Production-Ready DevSecOps Pipeline with Infrastructure as Code, Secrity Scanning & Ansible-Based Deployment

Automating Java Application Deployment with Terraform, Jenkins, Ansible, and Kubernetes on AWS

Project Objective

The primary objective of this project is to establish a robust, end-to-end CI/CD pipeline that automates the process of building, testing, and deploying a Java application. By integrating a modern DevOps toolchain including Git, Jenkins, Maven, SonarQube, Docker, Trivy, Ansible, Terraform, and Amazon EKS, this project aims to achieve rapid, reliable, and secure software delivery.

Project Goals

- **Infrastructure as Code (IaC):** Provision and manage complete AWS infrastructure using Terraform for consistency, repeatability, and version control.
- **Version Control:** Store and manage application and infrastructure code in Git to enable collaboration, tracking, and auditability.
- **Continuous Integration (CI):** Automate build, test, and validation processes using Jenkins to detect issues early in the development cycle.
- **Static Code Analysis:** Use SonarQube to continuously analyze code quality and identify bugs, vulnerabilities, and code smells.
- **Containerization:** Package the application and its dependencies into Docker containers for a consistent and portable runtime environment.
- **Container Security:** Scan Docker images with Trivy to detect and mitigate security vulnerabilities before deployment.

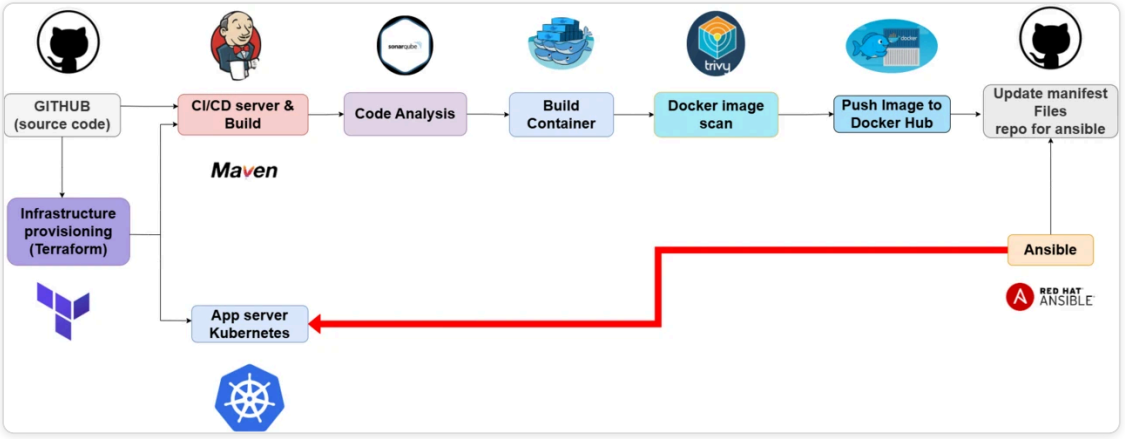
- **Continuous Deployment (CD):** Automate deployment to a Kubernetes cluster using Ansible for reliable and controlled application releases.

Architecture and Pipeline Flow

The project follows a logical flow from code commit to production deployment, orchestrated by Jenkins and leveraging a suite of best-in-class DevOps tools. The infrastructure is provisioned on AWS using Terraform.

Pipeline Flow Chart

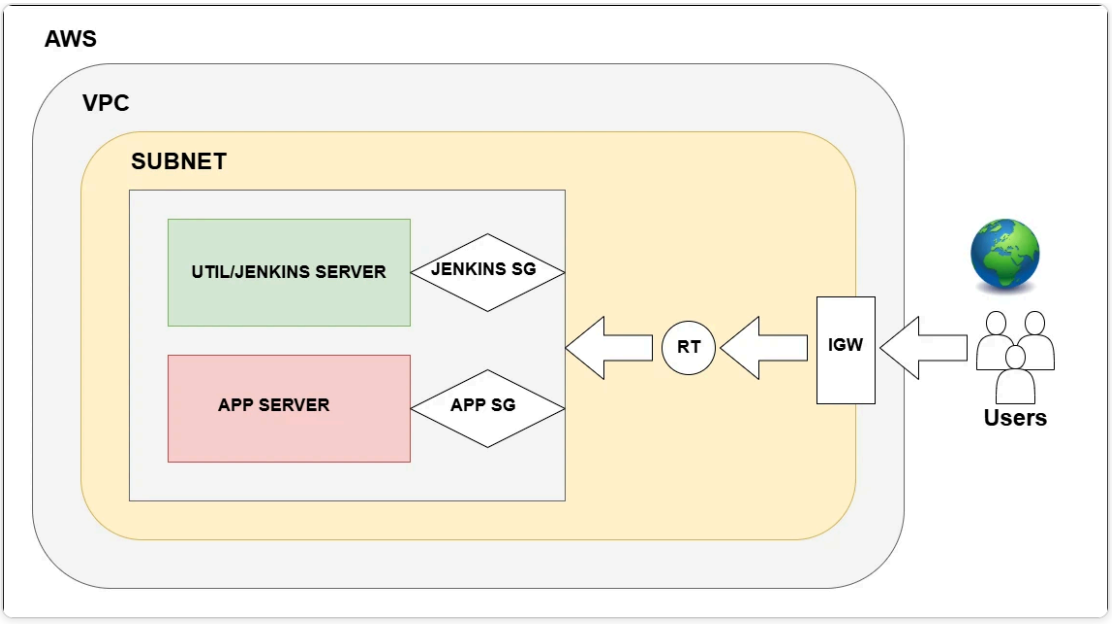
The following diagram illustrates the complete CI/CD pipeline, showing the interaction between different tools and stages.



