Software Requirements Specification (SRS)

DevSecOps Project

1. Introduction

This document outlines the Software Requirements Specification (SRS) for the DevSecOps project. The goal of the project is to create a fully functional, end-to-end DevSecOps pipeline on the Azure cloud platform using Jenkins, Terraform, Kubernetes, Docker, and other open-source tools. The purpose of this pipeline is to automate the process of building, testing, and deploying applications with integrated security checks at every stage.

2. Tools Used

The following tools and technologies are used in the DevSecOps pipeline:

- Azure: Cloud service platform used for provisioning and managing infrastructure.
- Terraform: Infrastructure as Code (IaC) tool for managing cloud resources.
- Jenkins: Automation server used for building, testing, and deploying code.
- Vault: A tool for securely accessing and storing credentials.
- SonarQube: A tool for static code analysis to identify bugs and vulnerabilities.
- Docker: A platform for building, running, and managing containers.
- Kubernetes: An orchestration platform for automating containerized applications.

3. Pipeline

The DevSecOps pipeline consists of various stages:

1. **Fetch**: Jenkins fetches the source code from GitHub.
2. **Scan**: The code is scanned using SonarQube for vulnerabilities and code quality.
3. **Build**: The application is built using Maven.
4. **Test**: Unit tests are run using JUnit to ensure code quality.

5. **Docker Build**: A Docker image is built and stored in a container registry.

6. **Vulnerability Scan**: The built Docker image is scanned for vulnerabilities using Trivy.

7. **Deploy**: The application is deployed to a Kubernetes cluster.

4. Detailed Workflow

The detailed workflow of the pipeline is as follows:

- The developer pushes the source code to GitHub.

- GitHub triggers Jenkins to start the build process.

- Jenkins fetches credentials from Vault and starts the build process.

- The code is scanned for vulnerabilities using SonarQube.

- If the scan passes, the code is built using Maven.

- The built application is tested using JUnit.

- A Docker image is built, pushed to the registry, and scanned for vulnerabilities.

- The image is then deployed to a Kubernetes cluster.

5. Security Approaches

Security is an integral part of the DevSecOps pipeline. The following approaches are used to ensure security at every stage:

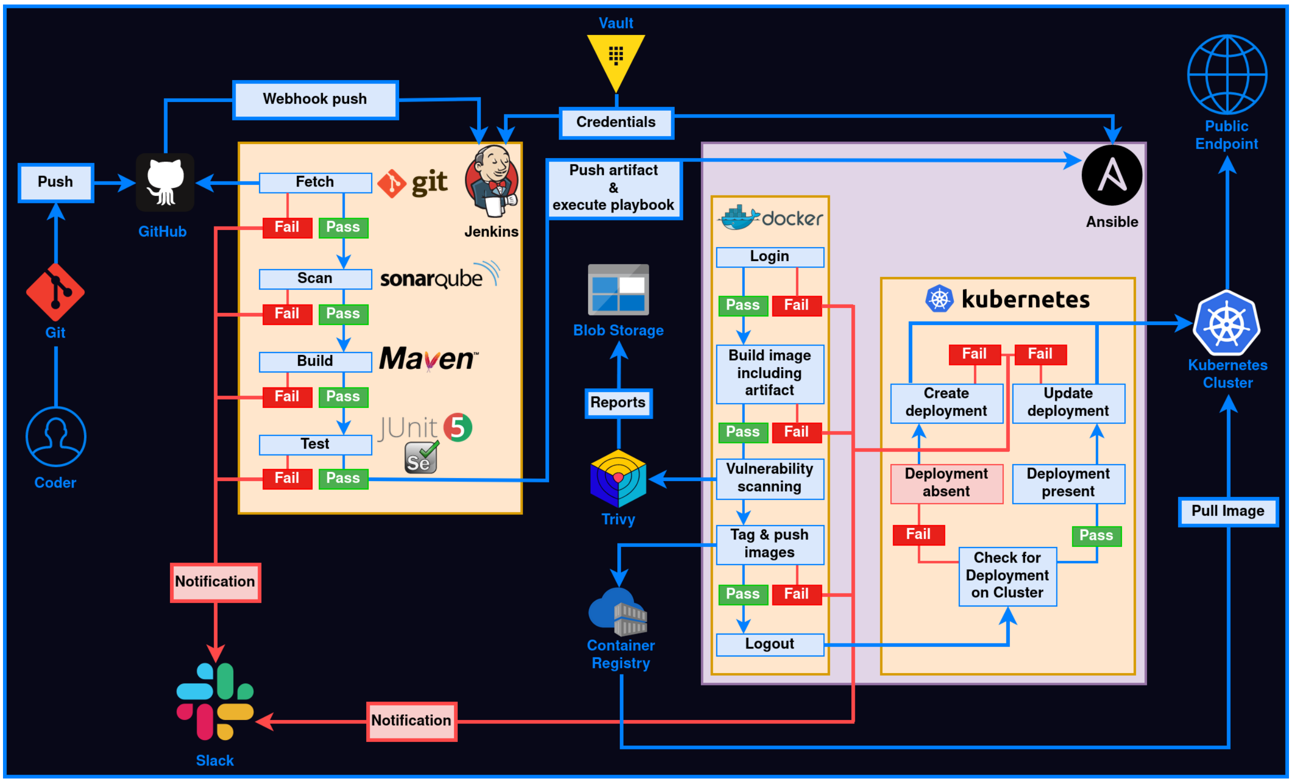
- The credentials for accessing various tools are stored securely in Vault.

