EPAM UNIVERSITY PROGRAM

DevOps Educational Program 2020 Q4-2021 Q1

FINAL COURSE WORK

Build and deploy of simple Spring Boot application by CI/CD pipeline.

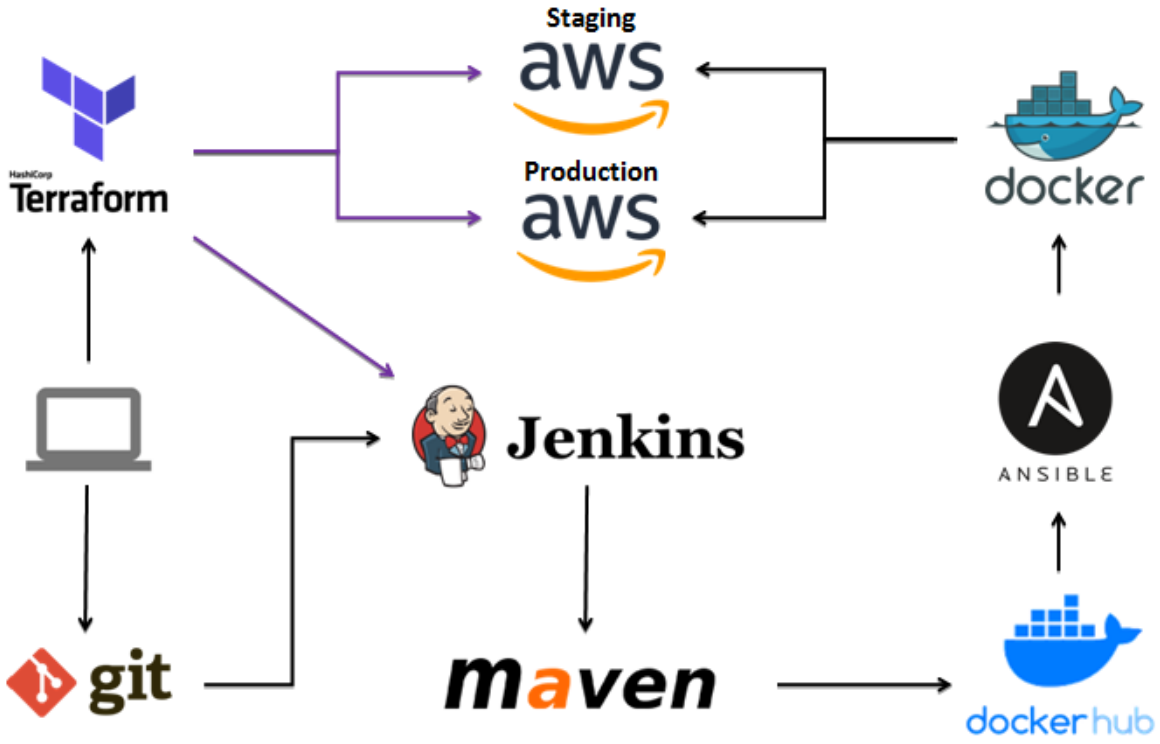
Made by Valerii Mankovskyi

Scenario of workflow

• Jenkins pulls latest changes and build artifact after each commit to master branch;
• Jenkins builds Docker image;
• Jenkins uploads Docker Image to repository (Docker Hub);
• Jenkins triggers deployment of Staging/Production environment after each success build (Continuous Integration / Continuous Deployment).

Architecture

The following architecture for CI/CD pipeline.



Prerequisites

Local machine
AWS account
Docker Hub account
GitHub account

Fork the code sample repository at https://github.com/spring-projects/spring-boot/tree/2.1.x/spring-boot-samples/spring-boot-sample-web-ui

Create a Jenkins CI server using Terraform

Provisioning a Jenkins CI (Continuous Integration) using Infrastructure as Code (IaC).
Log in to the AWS Management Console and create IAM role with policy for Terraform and get credentials for programmatic acces (access key ID and secret access key).

Structure of Provision directory on my local machine.

```
linux@linux:~/finalproject/Spring-boot$ tree Provision/
Provision/
├── builder_data.sh
├── jenkins_data.sh
├── main.tf
└── variables.tf
```

Terraform will provision two AWS EC2 instance (Jenkins Server and Build Server) and install Apache Maven, Docker, Java 11, Jenkins as shown in the **jenkins_data.sh** and **builder_data.sh**.

