/devopsshack

Real-Time Corporate DevOps Flow

Setting up Infrastructure Using IAC | Part-1

In this document, we'll cover the step-by-step process to create and configure infrastructure using Infrastructure as Code (IAC) tools such as Terraform . We'll create a private networking environment, set up an Elastic Kubernetes Service (EKS) cluster, deploy virtual machines (VMs).

1. Creating the Infrastructure with Terraform

Step 1: Define Infrastructure

- Create a Terraform configuration file (`main.tf`) to define the infrastructure components.
- Specify the networking environment, EKS cluster, and VMs.

Step 2: Network Configuration

- Define a private network environment using Terraform.
- Set up subnets, route tables, and security groups to ensure secure communication.

Step 3: Elastic Kubernetes Service (EKS)

- Configure Terraform to create an EKS cluster within the private network.
- Specify the desired configuration for the cluster, including node groups, instance types, and storage options.

Step 4: Virtual Machines

- Use Terraform to provision 4 virtual machines within the private network.
- Specify the instance types, operating system, and other necessary configurations.

Sample Terraform Script

Below is a Terraform script to achieve the mentioned requirements:

```
# Define AWS provider
provider "aws" {
  region = "us-west-2" # Change to your desired AWS region
}

# Create VPC
resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
}

# Create private subnet
resource "aws_subnet" "private_subnet" {
  vpc_id            = aws_vpc.my_vpc.id
  cidr_block        = "10.0.1.0/24"
  availability_zone = "us-west-2a" # Change to your desired availability zone
}

# Create EKS cluster
resource "aws_eks_cluster" "my_cluster" {
  name     = "my-eks-cluster"
  role_arn = "arn:aws:iam::123456789012:role/eks-service-role" # Change to your
EKS service role ARN
  version  = "1.21" # Change to your desired EKS version

  vpc_config {
    subnet_ids = [aws_subnet.private_subnet.id]
  }
}

# Create EKS worker nodes
resource "aws_eks_node_group" "my_node_group" {
  cluster_name    = aws_eks_cluster.my_cluster.name
  node_group_name = "my-node-group"
  node_role_arn   = "arn:aws:iam::123456789012:role/eks-node-role" # Change to
your EKS node role ARN
  subnet_ids      = [aws_subnet.private_subnet.id]
  instance_types  = ["t2.medium"]
  desired_capacity = 3
}

# Create Virtual Machines
resource "aws_instance" "my_instances" {
  count         = 4
  ami           = "ami-0c55b159cbfafe1f0" # Change to your desired AMI
  instance_type = "t2.medium"
```

```
  user_data = <<-EOF
                #!/bin/bash
                sudo apt update
                chmod 700 ~/.ssh
                chmod 600 ~/.ssh/authorized_keys
              EOF

  tags = {
    Name = "my-instance-${count.index + 1}"
  }
}

# Paste public key in authorized_keys file of newly created VMs
resource "null_resource" "copy_ssh_key" {
  depends_on = [aws_instance.my_instances]

  provisioner "remote-exec" {
    connection {
      type        = "ssh"
      host        = aws_instance.my_instances.*.public_ip[count.index]
      user        = "ubuntu" # Change to your desired username
      private_key = file("~/.ssh/your_private_key.pem") # Change to your private
key path
    }

    inline = [
      "echo 'YOUR_PUBLIC_KEY' >> ~/.ssh/authorized_keys" # Change to your public
key
    ]
  }
}
```

Make sure to replace placeholders like region, availability zone, AMI, IAM role ARNs, and public/private key paths with your actual configurations. Additionally, ensure that you have the necessary IAM roles and policies attached to your AWS account for EKS cluster creation and EC2 instance provisioning.

Setting up Infrastructure Using IAC | Part-2

In this document, we'll cover the step-by-step process to create and configure infrastructure using Infrastructure as Code (IAC) tools such as Ansible to configure VMs to install Jenkins, SonarQube Server, Nexus, and set up a Jenkins slave.

Configuring VMs with Ansible

Step 1: Install Ansible

- Set up Ansible on your local machine or a dedicated server.
- Install Ansible dependencies and configure the Ansible inventory file.

Step 2: Ansible Playbooks

- Create Ansible playbooks to configure the VMs:
    - **Jenkins Installation**: Install Jenkins on one of the VMs.
    - **SonarQube Server Installation**: Install SonarQube Server on another VM.
    - **Nexus Installation**: Install Nexus Repository Manager on another VM.
    - **Jenkins Slave Configuration**: Configure the fourth VM as a Jenkins slave.

