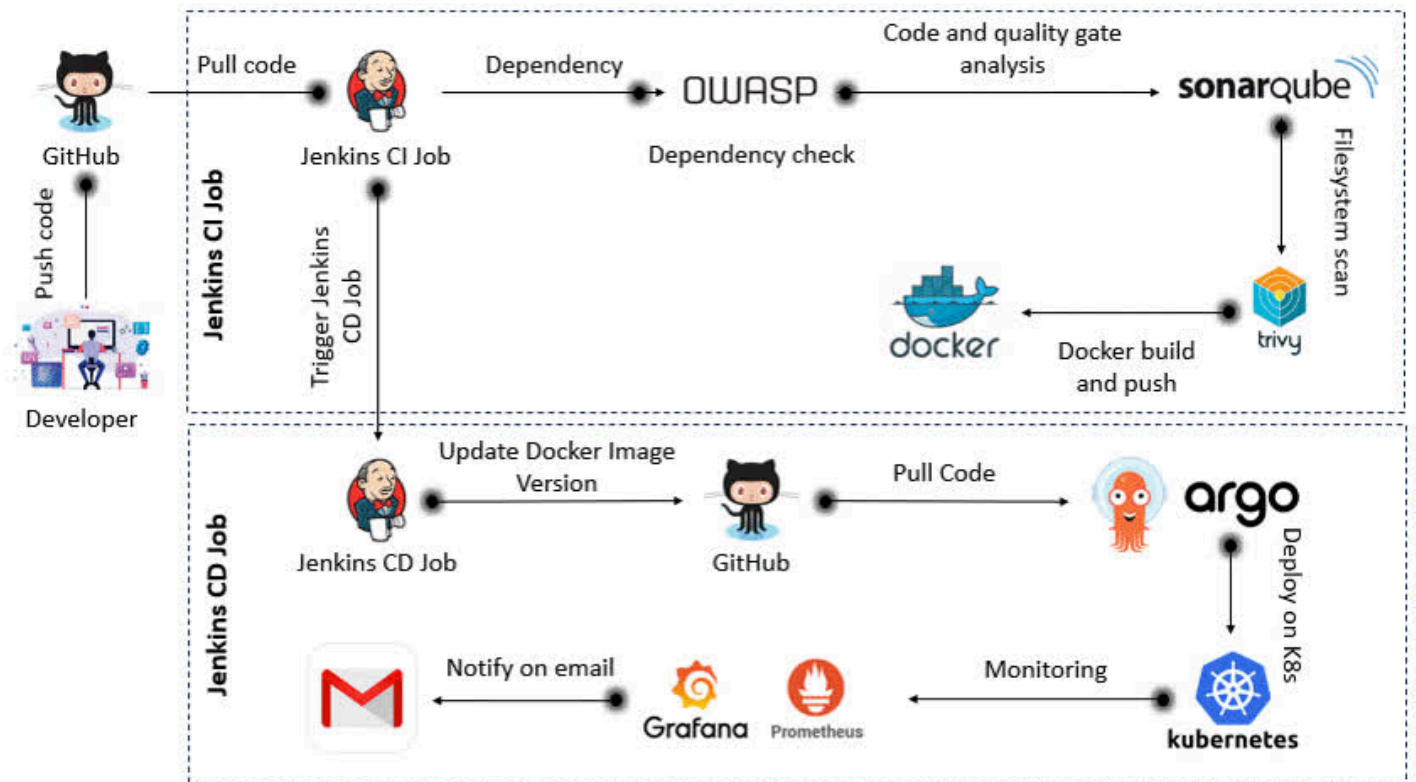


Jenkins CI/CD Setup Guide for Wanderlust Mega Project












Welcome to the Jenkins CI/CD Documentation for the Wanderlust Mega Project 🌍🌟 — a MERN stack travel blog deployed on AWS EKS with full DevSecOps tooling.