jenkins_data.sh

```
#!/bin/bash -v
apt-get update -y
apt-get install -y ansible default-jre maven awscli > /tmp/alllogs.log

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
apt-get update && sudo apt-get install terraform > /tmp/terraform.log

wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -

sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > \
    /etc/apt/sources.list.d/jenkins.list'

apt-get update
apt-get install jenkins -y > /tmp/jenkins.log
systemctl stop jenkins.service >> /tmp/jenkins.log
systemctl start jenkins.service >> /tmp/jenkins.log

apt install docker.io -y > /tmp/docker.log
sudo usermod -aG docker ubuntu >> /tmp/docker.log
newgrp docker
sudo systemctl start docker
sudo systemctl enable --now docker
sudo chown ubuntu:docker /var/run/docker.sock
```

builder_data.sh

```
#!/bin/bash -v
apt-get update -y
apt-get install -y ansible default-jre maven awscli > /tmp/logs.log

curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
apt-get update && sudo apt-get install terraform > /tmp/terraform.log

sudo apt-get install docker.io -y > /tmp/docker.log
sudo usermod -aG docker ubuntu >> /tmp/docker.log
newgrp docker
sudo systemctl start docker
sudo systemctl enable --now docker
sudo chown ubuntu:docker /var/run/docker.sock

sudo apt-get install -y ansible > /tmp/ansible.log
```

“Terraform apply” will also output the IP address of the Jenkins and Build servers.

main.tf

```
provider "aws" {
  region          = var.region
}

resource "aws_instance" "jenkins" {
  ami             = var.ami_id
  instance_type   = var.instance_type
  key_name        = var.key_name
  security_groups = [var.aws_security_group]
  user_data       = file("jenkins_data.sh")
  tags = {
    Name          = "Jenkins"
  }
}

resource "aws_eip" "jenkins" {
  instance        = aws_instance.jenkins.id
  vpc             = true
}

resource "aws_instance" "builder" {
  ami             = var.ami_id
  instance_type   = var.instance_type
  key_name        = var.key_name
  security_groups = [var.aws_security_group]
  user_data       = file("builder_data.sh")
  tags = {
    Name          = "Builder"
  }
}
```

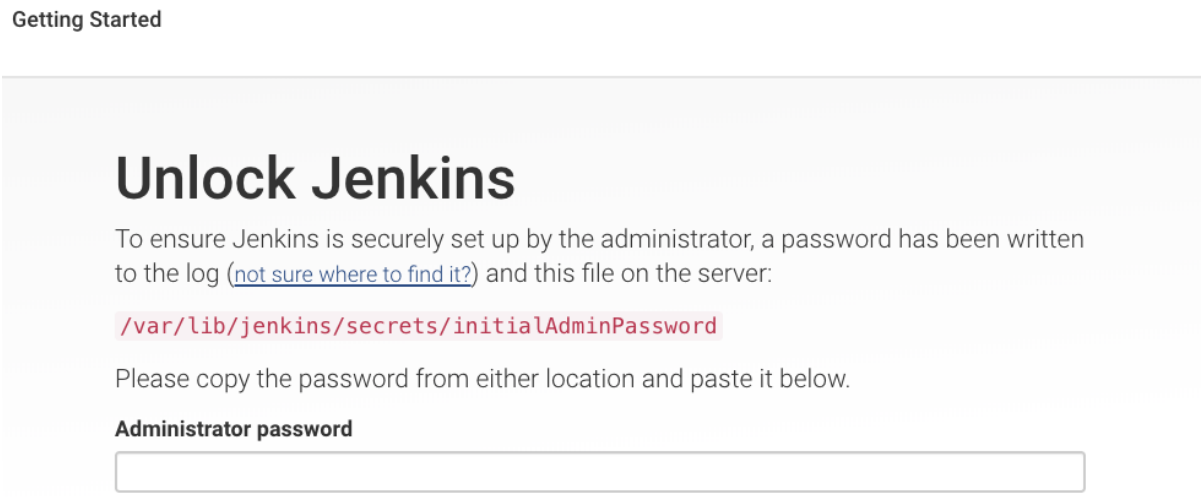
```
resource "aws_eip" "builder" {
  instance        = aws_instance.builder.id
  vpc             = true
}


output "jenkins_tag" {
  value           = aws_instance.jenkins.tags
}

output "jenkins_elastic_ip"   {
  value           = aws_eip.jenkins.public_ip
}

output "builder_tag" {
  value           = aws_instance.builder.tags
}

output "builder_elastic_ip"   {
  value           = aws_eip.builder.public_ip
}
```

Variables.tf file.

variables.tf

```
variable "ami_id" {
  default="ami-0767046d1677be5a0"
}

variable "region" {
  default="eu-central-1"
}

variable "instance_type" {
  default="t2.medium"
}

variable "key_name" {
  default = "MyProdServer"
}

variable "aws_security_group" {
  default = "launch-wizard-7"
}
```

Using a browser, open the page at **http://jenkins_ip_address:8080**.Tthe Jenkins admin page will be displayed:



Using the MobaXterm app log in to the Jenkins CI server, find the Administrator password by running the following command:

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

Enter this Administrator password on the Jenkins Console by pasting it into the input box, and click Next. Click Install suggested plugin.