Step 3: Playbook Execution

- Execute the Ansible playbooks to configure the VMs.
- Monitor the execution to ensure successful installation and configuration.

Sample Ansible Playbooks

Below are the Ansible playbooks to configure the VMs as requested:

1. Jenkins Installation

```
---
- name: Install Jenkins
  hosts: jenkins_vm
  become: true

  tasks:
    - name: Add Jenkins Repository
      apt_repository:
        repo: deb https://pkg.jenkins.io/debian-stable binary/
```

```
        state: present
        update_cache: yes

    - name: Import Jenkins Repository GPG Key
      apt_key:
        url: https://pkg.jenkins.io/debian/jenkins.io.key
        state: present

    - name: Install Jenkins
      apt:
        name: jenkins
        state: present

    - name: Start Jenkins Service
      service:
        name: jenkins
        state: started
        enabled: yes
```

2. SonarQube Server Installation

```
---
- name: Install SonarQube Server
  hosts: sonarqube_vm
  become: true

  tasks:
    - name: Install OpenJDK 11
      apt:
        name: openjdk-11-jdk
        state: present

    - name: Download SonarQube
      get_url:
        url: https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-
8.9.1.44547.zip
        dest: /opt/sonarqube.zip

    - name: Unzip SonarQube
      unarchive:
        src: /opt/sonarqube.zip
        dest: /opt/
        remote_src: yes

    - name: Set Permissions
      file:
        path: /opt/sonarqube
        state: directory
        mode: '0755'

    - name: Start SonarQube Service
      command: /opt/sonarqube/bin/linux-x86-64/sonar.sh start
```

3. Nexus Installation

```
---
- name: Install Nexus Repository Manager
  hosts: nexus_vm
  become: true

  tasks:
    - name: Install OpenJDK 8
      apt:
        name: openjdk-8-jdk
        state: present

    - name: Download Nexus
      get_url:
        url: https://download.sonatype.com/nexus/3/latest-unix.tar.gz
        dest: /opt/nexus.tar.gz

    - name: Unzip Nexus
      unarchive:
        src: /opt/nexus.tar.gz
        dest: /opt/
        remote_src: yes

    - name: Set Permissions
      file:
        path: /opt/nexus
        state: directory
        mode: '0755'

    - name: Start Nexus Service
      command: /opt/nexus/bin/nexus start
```

4. Jenkins Slave Configuration

```
---
- name: Configure Jenkins Slave
  hosts: jenkins_slave_vm
  become: true

  tasks:
    - name: Install Java
      apt:
        name: default-jre
        state: present

    # Additional tasks to configure Jenkins slave
```

Ensure to replace `jenkins_vm`, `sonarqube_vm`, `nexus_vm`, and `jenkins_slave_vm` with the respective host groups in your inventory file. Also, modify the tasks according to your specific requirements and environment setup.

Source Code Management

In the DevOps workflow, source code management plays a crucial role in ensuring collaboration, version control, and traceability of changes within the development team. In this section, we will outline the steps to establish an effective source code management system using Git.

1. Create a Private Git Repository

Overview:

- A private Git repository provides a secure and centralized location to store and manage your application source code.

Steps:

1. **Choose a Git Hosting Service**: Select a Git hosting service provider such as GitHub, GitLab, or Bitbucket.
2. **Create a New Repository**: Log in to your chosen Git hosting service and create a new repository.
3. **Set Repository Visibility**: Ensure that the repository is set to private to restrict access to authorized users only.
4. **Provide Repository Name and Description**: Give your repository a meaningful name and description to help identify its purpose.
5. **Initialize Repository with README (Optional)**: You may choose to initialize the repository with a README file to provide initial documentation or instructions.

2. Setup Permissions & Roles

Overview:

- Configuring permissions and roles ensures that only authorized users have access to the repository and defines their level of access.

Steps:

1. **Define Access Control Policies**: Determine who should have access to the repository and what actions they can perform (e.g., read, write, admin).
2. **Create Teams or Groups (Optional)**: Organize users into teams or groups based on their roles or responsibilities.
3. **Assign Permissions**: Assign appropriate permissions to individuals, teams, or groups based on their roles (e.g., developers, testers, administrators).
4. **Enable Two-Factor Authentication (Optional)**: Enhance security by enabling two-factor authentication for repository access.

3. Push the Application Source Code to Git Repository

Overview:

- Pushing the application source code to the Git repository makes it accessible to the development team and enables version control.

