform configuration files were configured to work with Flask so port 5000 was exposed. Since I containerized the url shortener application and ran it with gunicorn, I needed to replace all the places specifying port 5000 with port 8000 (lines 44 and 82 in main.tf, line 3 in ALB.tf, and lines 124 and 125 in vpc.tf).