Configure Jenkins

1.Plugins:

Log in to the Jenkins console, click Manage Jenkins → Manage Plugins → Available.
Choose and install some more plugins in addition to the default plugins: SSH Build Agents, CloudBees AWS Credentials, Docker Pipeline, AnsiColor, Pipeline, GitHub Groovy Libraries.
Then restart Jenkins by clicking the Restart Jenkins check box.

2.Credentials:

Docker Hub: Username with password. ID: Docker_Hub_Credentials.
GitHub: Username with private key. ID: GitHub.
AWS: AWS Access key and AWS Secret access key. ID: AWS_Credentials.
Jenkins Agent (Build Server): Username wuildServer_SSH_Cred.

3.System configuration:

Set home directory for Maven and Docker.

Configure the Jenkins job and pipeline

From the Jenkins console, click New item. Choose Pipeline, name it **Final_Project** and click OK.
Choose GitHub and from the drop-down select the GitHub credentials. Enter the Repository URL. Mark GitHub hook trigger for GITSCM polling check box.
In Pipeline section choose “Pipeline script form SCM” and type path to Jenkins file in the GitHUb repo.
After that apply and save the Jenkins pipeline.

The Pipeline we are defining has prerequisites and stages

At the beginning of Pipeline we set some environment variables for the next steps and some parameters to choose environment to deploy and artifact version.

Jenkinsfile

```
pipeline {

    environment {
        registryCredential = 'Docker_Hub_Credentials'
        imagename = "jundevops/maven"
        dockerImage = ''
    }
    parameters {
        choice(choices: ['apply','destroy'], description: 'CHOOSE ACTION.' , name: 'ACTION')
        choice(choices: ['staging','production','all'], description: 'CHOOSE SERVERS TO DEPLOY.' , name: 'HOSTS')
        string(name: 'BUILDS', defaultValue : 'latest', description: 'ENTER BUILD NUMBER/TAG TO DEPLOY.')
        }

    agent { label 'ubuntu' }

    options {
        timestamps ()
        ansiColor('xterm')

    }
```

The first stage is to get all files and code from our Github repository.

```
        stage('Git') {
            steps {
                echo '====== Pulling git ======'
                git credentialsId: 'GitHub', url: 'git@github.com:JuniorDevOps/Spring-boot.git'
            }
        }
```

The second one is the infrastructure provisioning stage. It will deploy staging and production environment.

```
stage('Infra Provisioning') {
    steps {
        echo '====== Infrastructure provisioning ======'
        dir ('Terraform') {
        withCredentials([[$class: 'AmazonWebServicesCredentialsBinding', accessKeyVariable: 'AWS_ACCESS_KEY_ID',
        script {
        sh "terraform init -input=false"
        sh "terraform plan -input=false"
        sh "terraform ${params.ACTION} -auto-approve"
                }
            }
        }
    }
}
```

Terraform will provision instances and install Docker and Ansible as shown in the servers_data.sh.
Terraform apply will also output the IP address of the staging/production servers and make inventory file for Ansible.

Main.tf file.

main.tf

```
provider "aws" {
  region          = var.region
}

resource "aws_instance" "stage" {
  ami             = var.ami_id
  instance_type   = var.instance_type
  key_name        = var.key_name
  security_groups = [var.aws_security_group]
  user_data       = file("servers_data.sh")
  count           = var.count_stage
  tags = {
    Name          = "Stage-${count.index + 1}"
  }
}
```

```
resource "aws_instance" "prod" {
  ami             = var.ami_id
  instance_type   = var.instance_type
  key_name        = var.key_name
  security_groups = [var.aws_security_group]
  user_data       = file("servers_data.sh")
  count           = var.count_prod
  tags = {
    Name          = "Prod-${count.index + 1}"
  }
}
```

```
output "stage_tags" {
  value           = aws_instance.stage.*.tags
}
output "stage-ip"   {
  value           = aws_instance.stage.*.public_ip
}
output "prod_tags"  {
  value           = aws_instance.prod.*.tags
}
output "prod-ip"    {
  value           = aws_instance.prod.*.public_ip
}
```

Variables.tf file.

variables.tf

```
variable "ami_id" {
  default = "ami-0767046d1677be5a0"
}
variable "region" {
  default = "eu-central-1"
}
variable "instance_type" {
  default = "t2.micro"
}
variable "count_stage" {
  default = 1
}
variable "count_prod" {
  default = 1
}
variable "key_name" {
  default = "MyProdServer"
}
variable "aws_security_group" {
  default = "launch-wizard-7"
}
```

Output.tf file.