Infrastructure Architecture

The AWS infrastructure is designed with a custom VPC to host the CI/CD tools and the application runtime. It consists of two primary server types:

- **UTIL Server:** An EC2 instance that hosts the core CI/CD components: Git, Jenkins, Ansible, Docker, Maven, SonarQube, and Trivy.
- **App Server (EKS Cluster):** An Amazon Elastic Kubernetes Service (EKS) cluster that serves as the runtime environment for the deployed application.



The CI/CD Workflow Explained

1. GitHub (Source Code Management)

Function: GitHub serves as the centralized version control system. All source code for the Java application is stored and managed here. The pipeline is triggered automatically when developers push new commits to the main branch.

Flow: A developer pushes code changes, which sends a webhook notification to Jenkins, initiating the CI process.

Repository URL: https://github.com/kadivetikavya/PROJECTS.git

2. Terraform (Infrastructure Provisioning)

Function: Terraform is used to implement Infrastructure as Code (IaC). It defines and provisions all necessary AWS resources, including the VPC, subnets, security groups, EC2 instances for Jenkins (UTIL Server), and the EKS cluster (App Server).

Flow: Before the pipeline can run, an operator executes Terraform scripts to create a consistent and version-controlled infrastructure environment.

3. Jenkins (CI/CD Orchestrator)

Function: Jenkins is the heart of the pipeline, automating and orchestrating every stage. It pulls the latest code from GitHub and executes the defined steps in sequence.

Flow: Upon receiving a trigger from GitHub, Jenkins checks out the code and begins executing the pipeline stages: build, test, analyze, package, scan, and deploy.

4. Maven (Build & Test)

Function: Maven is a powerful build automation tool for Java projects. It manages project dependencies, compiles the source code, runs unit tests, and packages the application into an executable artifact (e.g., a JAR or WAR file).

Flow: Jenkins invokes Maven to build the application as one of the initial stages in the pipeline.

5. SonarQube (Static Code Analysis)

Function: SonarQube performs static analysis on the source code to detect bugs, security vulnerabilities, and "code smells." It enforces a quality gate to maintain code health.

Flow: Jenkins triggers a SonarQube scan. If the code fails to meet the predefined quality gate standards (e.g., too many critical issues), the pipeline is halted, providing immediate feedback to developers.

6. Docker (Build & Push Image)

Function: Docker is used to containerize the Java application. A Dockerfile defines the steps to create a lightweight, portable image containing the application artifact and all its runtime dependencies.

Flow: After a successful build and analysis, Jenkins uses the Dockerfile to build a new image. This image is tagged with a unique version and pushed to Docker Hub, a central registry for storing and managing container images.

7. Trivy (Image Vulnerability Scan)

Function: Trivy is a comprehensive vulnerability scanner for container images. It inspects the image's operating system packages and application dependencies for known security vulnerabilities.

Flow: Jenkins runs Trivy to scan the newly built Docker image. If critical vulnerabilities are discovered, the pipeline is configured to stop, preventing insecure code from being deployed.

8. GitHub (Deployment Manifest Repo)

Function: A separate GitHub repository is used to store the Kubernetes deployment manifests (YAML files). This follows GitOps principles, where the Git repository is the single source of truth for the desired state of the application.

Flow: After the image is successfully built, scanned, and pushed, Jenkins automatically updates the Kubernetes deployment YAML file in this repository with the new Docker image tag. It then commits and pushes this change.

9. Ansible (Application Deployment)

Function: Ansible is an automation tool used here for configuration management and application deployment. It connects to the Kubernetes cluster and applies the desired state defined in the manifest files.

Flow: Jenkins triggers an Ansible playbook. Ansible pulls the latest deployment manifests from the dedicated GitHub repository and applies them to the EKS cluster using `kubectl` commands, triggering a rolling update of the application.

10. Kubernetes (EKS) (Application Runtime)

Function: Amazon EKS provides a managed Kubernetes control plane, simplifying the operation of the cluster. Kubernetes is the container orchestration platform that runs the application.

