Infrastructure Provision Using Terraform

ansible-host-sonarqube server is used for ansible and sonarqube and main-instance is used whole for jenkins and kubernetes,argo-cd,trivy,docker,prometheus and grafana

Step 1: Provision the Backend

1. Navigate to the `terraform/backend` directory.
2. Initialize Terraform by running:

```
terraform init
```

3. Apply the backend configuration:

```
terraform apply
```

Step 2: Provision the Infrastructure

1. Navigate to the `terraform/infra` directory.
2. Initialize Terraform:

```
terraform init
```

3. Apply the infrastructure configuration to provision the following:
    - VPC
    - IAM
    - EKS Cluster
    - Two EC2 instances:
        - One for the Ansible host
        - One as the main instance

```
terraform apply
```

Configuration Installation Using Ansible

Step 1: SSH Key Setup

1. SSH into the Ansible host:

2. Generate SSH keys on the Ansible node:

```
ssh-keygen
```

    - The keys will be created in `/home/ec2-user/.ssh`.
    - You will see two files: a public key (`.pub`) and a private key.

3. Copy the public key to the main server: in cd .ssh and authorized_keys

Step 2: Configure Ansible Hosts

1. Edit the Ansible hosts file:

```
sudo vi /etc/ansible/hosts
```

2. Add the main server under the `servers` group:

```
[servers]
[Main-server-public-IP]
```

3. run ansible-playbook ansible-sonarqube.yaml as localhost

Step 3: Install Dependencies Using Ansible

1. Run the playbook to install dependencies:

```
ansible-playbook main-server.yaml
```

SonarQube Setup

Goal

Set up the SonarQube server, record the Sonar token, and configure a webhook.

Step 1: Access SonarQube

1. Open SonarQube in a browser using the following URL:

```
http://[SonarQube-public-IP]:9000
```

2. Log in with the default credentials:
    - Username: `admin`
    - Password: `admin`

Step 2: Generate Sonar Token

1. Navigate to:
    - `Administration > Security > Users`
2. Click the menu icon (four lines to the left of the settings button).
3. Provide a name and expiry date, then generate a token.
4. Record the token (e.g., `squ_cba1e2403d6c70fbe99047f170b447b4d71b1b49`). **This token is shown only once.**

Step 3: Configure Webhook

1. Navigate to:
    - `Administration > Configurations > Webhooks`
2. Create a webhook with the following details:
    - Name: [Your Webhook Name]
    - URL: `http://[Jenkins-URL]:8080/sonarqube-webhook/`

Jenkins Setup

Goal

Ensure the Jenkins pipeline runs successfully by installing and configuring:

- SonarQube Scanner
- Node.js

- Bind necessary variables to credentials.

Step 1: Access Jenkins

1. Open Jenkins in a browser using the following URL:

```
http://[Jenkins-public-IP]:8080
```

2. Retrieve the initial admin password:

```
cat /var/lib/jenkins/secrets/initialAdminPassword
```

3. Log in and set up Jenkins.

Step 2: Install Node.js and SonarQube Scanner

1. Navigate to:
    - `Manage Jenkins > Plugins > Available Plugins`
2. Install:
    - Node.js
    - SonarQube Scanner
3. Configure tools:
    - Go to `Manage Jenkins > Tools >` click add on nodejs and sonar-scanner
    - Add tools with the following names:
        - Node.js: `nodejs`
        - SonarQube Scanner: `sonar-scanner`
    - Enable automatic installation.

Step 3: Configure Environment Variables

1. Navigate to:
    - `Manage Jenkins > System >` and check the environment variables button
2. Add SonarQube server details:
    - Navigate to `Manage Jenkins > System > SonarQube Servers`.
    - Add a new server:
        - Name: [SonarQube Server Name] should match the code
        - Server URL: [SonarQube URL]
        - Token: Use the token generated earlier (add it as `Secret Text`).

Jenkins Pipeline Configuration

Step 1: Add Node.js and SonarQube Scanner

Include the following `tools` block in the `Jenkinsfile`:

```
agent {
    tools {
        nodejs 'nodejs'  // Matches Node.js tool name
    }
}
environment {
    SCANNER_HOME = tool 'sonar-scanner'  // Matches SonarQube Scanner tool name
}
```

Step 2: SonarQube Scan Code

Use the following code for SonarQube scanning:

```
withSonarQubeEnv('sonar-server') {   // Matches the SonarQube server name which is configured in system
    sh ''' $SCANNER_HOME/bin/sonar-scanner \
        -Dsonar.projectName=frontend \
        -Dsonar.projectKey=frontend \
        -Dsonar.sources=. '''   // Scans all directories and subdirectories
}
```

Step 3: Quality Gate

Add a quality gate stage:

```
stage('Quality Gate frontend') {
    steps {
        script {
            timeout(time: 1, unit: 'MINUTES') {  // Waits for a maximum of 1 minute
                def qualityGate = waitForQualityGate()  // Waits for quality gate result
                if (qualityGate.status != 'OK') {  // Fails build if status is not OK
                    error "Quality Gate failed: ${qualityGate.status}"
                } else {
                    echo "Quality Gate passed"
                }
            }
        }
    }
}
```

Step 4: Bind Credentials to Variables

1. Go to `Pipeline Syntax` in Jenkins.
2. Choose `withCredentials: Bind credentials to variables`.
3. Add credentials as secret text with essential information.
4. Ensure the variable names match those used in the pipeline code.

