Certification Project – Medicure

Healthcare Domain

Submitted by : Rahul R

Date of submission: 22-05-25

The project involves creating a fully automated pipeline with Terraform, Jenkins, Docker, Kubernetes, Ansible, Prometheus, and Grafana for monitoring.

- ✓ **Git - For version control for tracking changes in the code files**
- ✓ **Jenkins - For continuous integration and continuous deployment**
- ✓ **Docker - For containerizing applications**
- ✓ **Ansible - Configuration management tools**
- ✓ **Selenium - For automating tests on the deployed web application**
- ✓ **Terraform - For creation of infrastructure.**
- ✓ **Kubernetes – for running containerized application in managed cluster.**

Medicure project code is attached below

https://github.com/rahul-reddy16/star-agile-health-care.git

Step 1: Create a VM and Install Terraform

1.Launch a VM in AWS:

1. Instance Type: t2.medium

2. Security Group: Allow All traffic (Anywhere).

2. Connect totheVM:

Connect through mobaxterm using public ip

3. Install Terraform:

wget -O - https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(grep -oP '(?<=UBUNTU_CODENAME=).*' /etc/os-release || lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update && sudo apt install terraform

verify the installation

terraform –version

4.Establishing a connection between Terraform and AWS.

Create AWSAccess KeyandSecret Key.

Creating an Access Key and Secret Key in AWS is a crucial step to allow Terraform to authenticate and interact with AWS resources.

1. **Create an AWS IAM User with AdministratorAccess**

- **Login to AWS Management Console.**
- **Navigate to IAM>Users.**
- **Create a NewUser:EnableProgrammaticAccess.**

AttachaPolicy:

Attach the **Administrator Access** policy to the user

GenerateAccessKeys: Save the Access Key ID and Secret Access Key securely (you will need these).

1.Configure AWS CLI

Install AWS CLI (if not installed):

sudo apt install awscli

Aws configure

Enter the Access KeyID, Secret Access Key, region (e.g., us-east-1),

and output format (json).

Verify AWS Configuration:

cat ~/.aws/credentials

5.Write TerraformConfiguration File:

Create dedicated directory and navigate.

mkdir health_care && cd health_care

vimaws_infra.tf

Paste the provided Terraform configuration code to create an EC2 instance, VPC, Subnet, Security Group, and install Ansible.

```
provider "aws" {
  region                  = "us-east-2"
  shared_credentials_files = ["~/.aws/credentials"]
}
```

```
resource "tls_private_key" "mykey" {
  algorithm = "RSA"
}

resource "aws_key_pair" "aws_key" {
  key_name   = "web-key"
  public_key = tls_private_key.mykey.public_key_openssh

  provisioner "local-exec" {
    command = "echo '${tls_private_key.mykey.private_key_openssh}' > ./web-key.pem && chmod 400 ./web-key.pem"
  }
}

resource "aws_vpc" "sl-vpc" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "sl-vpc"
  }
}

resource "aws_subnet" "subnet-1" {
  vpc_id                  = aws_vpc.sl-vpc.id
  cidr_block              = "10.0.1.0/24"
  map_public_ip_on_launch = true

  depends_on = [aws_vpc.sl-vpc]
```

```
  tags = {
    Name = "sl-subnet"
  }
}

resource "aws_route_table" "sl-route-table" {
  vpc_id = aws_vpc.sl-vpc.id

  tags = {
    Name = "sl-route-table"
  }
}

resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.subnet-1.id
  route_table_id = aws_route_table.sl-route-table.id
}

resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.sl-vpc.id

  depends_on = [aws_vpc.sl-vpc]

  tags = {
    Name = "sl-gw"
  }
}

resource "aws_route" "sl-route" {
```

```
  route_table_id         = aws_route_table.sl-route-table.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.gw.id
}

variable "sg_ports" {
  type    = list(number)
  default = [8080, 80, 22, 443]
}

resource "aws_security_group" "sl-sg" {
  name   = "sg_rule"
  vpc_id = aws_vpc.sl-vpc.id

  dynamic "ingress" {
    for_each = var.sg_ports
    iterator = port
    content {
      from_port   = port.value
      to_port     = port.value
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
```

```
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "myec2" {
  ami           = "ami-005fc0f236362e99f"
  instance_type = "t2.medium"
  key_name      = aws_key_pair.aws_key.key_name
  subnet_id     = aws_subnet.subnet-1.id

  vpc_security_group_ids = [aws_security_group.sl-sg.id]

  tags = {
    Name = "Health_Care"
  }

  provisioner "remote-exec" {
    connection {
      type        = "ssh"
      user        = "ubuntu"
      private_key = tls_private_key.mykey.private_key_pem
      host        = self.public_ip
    }

    inline = [
      "sudo apt update",
      "sudo apt install -y software-properties-common",
      "sudo add-apt-repository --yes --update ppa:ansible/ansible",
      "sudo apt install -y ansible"
```

```
    ]
  }
}
```