Steps:

1. **Clone the Repository**: Clone the empty repository to your local machine using the Git command-line interface or a Git client.
2. **Add Application Source Code**: Add your application source code files to the local repository directory.
3. **Commit Changes**: Commit the added files to the local repository with a descriptive commit message.
4. **Push Changes to Remote Repository**: Push the committed changes from the local repository to the remote repository on the Git hosting service.

4. Create the Required Branches

Overview:

- Branching in Git allows for parallel development, experimentation, and isolation of changes. Creating the required branches sets up the foundation for managing different stages of development.

Steps:

1. **Create Master Branch**: The master branch serves as the main branch and typically represents the production-ready version of the code.

2. **Create Development Branch (Optional)**: Create a development branch to integrate and test new features before merging them into the master branch.

3. **Create Feature Branches**: Create feature branches for each new feature or bug fix. Feature branches allow developers to work on changes independently.

4. **Create Release Branches (Optional)**: Create release branches to prepare for a new software release. Release branches facilitate stabilization and testing of features before deployment.

5. **Create Hotfix Branches (Optional)**: Create hotfix branches to address critical issues or bugs in the production environment. Hotfix branches allow for quick resolution without disrupting ongoing development.

By following these steps, you can establish a robust source code management system that ensures secure storage, controlled access, version control, and effective collaboration within your development team. This lays the foundation for successful software development and delivery in the DevOps environment.

Creating CI/CD Pipelines with Jenkins

In this document, we'll outline the steps to create CI/CD pipelines with Jenkins, including configuring Jenkins with necessary plugins and tools, setting up webhooks, and defining stages in the pipeline.

1. Configure Jenkins with Plugins & Tools

Plugins Installation

- Install Jenkins on your server.
- Navigate to the Jenkins dashboard and go to "Manage Jenkins" > "Manage Plugins".
- Install the following plugins:
    - Kubernetes Plugin: Enables Jenkins to dynamically provision Kubernetes clusters as build agents.
    - SonarQube Scanner Plugin: Integrates Jenkins with SonarQube for code analysis.
    - Nexus Platform Plugin: Provides integration with Nexus Repository Manager for artifact management.
    - Docker Pipeline Plugin: Allows Jenkins to build, tag, and push Docker images.
    - OWASP ZAP Plugin: Integrates OWASP ZAP for security scanning.
    - Pipeline Plugin: Enables Jenkins to define pipelines using Jenkinsfile.
    - Trivy Plugin: Integrates Trivy for filesystem and Docker image vulnerability scanning.
    - etc.

Tools Configuration

- Configure Jenkins to connect to Kubernetes cluster:
    - Go to "Manage Jenkins" > "Configure System".
    - Add Kubernetes cloud configuration and credentials to connect to the Kubernetes cluster.
- Configure SonarQube:
    - Go to "Manage Jenkins" > "Configure System".
    - Add SonarQube server configuration and credentials.
- Configure Nexus:

- Go to "Manage Jenkins" > "Configure System".
- Add Nexus repository manager configuration and credentials.

2. Create Jenkins Pipeline

Webhook Setup

- Set up a webhook in your Git repository to trigger Jenkins pipeline upon commits to the master branch.

Pipeline Definition

- Create a Jenkinsfile in your project repository with the following stages:
  i. Compile: Compile the application code.
  ii. Test: Run unit and integration tests.
  iii. Filesystem Scan with Trivy: Perform filesystem scanning using Trivy.
  iv. SonarQube Analysis: Run SonarQube analysis for code quality checks.
  v. Dependency Check: Check for any vulnerabilities in dependencies.
  vi. Build Application: Build the application.
  vii. Publish Application Artifact on Nexus: Publish the built artifact to Nexus repository.
  viii. Build & Tag Docker Image: Build and tag Docker image.
  ix. Scan Docker Image Using Trivy: Scan Docker image for vulnerabilities using Trivy.
  x. Push Docker Image to Docker Hub Repo: Push Docker image to Docker Hub repository.
  xi. Update Docker Image in YAML Manifest Files: Update Kubernetes YAML manifest files with the new Docker image.
  xii. Deploy the Application to Kubernetes: Deploy the application to Kubernetes cluster.
  xiii. Sleep for 60 sec: Pause pipeline execution for 60 seconds.
  xiv. Verify Deployment is Done: Check if the deployment is successful.
  xv. Run OWASP ZAP Scan: Perform security scanning using OWASP ZAP.