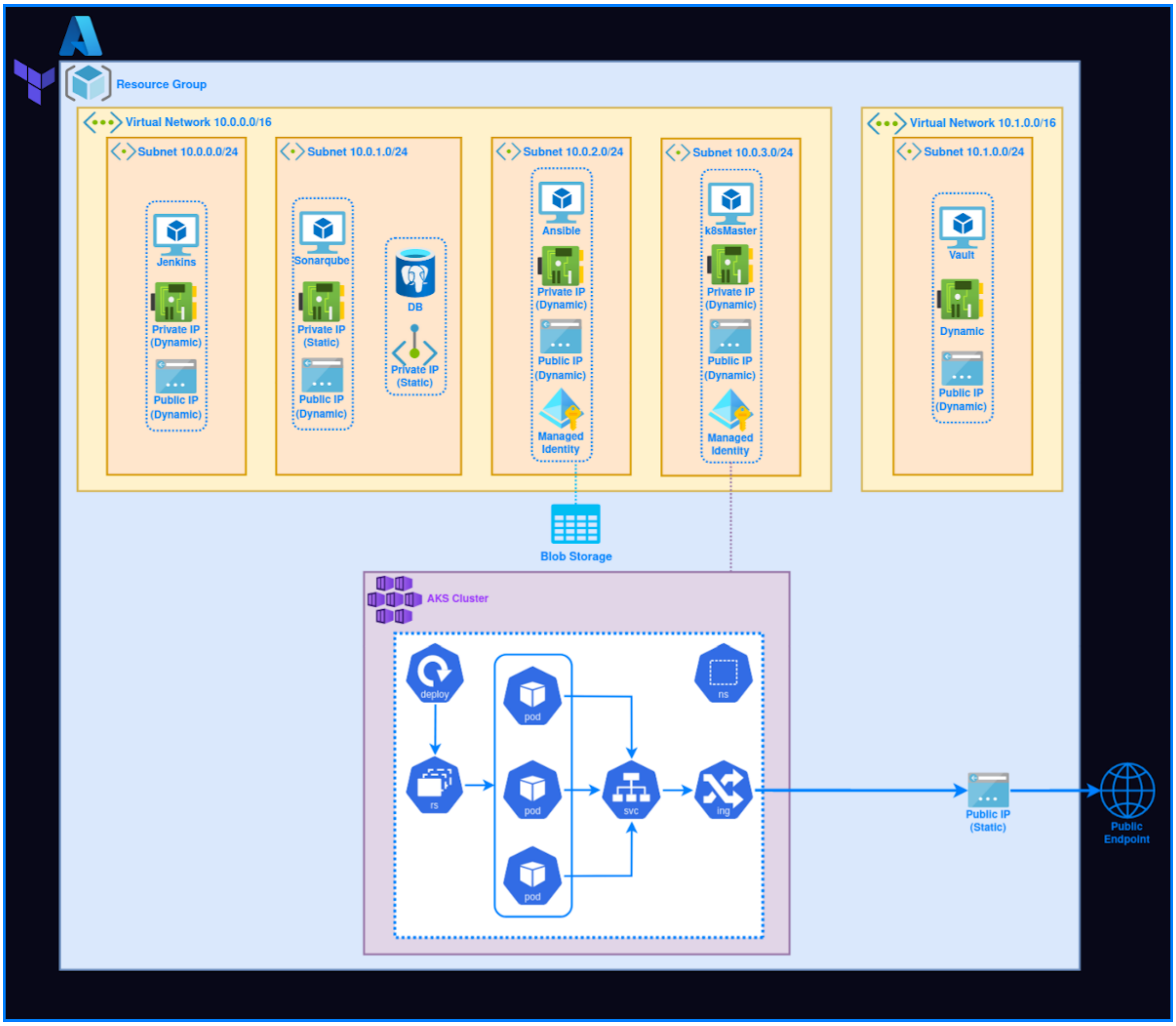
- All virtual machines are isolated in their own subnets, with strict security group rules.

- SSH access to virtual machines is restricted using public key authentication.

- The Docker images are scanned for vulnerabilities using Trivy before they are deployed.

6. Diagrams


Vault
Webhook push
Credentials
Push
GitHub
Git
Coder
Fetch
git
Jenkins
Fail
Pass
Scan
sonarqube
Fail
Pass
Build
Maven
Fail
Pass
Test
JUnit 5
Se
Fail
Pass
Push artifact & execute playbook
Blob Storage
Reports
Trivy
Container Registry
docker
Login
Pass
Fail
Build image including artifact
Pass
Fail
Vulnerability scanning
Tag & push images
Pass
Fail
Logout
Ansible
kubernetes
Fail
Fail
Create deployment
Update deployment
Deployment absent
Deployment present
Fail
Pass
Check for Deployment on Cluster
Public Endpoint
Kubernetes Cluster
Pull Image
Notification
Notification
Slack


Resource Group
Virtual Network 10.0.0.0/16
Subnet 10.0.0.0/24
Jenkins
Private IP (Dynamic)
Public IP (Dynamic)
Subnet 10.0.1.0/24
Sonarqube
Private IP (Static)
Public IP (Dynamic)
DB
Private IP (Static)
Subnet 10.0.2.0/24
Ansible
Private IP (Dynamic)
Public IP (Dynamic)
Managed Identity
Subnet 10.0.3.0/24
k8sMaster
Private IP (Dynamic)
Public IP (Dynamic)
Managed Identity
Virtual Network 10.1.0.0/16
Subnet 10.1.0.0/24
Vault
Dynamic
Public IP (Dynamic)
Blob Storage
AKS Cluster
deploy
rs
pod
pod
pod
svc
ns
ing
Public IP (Static)
Public Endpoint