Initialize and Apply Terraform:

terraform init- Prepares the working directory for Terraform by downloading plugins and setting up the backend

terraform validate- Checks for syntax errors and configuration validity.

terraform plan- Generates and displays an execution plan without making changes.

Terraform apply--auto-approve-Executes the plan and creates or updates infrastructure without requiring manual confirmation.

Instance has been created in us-east-2

Step 2: Configure Ansible on the Created EC2Instance

1. Connect to the Newly Created EC2Instance:
2. Use SSH with the generated private key (web-key.pem).
3. Check Ansible Installation
4. ansible—version

Install Java, git and docker Using Ansible Playbook:

Create a dedictaed directory fro ansible and create ansible playbook in it.

Create an Ansible playbook:

Vim health_care.yml Add the following

```
- name: Install and set up DevOps tools (Java, Git, Docker)
  hosts: localhost
  become: true

  tasks:
    - name: Update the apt repo
      apt:
        update_cache: yes

    - name: Install multiple packages
      apt:
        name: "{{ item }}"
        state: present
      loop:
        - git
        - docker.io
        - openjdk-17-jdk

    - name: Start Docker service
      service:
        name: docker
        state: started
        enabled: true
```

Run the playbook:

ansible-playbook health_care.yml

Verify installations.

Step 3: Installing and setup Jenkins CI/CDPipeline

1. Install Jenkins.

ADD GPG KEY:

```
sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
```

ADD Jenkins repository:

```
echo "deb [signed-by=/etc/apt/keyrings/jenkins-keyring.asc]" \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
```

Update

```
sudo apt-get update
```

Install

```
sudo apt-get install Jenkins
```

Start and EnableJenkins:

```
sudo systemctl start jenkins
sudo systemctl enable jenkins
```

Verify:

```
sudo systemctl status Jenkins
```

1. **Now access to the Jenkins dashboard using below url:**
   http://3.139.78.4:8080
   Default port: 8080.
2. **Set upplugins andadminuser.**
   cat /var/lib/jenkins/secrets/initialAdminPassword

3. **Configure Maven:**
   Go to Manage Jenkins>Global ToolConfiguration. Add Maven and set it up.

Make sure to give same name for maven in pipeline.

4. **Write docker file and add it in your github repository.**

```
FROMopenjdk:11
ARGJAR_FILE=target/*.jar
COPY${JAR_FILE} app.jar
EXPOSE8082
ENTRYPOINT["java","-jar","/app.jar"]
```

5. **Create Pipeline:**

   Points to be noted befor running pipeline:

   In source code go to src/main/resources/application.properties

   1. **Check server port**

2.Map the internal container port (8082) to an external port on your host machine

docker run-d-p 8082:8082 myproject1:$BUILD_NUMBER

3. Enable 8082 in your ec2security groups.

4. Also, same port number should be exposed in dockerfile.

5. Check and Configure Docker Hub Credentials in Jenkins

1. Navigate to your Jenkins Dashboard.
2. Click on "Manage Jenkins” > "Manage Credentials".
3. Under the appropriate scope (e.g., Global or a specific folder/job):
    - Click "(global)" if you're adding credentials for use across all jobs.
4. Click "Add Credentials" on the left sidebar.
5. In the "Kind" dropdown, select "Username with password".
6. Fill in the following fields:
    - Username: your-dockerhub-username
    - Password: your-dockerhub-password-or-access-token
    - ID: DOCKER_HUB_PASWD
7. Click "Save" to store the credentials.