Sample Pipeline

```
pipeline {
    agent any

    environment {
        // Define environment variables
        NEXUS_URL = 'https://nexus.example.com'
        DOCKER_REGISTRY = 'docker.io/yourusername'
```

```
        IMAGE_NAME = 'yourimage'
        K8S_MANIFEST = 'path/to/kubernetes/manifest.yaml'
    }

    stages {
        stage('Compile & Test') {
            steps {
                // Execute compilation and testing commands
                sh 'mvn compile'
                sh 'mvn test'
            }
        }

        stage('Filesystem Scan with Trivy') {
            steps {
                // Execute filesystem scan using Trivy
                sh 'trivy /path/to/your/project'
            }
        }

        stage('SonarQube Analysis') {
            steps {
                // Execute SonarQube analysis
                withSonarQubeEnv('SonarQube') {
                    sh 'mvn sonar:sonar'
                }
            }
        }

        stage('Dependency Check') {
            steps {
                // Execute dependency check
                sh 'mvn dependency:check'
            }
        }

        stage('Build & Publish Artifact') {
            steps {
                // Build and publish application artifact to Nexus
                sh 'mvn package'
                sh "curl -v -u username:password --upload-file target/artifact.jar
$NEXUS_URL/repository/maven-releases/"
            }
        }

        stage('Build & Tag Docker Image') {
            steps {
                // Build and tag Docker image
                sh 'docker build -t $DOCKER_REGISTRY/$IMAGE_NAME:latest .'
                sh 'docker tag $DOCKER_REGISTRY/$IMAGE_NAME:latest
$DOCKER_REGISTRY/$IMAGE_NAME:$BUILD_NUMBER'
            }
        }

        stage('Scan Docker Image with Trivy') {
            steps {
                // Scan Docker image using Trivy
                sh 'trivy $DOCKER_REGISTRY/$IMAGE_NAME:latest'
            }
```

```
        }

        stage('Push Docker Image to Docker Hub') {
            steps {
                // Push Docker image to Docker Hub repository
                sh 'docker push $DOCKER_REGISTRY/$IMAGE_NAME:latest'
                sh 'docker push $DOCKER_REGISTRY/$IMAGE_NAME:$BUILD_NUMBER'
            }
        }

        stage('Update Kubernetes Manifest') {
            steps {
                // Update Docker image in Kubernetes manifest file
                sh "sed -i 's|image:.*|image:
$DOCKER_REGISTRY/$IMAGE_NAME:$BUILD_NUMBER|g' $K8S_MANIFEST"
            }
        }

        stage('Deploy Application to Kubernetes') {
            steps {
                // Deploy application to Kubernetes
                sh 'kubectl apply -f $K8S_MANIFEST'
            }
        }

        stage('Verify Deployment') {
            steps {
                // Sleep for 60 seconds and verify deployment
                sleep time: 60, unit: 'SECONDS'
                sh 'kubectl get pods'
            }
        }

        stage('Run OWASP ZAP Scan') {
            steps {
                // Execute OWASP ZAP scan
                sh 'owasp-zap-scan-command-here'
            }
        }
    }

    post {
        always {
            // Clean up or perform post-build actions
        }
    }
}
```

Conclusion

By following the steps outlined above, you can set up a robust CI/CD pipeline with Jenkins, integrating various tools and plugins to automate the software delivery process. This pipeline enables efficient testing, code analysis, artifact management, Docker image creation, deployment to Kubernetes, and security scanning, ensuring the quality and security of your application throughout the development lifecycle.

Managing Security on EKS, Jenkins Server, Sonar Server, Nexus Server

Security is a critical aspect of any DevOps setup. In this document, we'll outline how to manage security on various components of your DevOps environment, including Elastic Kubernetes Service (EKS), Jenkins server, SonarQube server, and Nexus server.

1. EKS Security

Network Security

- Utilize network policies to control traffic between pods within the cluster.
- Implement security groups and network ACLs to restrict access to EKS API server.

Authentication & Authorization

- Configure IAM roles for service accounts to grant permissions to Kubernetes resources.
- Enable RBAC (Role-Based Access Control) to control access to cluster resources.

Secrets Management

- Use Kubernetes secrets to store sensitive information like passwords and API keys.
- Encrypt secrets at rest using Kubernetes native encryption mechanisms.

Monitoring & Logging

- Set up monitoring with tools like Prometheus and Grafana to monitor cluster health and performance.
- Configure logging using tools like Fluentd or CloudWatch to capture logs for auditing and troubleshooting.

2. Jenkins Server Security

Authentication & Authorization

- Enable authentication via LDAP, Active Directory, or OAuth to control user access.
- Implement role-based access control (RBAC) to assign permissions to Jenkins users and groups.