output.tf

```
resource "local_file" "AnsibleInventory" {
 content     = templatefile("inventory.tmpl",
 {
 stage-ip    = aws_instance.stage.*.public_ip
 prod-ip     = aws_instance.prod.*.public_ip
 }
 )
 filename    = "inventory"
}
```

Inventory.tmpl file that generate Ansible inventory file from output.tf

inventory.tmpl

```
[staging]
%{ for index, ip in stage-ip ~}
${ip} ansible_host=${stage-ip[index]}
%{ endfor ~}

[production]
%{ for index, ip in prod-ip ~}
${ip} ansible_host=${prod-ip[index]}
%{ endfor ~}

[servers_all:children]
staging
production
```

Inventory file.

inventory

```
1  [staging]
2  3.120.157.185 ansible_host=3.120.157.185
3
4  [production]
5  35.159.50.74 ansible_host=35.159.50.74
6
7  [servers_all:children]
8  staging
9  production
```

The third one is building and testing our application code and packing app in .jar file.

```
stage('Maven Build/Test') {
    when { expression { params.ACTION == "apply" }
           expression { params.BUILDS == "latest" }
         }
    steps {
        echo '====== Build/Test Application ======'
        dir ('Spring-App/spring-boot-samples/spring-boot-sample-web-ui/') {
        sh "mvn -f pom.xml clean install"
        sh "cd target && mv *.jar myapp.jar && mv myapp.jar ~/jenkins/workspace/${JOB_NAME}/Docker"
        }
    }
}
```

The fourth step is building a docker image with our app.

```
stage('Image Build') {
    when { expression { params.ACTION == "apply" }
           expression { params.BUILDS == "latest" }
         }
    steps {
        echo '====== Building Application Docker Image ======'
        script {
            dir ('Docker') {
            dockerImage = docker.build imagename
            }
        }
    }
}
```

Dockerfile file.

Dockerfile

```
FROM jundevops/alpine-java
EXPOSE 8080
COPY myapp.jar .
ENTRYPOINT ["java","-jar","myapp.jar"]
```

The fifth step is Continuous Delivery stage – deliver our Docker Image to Docker Hub. Also we implement version control containers in case if the latest version is not working.

The easiest solution is to maintain version for each build. This can be achieved by using environment variables $BUILD_NUMBER.

```
stage('Image Delivery') {
    when { expression { params.ACTION == "apply" }
           expression { params.BUILDS == "latest" }
         }
    steps {
        echo '====== Pushing Application Docker Image ======'
        script {
           docker.withRegistry( '', registryCredential ) {
           dockerImage.push("$BUILD_NUMBER")
           dockerImage.push('latest')
           }
        }
    }
}
```

The sixth stage is cleaning up the previously built image on the Build server.

```
stage('Local Images Remove') {
    when { expression { params.ACTION == "apply" }
           expression { params.BUILDS == "latest" }
         }
    steps {
        echo '====== Remove Local Application Docker Image ======'
        sh "docker rmi $imagename:$BUILD_NUMBER"
        sh "docker rmi $imagename:latest"
    }
}
```

The seventh step is deploying our containers to particular environment.

Ansible config file.

ansible.cfg

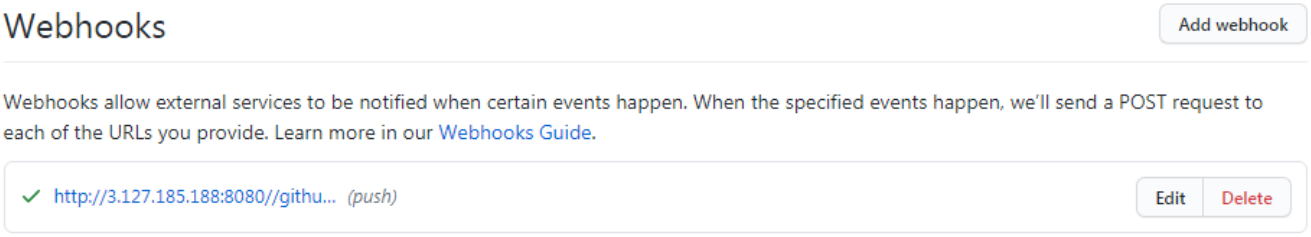
```
[defaults]
host_key_checking = false
inventory         = ~/jenkins/workspace/${JOB_NAME}/Terraform/inventory
```

Ansible-playbook create_container.yml file.

