# Hands-on: Secure CI/CD Pipeline Creation – Step-by-Step Guide with Real-Time Example

## 🎯 Objective

This hands-on guide walks you through the process of creating a **secure CI/CD pipeline** using popular open-source tools. It integrates security at each phase of the pipeline to detect vulnerabilities early and enforce compliance.

## 🔧 Tools and Technologies

| Purpose | Tool |
| --- | --- |
| Source Control | GitHub |
| CI/CD Pipeline | Jenkins or GitHub Actions |
| Static Analysis (SAST) | SonarQube |
| Secrets Scanning | GitGuardian / Gitleaks |
| Dependency Scanning | Snyk |
| Container Scanning | Trivy |
| Deployment Environment | Docker / Kubernetes (Minikube) |
| Notifications | Slack / Email |

## 🛠️ Step-by-Step Pipeline Setup

### 🔹 Step 1: Source Code and GitHub Setup

* Create a sample project (e.g., Node.js or Java web app)
* Push the code to a GitHub repository
* Add .gitignore, README.md, and Jenkinsfile (if using Jenkins)

### 🔹 Step 2: CI/CD Configuration (Jenkins Example)

**Install Jenkins Plugins:** - Git - Pipeline - SonarQube Scanner - Slack Notification

**Create a Jenkins Pipeline Job:** - Configure GitHub webhook to trigger the job on code push - Define the stages in a Jenkinsfile:

pipeline {  
 agent any  
 stages {  
 stage('Checkout') {  
 steps {  
 git 'https://github.com/your-repo.git'  
 }  
 }  
 stage('Secrets Scanning') {  
 steps {  
 sh 'gitleaks detect --source=.'  
 }  
 }  
 stage('Static Analysis') {  
 steps {  
 sh 'sonar-scanner'  
 }  
 }  
 stage('Dependency Scan') {  
 steps {  
 sh 'snyk test'  
 }  
 }  
 stage('Build & Test') {  
 steps {  
 sh 'npm install && npm test'  
 }  
 }  
 stage('Container Scan') {  
 steps {  
 sh 'docker build -t secure-app .'  
 sh 'trivy image secure-app'  
 }  
 }  
 stage('Deploy') {  
 steps {  
 sh 'kubectl apply -f k8s/deployment.yaml'  
 }  
 }  
 }  
 post {  
 always {  
 slackSend(channel: '#ci-cd-alerts', message: "Pipeline executed.")  
 }  
 }  
}

### 🔹 Step 3: Secure Configuration & Secrets Management

* Use environment variables or secret managers (Vault, AWS Secrets Manager)
* Never hardcode secrets in Jenkinsfile or source code

### 🔹 Step 4: Integrate Quality Gates

* In SonarQube, define a **quality gate**: build fails if code has critical security vulnerabilities
* Set up Snyk policy to reject vulnerable dependencies

### 🔹 Step 5: Post-deployment Security Checks

* Use tools like OWASP ZAP for dynamic testing
* Monitor logs with ELK or Splunk

## ✅ Real-Time Example: Securing a Node.js App

### Scenario:

An e-commerce company wants to deploy a Node.js app to Kubernetes securely.

### Outcome:

1. Code is scanned by Gitleaks before build
2. SonarQube checks for XSS/SQLi vulnerabilities
3. Snyk identifies an outdated lodash library and suggests an update
4. Docker image is scanned by Trivy
5. Secure image is deployed to Kubernetes
6. All alerts and logs are shared via Slack

## 📜 Benefits

* Early detection of security flaws
* Automated compliance enforcement
* End-to-end traceability and audit logging
* Faster secure software delivery

## 🔚 Conclusion

By following this hands-on setup, you embed security across every phase of your pipeline. This ensures faster feedback, improved developer collaboration, and significantly reduced production risk.

Would you like this document exported as a PDF or Word file?