Plugin Security

- Regularly update Jenkins plugins to ensure they are free from vulnerabilities.
- Disable unnecessary plugins to reduce attack surface.

Secure Jenkins Configuration

- Configure HTTPS to encrypt traffic between clients and the Jenkins server.
- Use Jenkins credentials plugin to securely store sensitive information like API tokens and passwords.

Backup & Recovery

- Regularly backup Jenkins configuration and data to prevent data loss.
- Test backup and recovery procedures to ensure they are reliable.

3. SonarQube Server Security

Authentication & Authorization

- Enable LDAP or Active Directory integration for user authentication.
- Define user roles and permissions within SonarQube to control access to projects and features.

Data Protection

- Encrypt sensitive data stored in the SonarQube database and file system.
- Implement regular database backups to prevent data loss.

Vulnerability Scanning

- Periodically scan SonarQube server for vulnerabilities using security scanning tools.
- Apply security patches and updates promptly to mitigate known vulnerabilities.

4. Nexus Server Security

Authentication & Authorization

- Implement fine-grained access control using Nexus roles and privileges.
- Integrate with LDAP or Active Directory for user authentication.

Repository Security

- Apply repository-level permissions to control access to artifacts.
- Enable HTTPS for secure communication between clients and Nexus server.

Data Protection

- Encrypt sensitive data stored in Nexus repositories.
- Implement backup and recovery procedures to protect against data loss.

Conclusion

By implementing robust security measures on EKS, Jenkins server, SonarQube server, and Nexus server, you can mitigate security risks and ensure the integrity and confidentiality of your DevOps environment. Regular monitoring, updates, and audits are essential to maintaining a secure infrastructure and protecting against emerging threats.

Monitoring Setup with Grafana

Monitoring is a critical aspect of any DevOps setup as it provides insights into the performance and health of deployed applications and infrastructure. Grafana is a popular open-source monitoring and visualization tool that allows you to create customizable dashboards for real-time monitoring. In this section, we'll discuss how to set up Grafana and configure it to monitor the deployed application and worker node usage.

Setting Up Grafana

Step 1: Install Grafana

- Install Grafana on a server within your infrastructure. You can follow the official installation guide based on your operating system: Grafana Installation Guide.

Step 2: Configure Data Source

- Once Grafana is installed, access the Grafana web interface (usually on port 3000).
- Log in with the default credentials (admin/admin) and change the password.
- Navigate to Configuration > Data Sources and click on "Add data source".
- Choose the appropriate data source for your monitoring data. Common choices include Prometheus, InfluxDB, and Graphite.
- Configure the connection details for the selected data source (e.g., URL, authentication).
- Test the connection to ensure it's working correctly and save the data source.

Monitoring Deployed Application

Step 3: Create Dashboards

- Go to Dashboards > Manage in the Grafana UI.
- Click on "New dashboard" to create a new dashboard.
- Add panels to the dashboard to visualize different metrics such as:
    - CPU and memory usage of the application containers.
    - Request latency and throughput.
    - Error rates and status codes.
    - Database query performance (if applicable).
- Customize the panels by selecting appropriate visualization types (e.g., graphs, gauges, tables).

- Configure queries to fetch data from the configured data source.
- Arrange the panels on the dashboard to create an informative layout.

Step 4: Set Up Alerts

- Define alert rules to notify you of any anomalies or critical events.
- Navigate to the Alerts section in Grafana and click on "New alert rule".
- Configure conditions based on metric thresholds or patterns.
- Specify notification channels (e.g., email, Slack) to receive alerts.
- Test the alert rule to ensure it triggers notifications correctly.

Monitoring Worker Node Usage

Step 5: Collect Metrics from Worker Nodes

- Configure your monitoring solution (e.g., Prometheus, Telegraf) to collect metrics from the worker nodes in your EKS cluster.
- Ensure that metrics such as CPU usage, memory usage, disk I/O, and network traffic are being collected.

Step 6: Create Additional Dashboards

- Create separate dashboards to monitor the usage and performance of worker nodes.
- Include panels to visualize metrics such as CPU and memory utilization, disk space usage, and network activity.
- Customize the dashboards to provide insights into the health and scalability of the infrastructure.

Conclusion

By following these steps, you can set up Grafana to effectively monitor your deployed application and worker node usage in your EKS cluster. Grafana's flexibility and visualization capabilities enable you to create comprehensive dashboards tailored to your monitoring needs, helping you identify issues quickly and ensure the smooth operation of your DevOps environment.