create_container.yml

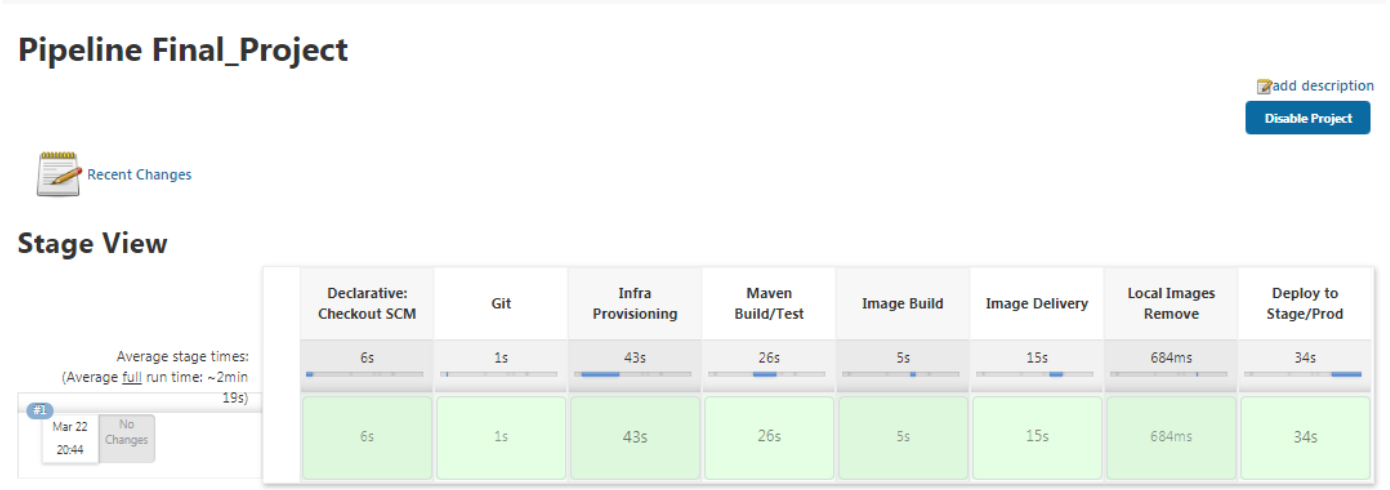
```
---
 - hosts: "{{ HOST }}"
   become: true
   ignore_errors: true
   vars:
      imagename: jundevops/maven
      contname: jundevops
   tasks:
    - name: Stop previous docker container
      shell: docker stop "{{ contname }}"

    - name: Remove stopped container
      shell: docker rm -f "{{ contname }}"

    - name: Remove docker images
      shell: docker image rm -f "{{ imagename }}"

    - name: Create a docker container
      shell: docker run -d --name "{{ contname }}" -p 8888:8080 "{{ imagename }}":"{{ BUILD }}"
```

Don’t forget to define a webhook to run Pipeline when a commit is submitted to your Github repository.