6.Befor building the job allow Jenkins to RunDockerCommands:

Chmod -R777/var/run/docker.sock

7.Navigate to New Item >Pipeline.

```
pipeline {
   agent any

   tools {
      maven 'maven'  // Ensure 'maven' is configured in Jenkins Global Tools
   }

   environment {
      IMAGE_NAME = "health_care"
   }
```

```
stages {
    stage('Clone Repo') {
        steps {
            git 'https://github.com/Manjunath348/star-agile-health-care.git'
        }
    }

    stage('Test Code') {
        steps {
            sh 'mvn test'
        }
    }

    stage('Build Code') {
        steps {
            sh 'mvn package'
        }
    }

    stage('Build Docker Image') {
        steps {
            sh "docker build -t ${IMAGE_NAME}:${BUILD_NUMBER} ."
        }
    }

    stage('Push Docker Image to Docker Hub') {
```

```
        steps {
            withCredentials([usernamePassword(
                credentialsId: 'DOCKER_HUB_PASWD',  // Jenkins credential ID
                usernameVariable: 'DOCKER_USER',
                passwordVariable: 'DOCKER_PASS'
            )]) {
                sh """
                    echo "${DOCKER_PASS}" | docker login -u "${DOCKER_USER}" --
password-stdin
                    docker tag ${IMAGE_NAME}:${BUILD_NUMBER}
${DOCKER_USER}/${IMAGE_NAME}:${BUILD_NUMBER}
                    docker push
${DOCKER_USER}/${IMAGE_NAME}:${BUILD_NUMBER}
                """
            }
        }
    }
  }
}
```

· **Clone Repo:** Clones the GitHub repository containing the project.

· **Test Code**:Runs tests on the project using Maven (mvn test).

· **Build Code**: Packages the project into a JAR file using Maven (mvn package).

· **Build Image:**Builds a Docker image with the JARfile (docker build).

· **Push the Image to Dockerhub:**LogsintoDockerHubandpushes the newly built image to a Dockerrepository.

Build the job successfully

Verify docker image.

Verify whether docker image is successfully pushed to your docker hub.

Step 4: Installing and setup Kubernetes controller and worker.

Prerequisites:

Install docker and container-d

Install kubernetes component

Initiate kubernetes

Install Container network interface

Set Up kubernetes Master Node:

Using AWS we will create Ubuntu master server of 4 GB RAM2 CPU core → t2.medium -> connect to the instance and create k8-master.sh file to run the script

Install Docker

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installDocker.sh -P /tmp

sudo chmod 755 /tmp/installDocker.sh

sudo bash /tmp/installDocker.sh

sudo systemctl restart docker.service

Install CRI-Docker

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installCRIDockerd.sh -P /tmp

sudo chmod 755 /tmp/installCRIDockerd.sh

sudo bash /tmp/installCRIDockerd.sh

sudo systemctl restart cri-docker.service

Install kubernetes

sudo wget https://raw.githubusercontent.com/lerndevops/labs/master/scripts/installK8S.sh -P /tmp

sudo chmod 755 /tmp/installK8S.sh

sudo bash /tmp/installK8S.sh

Initialize kubernetes Master Node

sudo kubeadm init --cri-socket unix:///var/run/cri-dockerd.sock --ignore-preflight-errors=all

sudo mkdir -p $HOME/.kube

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

sudo chown $(id -u):$(id -g) $HOME/.kube/config

install networking driver -- Weave/flannel/canal/calico etc...

below installs calico networking driver

kubectl apply -f https://raw.githubusercontent.com/projectcalico/calico/v3.24.1/manifests/calico.yaml

Set Up kubernetes worker Node:

Create an Ubuntu server: instance type t2.medium(AWS), e2 medium(GCP)

OS: ubuntu 22 -> connect to the server -> here create a node.sh file to run the script

Allow all the ports in security group on your master and worker nodes.

Steps to Deploy a Docker Image in Kubernetes:

1. **Create a Deployment YAML File**

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app-deployment
  labels:
    app: my-app
spec:
  replicas: 2  # Number of pods to run
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
```

```
    spec:
      containers:
        - name: health-care
          image: reddyrahul/health_care:5  # Replace with your image and
tag
          ports:
            - containerPort: 8082  # Replace with the port your app listens on
```

2. Apply the Deployment

```
kubectl apply-f deployment.yaml
```

3.Create the Node Port Service

Create another YAML file named service.yaml with the following content:

```
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8082
  type: NodePort
```

Apply the service:

```
kubectl apply-f service.yaml
```

5.Verify the Deployment and Service

CheckthePods:

kubectl get pods

kubectl get svc

Verify that the service has a auto assigned NodePort (32411)

Access the container.

http://< public ip of worker node>:32411

http://18.221.171.38:33411/

STEP4: WEBHOOK CONFIGURATION:

Go to your Jenkins dashboard and add build trigger- GitHub hook trigger for GITScm polling

Configuring a webhook in Git is a crucial step to enable automated actions, such as triggering a CI/CD pipeline in Jenkins or notifying an application about repository events.

Step 1: Open Your Repository Settings

1. Navigate to the repository where you want to configure the webhook.

Step 2: Find the Webhooks Section

Goto Repository Settings > Webhooks.

Step 3: Add a New Webhook

Fill in the webhook details:

Payload URL:

· This is the endpoint that will receive the webhook payload.

· http:///github-webhook/

Step 4:Commit changes and save

Note : (I have missed out the screen shots of configuration)

Step 5: Setup Monitoring with Prometheus and Grafanain K8S controller

Monitor your Kubernetes (K8s) cluster with Prometheus and Grafana

Steps to set up Prometheus and Grafana:

1. Install Prometheus

Prometheus is an open-source monitoring and alerting toolkit used to collect and store metrics in Kubernetes.

You can install Prometheus using Helm (a package manager for Kubernetes) for easy installation.

Install Helm (if not already installed):

curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 | bash

Add Prometheus Helm chart repository:

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm repo add grafana https://grafana.github.io/helm-charts

helm repo update

Install Prometheus and grafana using Helm:

kubectl create namespace monitoring

#Createmonitoring namespace

Helm install prometheus prometheus-community/kube-prometheus stack--namespace monitoring

This commandinstalls both Prometheus and Grafana along with other monitoring components like node-exporter, kube-state metrics, and Alertmanager.

2.Verify Prometheus Installation

After the installation completes, you can check if the Prometheus components are running by checkingthe pods inthe monitoring namespace:

kubectl get pods-n monitoring

You should see pods like prometheus-xxxx, grafana-xxxx, and other components related to monitoring**.**

3 Access Prometheus Dashboard

You can access Prometheus through a port-forwarding command (for testing)

kubectl port-forward svc/prometheus-kube-prometheus prometheus-n monitoring 9090:9090

This commandcreatesservice type cluster IP

Below is the command to swich it to node port type.

kubectl patch svc prometheus-kube-prometheus-prometheus-n monitoring-p '{"spec": {"type": "NodePort"}}'

Check NodePort:

Access prometheus dashboard.

http:// 3.136.112.147:31978/

note : (I have missed out the Prometheus dashboard screenshort)

4.Access GrafanaDashboard

Grafana will be installed alongside Prometheus. To access it, you can port-forward it similarly:

kubectl port-forward svc/prometheus-grafana-n monitoring 3000:80

This command again creates service type cluster IP

Below is the command to swichit to node port type.

kubectl patch svc prometheus-grafana-n monitoring-p '{"spec": {"type": "NodePort"}}'}'

Check nodeport:

kubectl get svc-n monitoring prometheus-grafana

http://3.136.112.147:31371/

The default login credentials are:

- Username:admin
- Password:prom-operator

5.Configure Grafanawith PrometheusDataSource

Once you're inside Grafana, follow these steps to connect it to Prometheus as a datasource:

- GotoConfiguration(gearicon) >DataSources.

Select Prometheusand set the URLtohttp:// 3.136.112.147:31978/

Click Save & Test to ensure the connection is successful.

6. Import Grafana Dashboards

Grafana has pre-built dashboards for Kubernetes monitoring that you can import.

- GotoCreate(plusicon)> Import
- Enter kubernetes monitoring dashboard soruse specific dashboard IDs like 186 for Kubernetes
- Select Load and then select database as promrtheus and import

- Now,you'll have an interactive dashboard displaying Kubernetes metrics, including pod status, node health, CPU usage, memory usage, and more.

Grafana
Home › Dashboards › Node Exporter Full
Datasource prometheus
Job kubernetes-service-endpoints
Nodename ip-172-31-20-137
Instance 172.31.20.137:9100
Last 24 hours
Refresh 1m
CPU Basic
Busy System
Busy User
Busy Iowait
Busy IRQs
Busy Other
Idle
Memory Basic
Total
Used
Cache + Buffer
Free
Swap used
Network Traffic Basic
Disk Space Used Basic
/boot/efi
/boot
/run
/run/lock
CPU / Memory / Net / Disk (10 panels)

Grafana
Home › Dashboards › Node Exporter Full
Datasource prometheus
Job kubernetes-service-endpoints
Nodename ip-172-31-20-137
Instance 172.31.20.137:9100
Last 24 hours
Refresh 1m
Node Exporter
Node Exporter Scrape Time
Exporter Process CPU Usage
Exporter Processes Memory
Exporter File Descriptor Usage
Node Exporte...