Step 5: Generate GitHub Token

1. Go to `Settings > Developer Settings > Personal Access Token > Tokens (Classic)` in GitHub.
2. Generate a token.

Main Server Configuration

1. SSH into the main server

AWS Load Balancer Controller and ArgoCD Setup Documentation

Prerequisites

Server Configuration

1. **Configure AWS CLI**:

```
aws configure
aws eks update-kubeconfig --region <region> --name <cluster-name>
```

2. **Setup Docker for Jenkins**:

```
usermod -aG docker jenkins
systemctl restart docker
systemctl restart jenkins
```

Goal: Install AWS Load Balancer Controller and ArgoCD

AWS Load Balancer Controller Installation

Tagging VPC and Subnets

Edit and replace with the appropriate VPC and public subnet IDs:

```
aws ec2 create-tags \
    --resources vpc-07e9c6adda21927b1 \
    --tags Key=kubernetes.io/cluster/three-tier-project,Value=shared

aws ec2 create-tags \
    --resources subnet-0dab413b0390d2769 \
    --tags Key=kubernetes.io/cluster/three-tier-project,Value=shared Key=kubernetes.io/role/elb,Value=1

aws ec2 create-tags \
    --resources subnet-027ab67a23e60dbd0 \
    --tags Key=kubernetes.io/cluster/three-tier-project,Value=shared Key=kubernetes.io/role/elb,Value=1
```

Install Helm

```
sudo yum install -y curl jq
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

Create IAM Policy for Load Balancer Controller

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.5.4/docs/install/iam_policy.json

aws iam create-policy --policy-name AWSLoadBalancerControllerIAMPolicy --policy-document file://iam_policy.json
```

Associate OIDC Provider

```
eksctl utils associate-iam-oidc-provider --region=us-east-1 --cluster=three-tier-project --approve
```

Create IAM Service Account

```
eksctl create iamserviceaccount \
  --cluster=three-tier-project \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::537124971455:policy/AWSLoadBalancerControllerIAMPolicy \
  --approve \
  --region=us-east-1 \
  --override-existing-serviceaccounts
```

Create a Policy for ALB Ingress

Create a policy named **alb-ingress** with the following permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:*",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeInstances",
        "ec2:DescribeNetworkInterfaces",
        "ec2:CreateSecurityGroup",
        "ec2:CreateTags",
        "ec2:DeleteTags",
        "iam:CreateServiceLinkedRole",
        "iam:GetServerCertificate",
        "iam:ListServerCertificates",
        "cognito-idp:DescribeUserPoolClient",
        "waf-regional:GetWebACL",
        "waf-regional:GetWebACLForResource",
        "waf-regional:AssociateWebACL",
        "waf-regional:DisassociateWebACL",
        "wafv2:GetWebACL",
        "wafv2:GetWebACLForResource",
        "wafv2:AssociateWebACL",
        "wafv2:DisassociateWebACL",
        "shield:GetSubscriptionState",
        "shield:DescribeProtection",
        "shield:CreateProtection",
        "shield:DeleteProtection"
      ],
      "Resource": "*"
    }
  ]
}
```

Attach the Policy to the Role

1. Search for the role **AmazonEKSLoadBalancerControllerRole** in the AWS IAM section.
2. Attach the **alb-ingress** policy to this role.
3. Verify the policy is attached successfully.

Install the Load Balancer Controller

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update

helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  --namespace kube-system \
  --set clusterName=three-tier-project \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --set region=us-east-1 \
  --set vpcId=vpc-07e9c6adda21927b1

kubectl get deployment -n kube-system aws-load-balancer-controller
```

Install ArgoCD

Install Helm

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

Create Namespace and Install ArgoCD

```
kubectl create namespace argocd

helm repo add argo-cd https://argoproj.github.io/argo-helm
helm repo update

helm install argocd argo-cd/argo-cd --namespace argocd

kubectl get svc -n argocd
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'

kubectl get svc argocd-server -n argocd
```

Retrieve Load Balancer DNS and Admin Credentials

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo
```

- **Default Username**: `admin`
- **Default Password**: The output of the command above (e.g., `twAhFDiPdhll1zDu`).

ArgoCD Setup Process

1. Navigate to the ArgoCD Load Balancer DNS.
2. Log in with the default credentials.
3. Create a new application:
    - **Name**: Specify the app name.
    - **Project**: Default.
    - **Sync Policy**: Automatic (check `Prune Resources`, `Self Heal`, and `Auto Create Namespace`).
    - **Source**: Provide the repository URL and path to the Kubernetes resources.
    - **Destination**: Select the cluster URL and specify the namespace.
4. Save the configuration.

Retrieve the Ingress Address

```
kubectl get ingress -n three-tier-app
```

Update the Route 53 record to point to the ingress address using the load balancer endpoint.

Install Prometheus and Grafana

Install Helm

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

Add Prometheus Community Helm Repository

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

Create Namespace and Install Prometheus and Grafana

```
kubectl create namespace prometheus

helm install stable prometheus-community/kube-prometheus-stack -n prometheus

kubectl get pods -n prometheus
kubectl get svc -n prometheus
```

Configure Load Balancers for Prometheus and Grafana

```
kubectl patch svc stable-kube-prometheus-sta-prometheus -n prometheus -p '{"spec": {"type": "LoadBalancer"}}'
kubectl patch svc stable-grafana -n prometheus -p '{"spec": {"type": "LoadBalancer"}}'

kubectl get svc -n prometheus
```

Access Prometheus and Grafana

- **Prometheus**: `<LoadBalancerDNS>:9090`
- **Grafana**: `<LoadBalancerDNS>` (port 80).

Grafana Credentials

- **Username**: `admin`
- **Password**: `prom-operator` (default).

To retrieve the Grafana password if the default does not work:

```
kubectl get secret --namespace prometheus stable-grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo
```