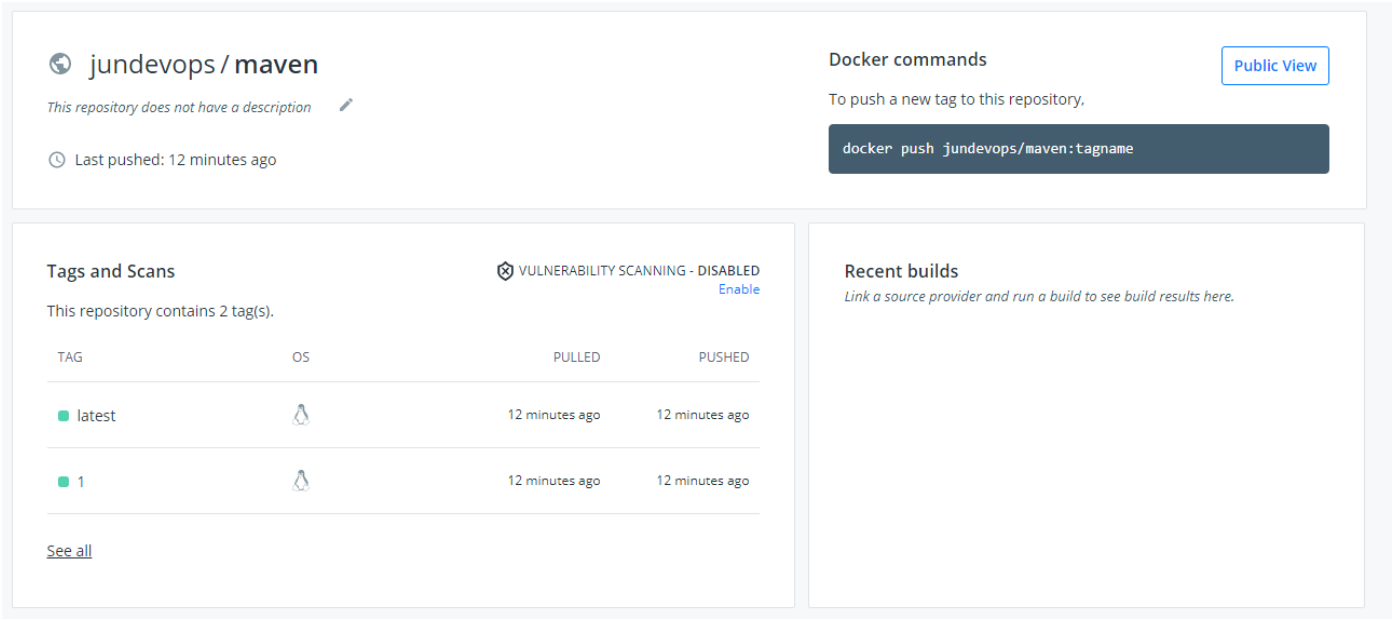


RESULTS OF MY WORK

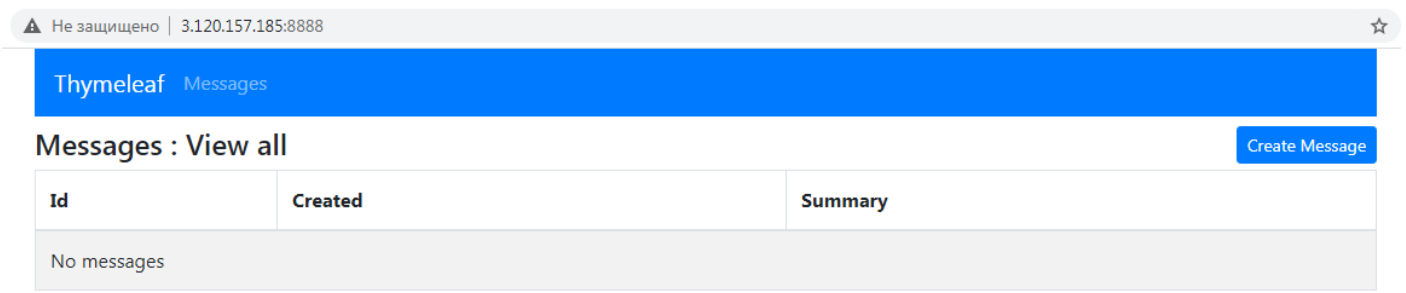
Below is a screenshot of the final run of Pipeline.
If all goes well, you will see how your Pipeline ran without errors.



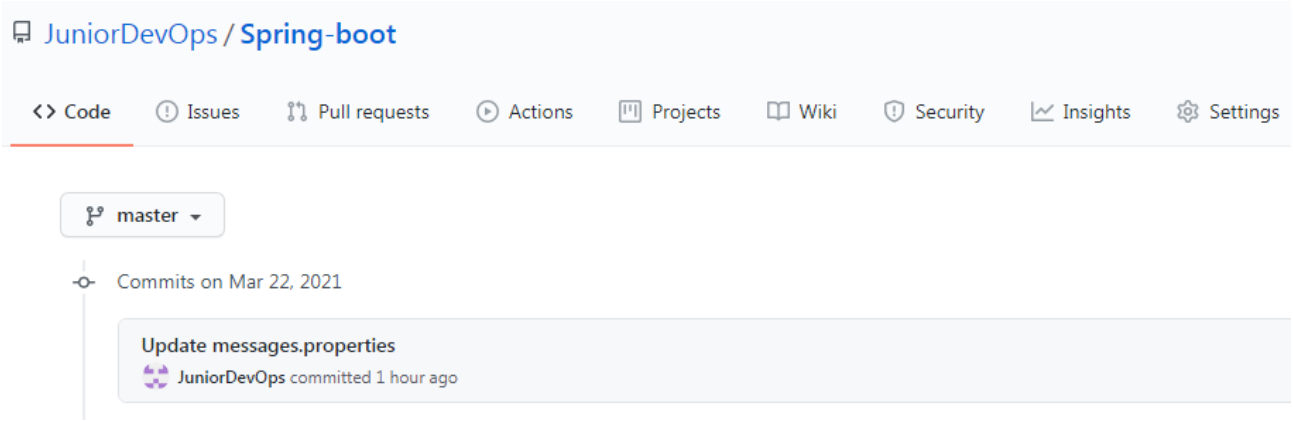
A new Docker image is pushed to your Docker registry.



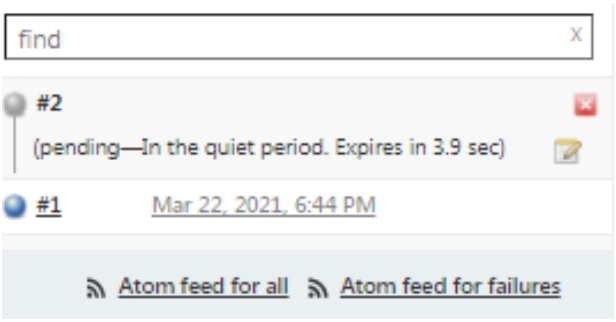
And application page on given by Terraform ip address.