Flow: Kubernetes receives the deployment instructions from Ansible. It pulls the specified Docker image from Docker Hub and deploys it as pods. Kubernetes then manages the application's entire lifecycle, including scaling, networking (via services and load balancers), and self-healing.

Prerequisites, Installation, and Configuration

This section provides the necessary steps to set up the environment and tools required for the project.

a. Prerequisites

- **Java Application Code:** A Java application hosted in a Git repository. This project uses a sample application available at https://github.com/kadivetikavya/PROJECTS/tree/main/ansible-k8s-CICD-project

b. Tool Installation and Setup

Terraform Installation (on Windows)

1. Download the Terraform ZIP file from the official releases page: `terraform_1.8.5_windows_386.zip`.
2. Extract the ZIP file to a dedicated directory (e.g., `C:\terraform`).
3. Add the path to this directory (`C:\terraform`) to your system's `PATH` environment variable to make the `terraform` command accessible from any terminal.

AWS CLI Configuration (on Windows)

1. Install the AWS CLI by downloading the installer from the official AWS documentation and following the on-screen instructions.
2. Configure the CLI by running the following command in your terminal:

```
aws configure
```

3. You will be prompted to enter your credentials. Provide your AWS Access Key ID, Secret Access Key, default AWS Region (e.g., `us-east-1`), and default output format (e.g., `json`).

Infrastructure Provisioning with Terraform

Define Terraform scripts to create VPC, subnets, security groups, EC2 instances(Jenkins and app server),and other necessary AWS resources.

1. **Initialize Terraform:** Downloads the necessary providers.

```
terraform init
```

2. **Plan Changes:** Creates an execution plan to preview the changes.

```
terraform plan
```

3. **Apply Changes:** Provisions the infrastructure on AWS.

```
terraform apply
```

c. Jenkins Configuration

After provisioning the UTIL Server with Terraform, connect to it and configure Jenkins.

Required Jenkins Plugins

Navigate to **Manage Jenkins > Plugins** and install the following plugins:

- ansible
- pipeline stage view
- blue ocean
- docker
- aws credentials

Global Credentials Setup

1. **GitHub Token:**
   - In GitHub, go to **Settings > Developer settings > Personal access tokens** and generate a new token with `repo` scope.
   - In Jenkins, go to **Manage Jenkins > Credentials**, select the global domain, and click **Add Credentials**.
   - Choose **Secret text** as the kind, paste your token into the "Secret" field, and give it a descriptive ID (e.g., `github-token`).
2. **Docker Hub Credentials:**
   - In Jenkins, go to **Manage Jenkins > Credentials > Add Credentials**.
   - Choose **Username with password** as the kind.
   - Enter your Docker Hub username and password. Give it a descriptive ID (e.g., `dockerhub-credentials`).

Ansible Tool Configuration

In Jenkins, navigate to **Manage Jenkins > Tools**. Find the "Ansible installations" section and click **Add Ansible**. Configure it as follows:

- **Name:** `ansible` (or a name of your choice)
- **Path to ansible executables directory:** `/usr/bin` (or the correct path where Ansible is installed on your UTIL server)

d. Server-to-Server Connectivity for Ansible

To allow Jenkins (via Ansible) to deploy to the App Server/EKS cluster, you need to set up SSH key-based authentication.

1. **On the Jenkins (UTIL) server**, as the `ec2-user`, generate an SSH key pair:

   ```
   ssh-keygen -t rsa
   ```

   > Press Enter to accept the default file location and skip setting a passphrase.

2. Copy the public key from the Jenkins server. Display it using:

   ```
   cat ~/.ssh/id_rsa.pub
   ```

3. **On the App Server node(s)**, as the `ec2-user`, paste the copied public key into the `~/.ssh/authorized_keys` file.
4. **On the Jenkins server**, add the public IP address or DNS of your App Server node(s) to the Ansible inventory file:

   ```
   sudo nano /etc/ansible/hosts
   ```

   Add the IP under a group, for example:

   ```
   [appservers]
   54.123.45.67
   ```

5. Verify the connection from the Jenkins server:

   ```
   ansible all -m ping
   ```

e. Kubernetes Tools Installation (on App Server/Control Node)

Install `kubectl` and `eksctl` on a machine that will be used to interact with the EKS cluster (this could be the UTIL server or a dedicated bastion host).

kubectl Installation (on AWS Linux)

```
#Install kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/releas
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/releas
echo "$(cat kubectl.sha256)  kubectl" | sha256sum --check
install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
kubectl version --client
```

eksctl Installation (on AWS Linux)