GitHub Repos

- Main Repository: [project1-wanderlust-mega](#)
- Reference Repository: [LondheShubham153/Wanderlust-Mega-Project](#)

Jenkins Timeline Overview

-  Jenkins Setup on AWS EC2 VM
-  Jenkins Installation
-  Jenkins UI/Dashboard & Jobs
-  Jenkins Freestyle Project
-  Declarative Pipeline
-  Jenkins Agents (Multi-Node Setup)
-  Declarative Pipeline - Demo
-  Shared Libraries
-  Jenkins User Management (RBAC)
-  CI/CD with Kubernetes, ArgoCD & Prometheus
-  Shared Library Realtime Use Case + Demo

- 🔑 DevSecOps: OWASP, Trivy, SonarQube Integration
- 📧 Email Notification on Pipeline Success/Failure

🔧 Jenkins Installation (on EC2)

1. 🚀 Launch an Ubuntu EC2 instance (t2.medium recommended)
2. 🍲 Install Java:
3. `sudo apt update && sudo apt install openjdk-11-jdk -y`
4. 🔑 Add Jenkins key & repository:
5. `curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null`
6. `echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`
7. 🔧 Install Jenkins:
8. `sudo apt update && sudo apt install jenkins -y`
9. `sudo systemctl start jenkins`
10. `sudo systemctl enable jenkins`
11. 🚪 Open port 8080 in EC2 Security Group

🔍 Jenkins Dashboard & Job Setup

- 🌐 Access Jenkins: `http://<EC2-Public-IP>:8080`
- 🔑 Unlock Jenkins:
- `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`
- ✅ Install suggested plugins
- 👤 Create admin user
- + Start creating Freestyle or Pipeline Jobs

🧱 Jenkins Declarative Pipeline Example

A simple pipeline with Build, Test, Deploy stages:

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo '🔧 Building...'
            }
        }
    }
}
```

```
}

stage('Test') {

    steps {

        echo '✅ Testing...'

    }

}

stage('Deploy') {

    steps {

        echo '🚀 Deploying...'

    }

}

}

post {

    success {

        echo '🎉 Pipeline Succeeded!'

    }

    failure {

        echo '❌ Pipeline Failed!'

    }

}

}
```

💡 Tip: Use environment variables, credentials, and parallel stages as your pipeline evolves!

Jenkins Shared Library Pipeline - Documentation

Overview

This Jenkins pipeline uses a shared library to modularize code and applies DevSecOps best practices by integrating tools like Trivy, OWASP, SonarQube, and Docker. It builds and pushes Docker images for both the frontend and backend of the Wanderlust Mega Project, ensuring secure and efficient CI/CD operations.

Pipeline Script

groovy

CopyEdit

```
@Library('Shared') _ // Import shared Jenkins library
```

```
pipeline {
```

```
    agent { label 'Node' } // Use labeled agent node
```

```
    environment {
```

```
        SONAR_HOME = tool "Sonar" // Reference to SonarQube scanner installation
```

```
    }
```

```
    parameters {
```

```
        string(name: 'FRONTEND_DOCKER_TAG', defaultValue: '', description: 'Setting docker image for latest push')
```

```
        string(name: 'BACKEND_DOCKER_TAG', defaultValue: '', description: 'Setting docker image for latest push')
```

```
    }
```

```
    stages {
```

```
        // Workspace Cleanup
```

```
        stage("Workspace cleanup") {
```

```
steps {  
  
    script {  
  
        cleanWs() // Clean up previous workspace files  
  
    }  
  
}  
  
}  
  
// Git Code Checkout  
  
stage("Git: Code Checkout") {  
  
    steps {  
  
        script {  
  
            code_checkout("https://github.com/DevMadhup/Wanderlust-Mega-Project.git",  
"main") // Checkout code from GitHub  
  
        }  
  
    }  
  
}  
  
// Trivy Vulnerability Scan  
  
stage("Trivy: Filesystem scan") {  
  
    steps {  
  
        script {  
  
            trivy_scan() // Run Trivy for vulnerability scanning  
  
        }  
  
    }  
  
}
```

// OWASP Dependency Check

stage("OWASP: Dependency check") {

steps {

script {

owasp_dependency() // Scan dependencies for vulnerabilities

}

}

}

// SonarQube: Code Analysis

stage("SonarQube: Code Analysis") {

steps {

script {

**sonarqube_analysis("Sonar", "wanderlust", "wanderlust") // SonarQube analysis
on code**

}

}

}

// SonarQube: Quality Gates

stage("SonarQube: Code Quality Gates") {

steps {

script {

sonarqube_code_quality() // Check SonarQube quality gates

}

}