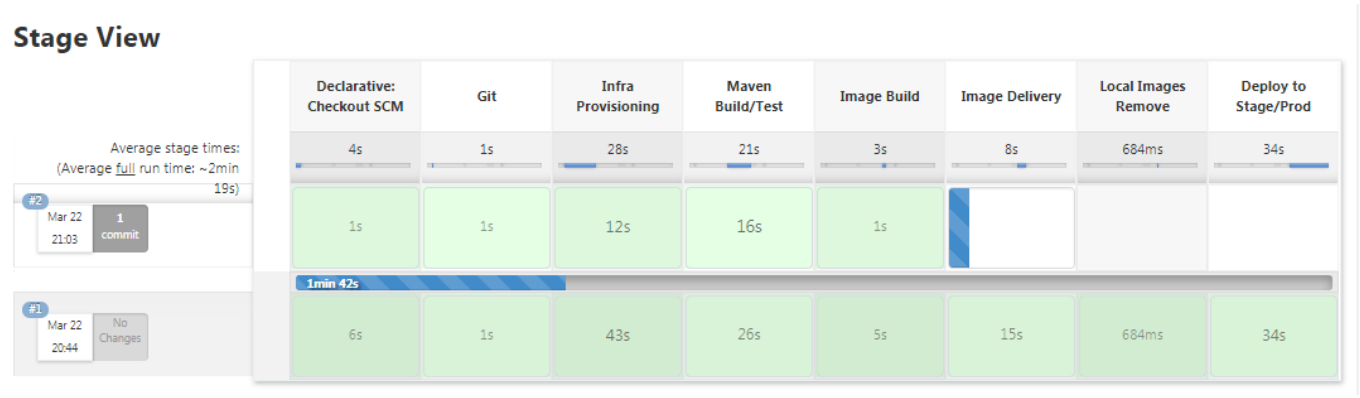


Let`s change something in our application resources. For example, add a string to list.title “DevOps messages from sender “EPAM University”, make commit and push to repo.
Webhook will fire.

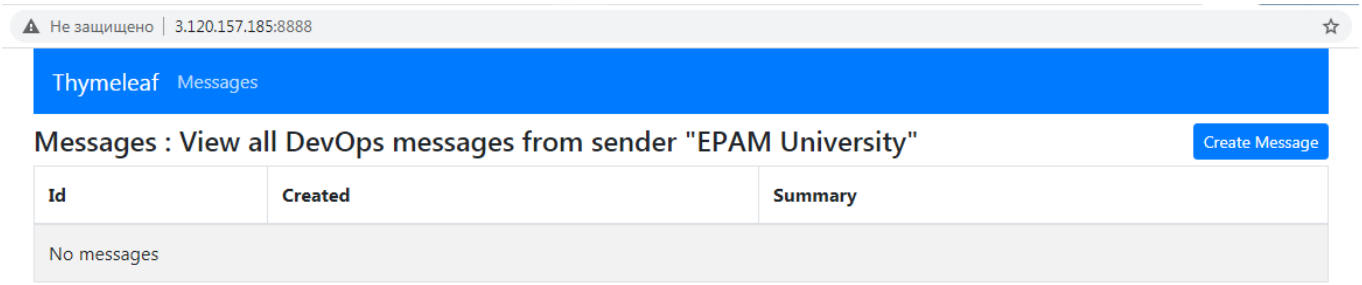


And our pipeline starts again.