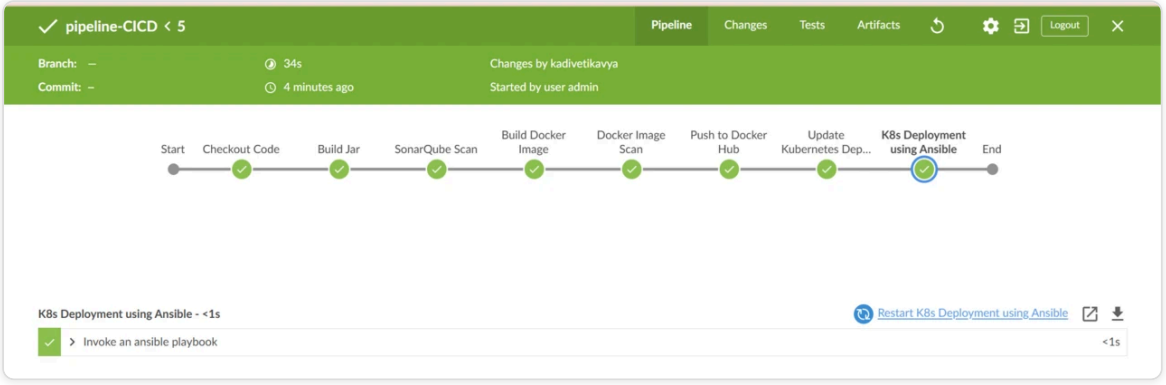
```
# Install eksctl
curl --silent --location "https://github.com/weaveworks/eksctl/releases/l
| tar xz -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
```

Pipeline Execution and Results

Once the Jenkins pipeline is configured and triggered, it will execute each stage. The following images show a successful pipeline run.

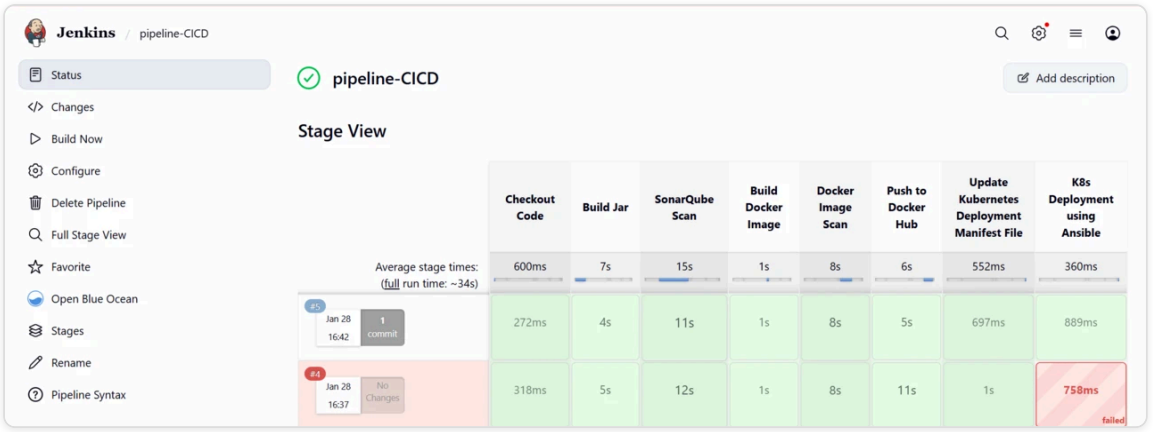
Jenkins Blue Ocean View

The Blue Ocean interface provides a modern, visual representation of the pipeline flow, clearly showing the status of each stage.



Jenkins Stage View

The classic Stage View provides a summary of recent builds and the average time taken for each stage, which is useful for identifying bottlenecks.



Deployment Verification

After the pipeline completes successfully, the final step is to verify that the application is running correctly on the Kubernetes cluster.

You can check the status of all resources in the cluster by running `kubectl get svc -n ansiblek8s`. The output below confirms that the deployment, pods, and service are all running as expected.

```
[ec2-user@app-server ~]$ sudo su -
Last login: Thu Jan 29 12:36:47 UTC 2026 on pts/3
[root@app-server ~]# kubectl get svc -n ansiblek8s
NAME          TYPE           CLUSTER-IP    EXTERNAL-IP
T(S)        AGE
app-service   LoadBalancer   10.100.25.7   adf1f5bcd753f43f1842eafd0fba4069-1375844474.us-east-1.elb.amazonaws.com
30007/TCP   66m
```

The output shows:

- Two pods for the application (my-app-...) are in the `Running` state.
- A "LoadBalancer" service (my-app-service) has been created and assigned an external IP address (an AWS ELB URL).
- The deployment (my-app) shows that 2 out of 2 replicas are ready and available.

Accessing the Application

The application is now accessible to users via the external URL provided by the LoadBalancer service. Paste the following URL into your browser to see the running application:


```
http://adf1f5bcd753f43f1842eafd0fba4069-1375844474.us-east-1.elb.amazonav
```