```
}
```

```
// Export Environment Variables (Frontend & Backend)
```

```
stage("Exporting environment variables") {
```

```
parallel {
```

```
stage("Backend env setup") {
```

```
steps {
```

```
script {
```

```
dir("Automations") {
```

```
sh "bash updatebackendnew.sh" // Run backend setup script
```

```
}
```

```
}
```

```
}
```

```
}
```

```
stage("Frontend env setup") {
```

```
steps {
```

```
script {
```

```
dir("Automations") {
```

```
sh "bash updatefrontendnew.sh" // Run frontend setup script
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
// Docker: Build Images

stage("Docker: Build Images") {

    steps {

        script {

            dir("backend") {

                docker_build("wanderlust-backend-beta", "${params.BACKEND_DOCKER_TAG}",
"madhupdevops") // Build backend Docker image

            }

            dir("frontend") {

                docker_build("wanderlust-frontend-beta",
"${params.FRONTEND_DOCKER_TAG}", "madhupdevops") // Build frontend Docker image

            }

        }

    }

}

// Docker: Push Images to DockerHub

stage("Docker: Push to DockerHub") {

    steps {

        script {

            docker_push("wanderlust-backend-beta", "${params.BACKEND_DOCKER_TAG}",
"madhupdevops") // Push backend image to DockerHub

            docker_push("wanderlust-frontend-beta", "${params.FRONTEND_DOCKER_TAG}",
"madhupdevops") // Push frontend image to DockerHub

        }

    }

}
```



```

    }

    }

}

post {
    success {
        archiveArtifacts artifacts: '*.xml', followSymlinks: false // Archive test reports or build
data
        build job: "Wanderlust-CD", parameters: [
            string(name: 'FRONTEND_DOCKER_TAG', value:
"${params.FRONTEND_DOCKER_TAG}"),
            string(name: 'BACKEND_DOCKER_TAG', value: "${params.BACKEND_DOCKER_TAG}")
        ] // Trigger CD pipeline with Docker tags
    }
}
}

```

How It Works (Step-by-Step)

1. @Library('Shared') _

Purpose:

This imports the Shared Jenkins library, which contains reusable functions and logic for various stages in the pipeline, reducing redundancy and promoting best practices.

2. agent { label 'Node' }

Purpose:

This defines the agent node where the pipeline will run. In this case, it will execute on a node labeled as Node.

3. environment

Purpose:

Defines environment variables like SONAR_HOME, which links to the SonarQube tool for code analysis.

4. parameters

Purpose:

Defines two parameters for the Docker tags (FRONTEND_DOCKER_TAG and BACKEND_DOCKER_TAG) that determine the versions of the Docker images to be pushed to DockerHub.

5. stages

Purpose:

These are the main building blocks of the pipeline, each defining a stage of the CI/CD process.

Key Stages Explained

Stage 1: Workspace Cleanup

Purpose:

Clears out any residual files or workspace configurations from previous builds to ensure a fresh start for the current pipeline.

groovy

CopyEdit

script {

```
    cleanWs() // Clean up previous workspace files
```

}

Stage 2: Git: Code Checkout

Purpose:

Checks out the code from the GitHub repository and prepares it for the build process.

groovy

CopyEdit

script {