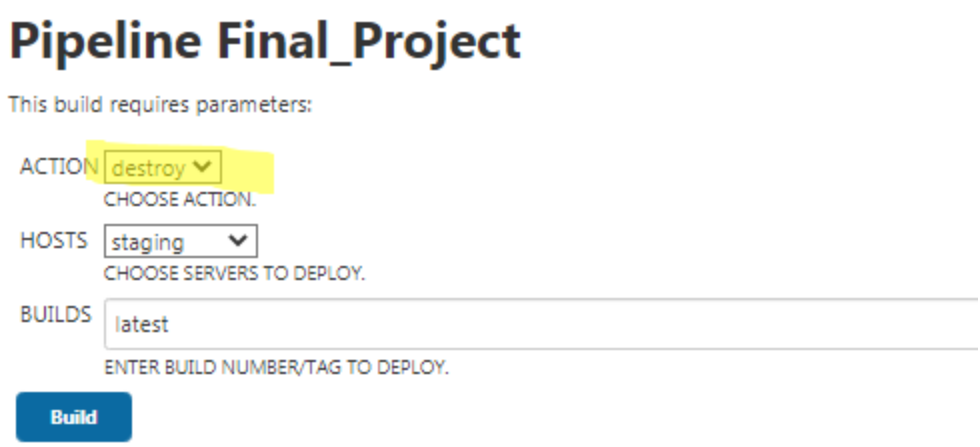


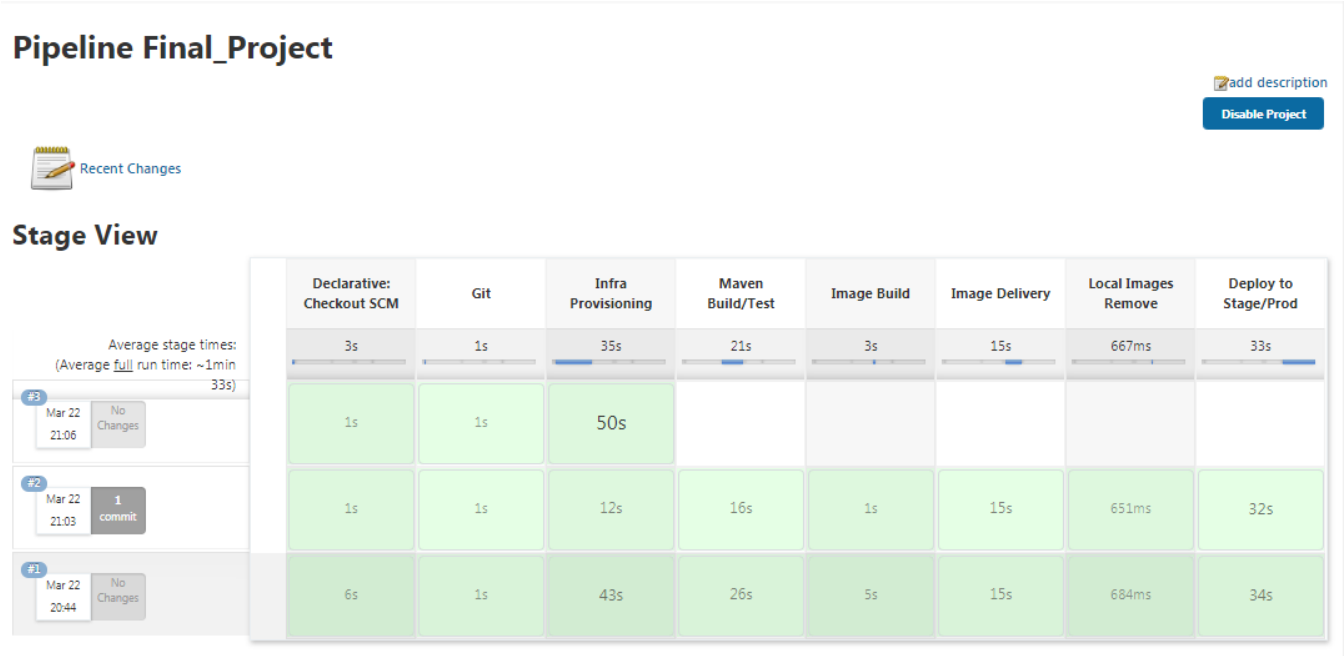


Check up our web page.



Now I will destroy all our infrastructure.






```
21:06:56  Destroy complete! Resources: 3 destroyed.
```

CONCLUSION

This was a basic example of how to work with Pipelines and integrate different components of your deployments.
There are possibilities of creating numerous complex integrations through Pipelines as you go ahead.
Some ideas to go further with Jenkins:

- We can create multibranch pipeline for different branches with their own environments;
- Set a notification by Email/Telegram/Slack with the status and/or output of your Pipeline;
- Store Terraform tfstate in Amazon S3 Bucket;
- Make tags with short commit hash and Git Tag to Docker image instead Build_Number;
- Create a User in Dockerfile for better security.

Structure of my finalproject directory on a local machine.

```
.
├── Ansible
│   ├── ansible.cfg
│   └── create_container.yml
├── Docker
│   └── Dockerfile
├── Jenkinsfile
├── Provision
│   ├── builder_data.sh
│   ├── jenkins_data.sh
│   ├── main.tf
│   └── variables.tf
├── Spring-App
│   ├── ci
│   ├── CODE_OF_CONDUCT.adoc
│   ├── CONTRIBUTING.adoc
│   ├── eclipse
│   ├── git
│   ├── LICENSE.txt
│   ├── mvnw
│   ├── mvnw.cmd
│   ├── pom.xml
│   ├── README.adoc
│   ├── spring-boot-project
│   ├── spring-boot-samples
│   ├── spring-boot-samples-invoker
│   ├── spring-boot-tests
│   ├── src
│   └── SUPPORT.adoc
└── Terraform
    ├── inventory.tmpl
    ├── main.tf
    ├── output.tf
    ├── servers_data.sh
    └── variables.tf
```

Structure of my GitHub repo.