```
    code_checkout("https://github.com/DevMadhup/Wanderlust-Mega-Project.git", "main")
```

}

Stage 3: Trivy: Filesystem Scan

Purpose:

Runs Trivy, a vulnerability scanner, on the filesystem to detect any security issues in the codebase.

groovy

CopyEdit

script {

trivy_scan() // Run Trivy for vulnerability scanning

}

Stage 4: OWASP Dependency Check

Purpose:

Uses OWASP Dependency-Check to identify any known security vulnerabilities in the project dependencies.

groovy

CopyEdit

script {

owasp_dependency() // Scan dependencies for vulnerabilities

}

Stage 5: SonarQube: Code Analysis

Purpose:

Analyzes the code quality using SonarQube, ensuring that the code adheres to defined standards.

groovy

CopyEdit

script {

sonarqube_analysis("Sonar", "wanderlust", "wanderlust")

}

Stage 6: SonarQube: Code Quality Gates

Purpose:

Enforces quality gates, meaning the pipeline will fail if the code does not meet the defined quality standards.

groovy

CopyEdit

script {

```
    sonarqube_code_quality() // Check SonarQube quality gates
```

}

Stage 7: Exporting Environment Variables 🌿

Purpose:

Runs separate setup scripts for the backend and frontend, ensuring environment variables and configuration files are properly set up for each.

groovy

CopyEdit

script {

```
    dir("Automations") {
```

```
        sh "bash updatebackendnew.sh" // Run backend setup script
```

```
    }
```

}

Stage 8: Docker: Build Images 🐳

Purpose:

Builds Docker images for both the frontend and backend using the specified Docker tags and pushes them to the DockerHub repository.

groovy

CopyEdit

script {

```
    docker_build("wanderlust-backend-beta", "${params.BACKEND_DOCKER_TAG}",  
    "madhupdevops")
```

}

Stage 9: Docker: Push to DockerHub

Purpose:

Pushes the Docker images for both the frontend and backend to DockerHub for storage and further use in deployment.

groovy

CopyEdit

script {

```
    docker_push("wanderlust-backend-beta", "${params.BACKEND_DOCKER_TAG}",
"madhupdevops")

}
```

Post Actions (Success)

Purpose:

If the pipeline is successful, it archives build data (test reports or other artifacts) and triggers the CD pipeline for further deployment.

groovy

CopyEdit

archiveArtifacts artifacts: '*.xml', followSymlinks: false

build job: "Wanderlust-CD", parameters: [

```
    string(name: 'FRONTEND_DOCKER_TAG', value: "${params.FRONTEND_DOCKER_TAG}"),
```

```
    string(name: 'BACKEND_DOCKER_TAG', value: "${params.BACKEND_DOCKER_TAG}")
```

```
] // Trigger CD pipeline with Docker tags
```

Jenkins Master-Agent Setup (Multi-Node)

- Add agent node: Manage Jenkins → Nodes
- Use SSH credentials
- On the agent node:
 - `sudo apt install docker.io -y`
 - `sudo usermod -aG docker ubuntu && newgrp docker`

Jenkins Shared Libraries

- Define reusable pipeline logic
- Configure in: Manage Jenkins → Global Pipeline Libraries

Example:

```
// vars/myPipeline.groovy

def call() {

    echo '📦 Running Shared Library Logic'

}
```

🔒 Jenkins User Management (RBAC)

- Install Role-Based Authorization Strategy Plugin
- Configure roles in: Manage Jenkins → Manage & Assign Roles

🍀 CI/CD Pipeline: EKS + ArgoCD + Prometheus

- 🐳 CI: Docker image builds & pushes
- 🚢 CD: Auto-sync via ArgoCD to EKS
- 📊 Monitoring: Prometheus & Grafana (via Helm)

🔒 DevSecOps Integration

- 🐛 OWASP Dependency-Check
- ⚠️ Trivy for container scanning
- 📊 SonarQube for code quality

✉️ Jenkins Email Notifications

- Configure Gmail SMTP: Manage Jenkins → Configure System
- Enable Extended E-mail Notification
- Use App Password for Gmail 2FA

Example:

```
post {

    failure {

        emailext subject: "❌ Build Failed",

                body: "Pipeline failed. Check console output.",

                recipientProviders: [[class: 'DevelopersRecipientProvider']]

    }

}
```

}

Next Steps

-  Explore full pipeline examples in the Jenkinsfile
-  Dive into CI/CD automation in Automation, GitOps, and Kubernetes directories

Happy Building!   