Building a Robust CI/CD Pipeline for a Modern E-commerce Platform

By Using Jenkins, SonarQube, Maven, Nexus, and Tomcat

1. Architecting the Modern Software Delivery Engine

In the hyper-competitive world of e-commerce, the ability to rapidly and reliably deliver new features—from promotional updates to enhanced checkout flows—is a critical business advantage. This necessitates a shift from manual, error-prone deployment processes to a streamlined, automated framework. A Continuous Integration/Continuous Deployment (CI/CD) pipeline is the engine that powers this transformation.

This project provides a detailed blueprint for constructing an end-to-end CI/CD pipeline tailored for a Java-based e-commerce website. We will explore a classic, powerful, and widely adopted technology stack: **Jenkins** for orchestration, **Maven** for build and dependency management, **SonarQube** for code quality assurance, **Nexus Repository Manager** for artifact storage, and **Apache Tomcat** as the deployment server. The goal is to create a fully automated workflow that takes developer code from a Git repository to a live production environment, ensuring quality, security, and traceability at every step.

> **The Core Workflow:** The pipeline automates the following sequence:
>
> 1. **Code Commit:** A developer pushes new code to a version control system.
> 2. **Build & Analyze:** The pipeline triggers, compiling the code and running static analysis to check for bugs and vulnerabilities.
> 3. **Test:** Automated unit and integration tests are executed.

4. **Package & Store:** The application is packaged into a deployable artifact (a `.war` file) and stored in a versioned repository.
5. **Deploy:** The new version of the application is automatically deployed to the target server.

2. Deconstructing the Toolbox: Roles and Synergies

The strength of this pipeline lies in the seamless integration of specialized tools, each performing a critical function. Understanding the role of each component is key to architecting a resilient system.

2.1. Jenkins: The Pipeline Orchestrator

Jenkins is the heart of our CI/CD pipeline. It's an open-source automation server that acts as the central controller, orchestrating the entire workflow from code commit to final deployment. Its primary responsibilities include:

- **Triggering Builds:** It monitors the source code repository (e.g., Git) and automatically starts the pipeline upon detecting new commits.
- **Executing Stages:** It runs the defined steps in sequence—calling Maven to build, triggering SonarQube for analysis, and commanding the deployment to Tomcat.
- **Managing State:** It maintains the state of the pipeline, handling failures, sending notifications (e.g., via email or Slack), and providing a detailed log of every action.
- **Extensibility:** A vast ecosystem of plugins allows Jenkins to integrate with virtually any tool, making it a highly flexible orchestrator. For this pipeline, we rely on plugins for Git, Maven, SonarQube, Nexus, and Tomcat.

Modern Jenkins implementations heavily favor **Pipeline as Code** using a `Jenkinsfile`, which defines the entire CI/CD process in a script that is versioned alongside the application's source code. This provides durability, reviewability, and reproducibility for the pipeline itself.

```
Installation commands:
sudo yum update –y
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

```
sudo yum upgrade
sudo yum install java-17-amazon-corretto -y
sudo yum install jenkins -y
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
```

2.2. Maven: The Build and Dependency Master

Maven is a powerful build automation tool that standardizes how a Java project is built, packaged, and managed. Its core is the Project Object Model (POM), defined in a `pom.xml` file.

- **Dependency Management:** Maven automatically downloads and manages project dependencies (libraries and frameworks) from remote repositories, ensuring consistent builds across all developer machines and the CI server.
- **Standardized Lifecycle:** It defines a clear build lifecycle with phases like `validate`, `compile`, `test` (runs unit tests), `package` (creates the `.war` file), and `install`. When Jenkins runs `mvn package`, it executes all preceding phases in order.
- **Plugin-Based Architecture:** Maven's functionality is extended through plugins. For our pipeline, the `sonar-scanner-maven` plugin is crucial for integrating with SonarQube, and the `maven-war-plugin` is used to package our e-commerce site.And you can access it at:

  ```
  sudo yum install maven && git -y
  ```

2.3. SonarQube: The Code Quality Gatekeeper

Shipping code quickly is useless if it's buggy, insecure, or unmaintainable. SonarQube is a static code analysis platform that acts as an automated quality gate.

- **Continuous Inspection:** It analyzes source code to find bugs, security vulnerabilities (like SQL injection or cross-site scripting), and "code smells" (patterns that indicate deeper design problems).
- **Quality Gates:** This is SonarQube's most powerful feature. A Quality Gate is a set of conditions the code must meet to pass, for example: "Code Coverage on New Code >

80%", "No new Blocker-level Security Vulnerabilities", or "Maintainability Rating is A".

- **Pipeline Integration:** Jenkins is configured to pause the pipeline after the SonarQube analysis. If the Quality Gate fails, Jenkins aborts the build, preventing low-quality code from ever reaching production. This provides immediate feedback to developers.

```
Installation commands:

sudo yum install java-17-amazon-corretto -y
sudo wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube
unzip sonarqube-10.0.0.68432.zip
cp -rp /opt/sonarqube-10.0.0.68432 /home/ec2-user/
ll
cp -R sonarqube-10.0.0.68432 /opt
sudo useradd sonar
sudo passwd sonar
chown -R sonar:sonar sonarqube-10.0.0.68432
/usr/bin/java -version

Enter to keep the current selection[+], or type selection number: 1 //enter
sudo su - sonar
//paste this
echo 'export JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto' >> ~/.bashrc
echo 'export PATH=$JAVA_HOME/bin:$PATH' >> ~/.bashrc
source ~/.bashrc

java -version

#If everything's fine, you'll see:
openjdk version "17.0.17" 2025-10-21 LTS
OpenJDK Runtime Environment Corretto-17.0.17.10.1 (build 17.0.17+10-LTS)
OpenJDK 64-Bit Server VM Corretto-17.0.17.10.1 (build 17.0.17+10-LTS, mixed

cd /opt/
ll

drwxr-xr-x.  4 root  root         33 Oct 23 00:08 aws
drwxr-xr-x. 11 sonar sonar       172 Mar 31  2023 sonarqube-10.0.0.68432
-rw-r--r--.  1 root  root  352909963 Mar 31  2023 sonarqube-10.0.0.68432.zi

cd sonarqube-10.0.0.68432/bin/linux-x86-64/
ll
```

```
./sonar.sh start

sudo su -
sudo chown -R sonar:sonar /opt/sonarqube-10.0.0.68432
ls -ld /opt/sonarqube-10.0.0.68432
#If everything's fine, you'll see:

drwxr-xr-x. 11 sonar sonar 172 Mar 31  2023 /opt/sonarqube-10.0.0.68432
su - sonar
cd /opt/sonarqube-10.0.0.68432/bin/linux-x86-64
./sonar.sh start
./sonar.sh status

cd /opt/sonarqube-10.0.0.68432/bin/linux-x86-64/
ll
sonar.sh
./sonar.sh restart
/usr/bin/java
Gracefully stopping SonarQube...
SonarQube was not running.
Starting SonarQube...
Started SonarQube.
./sonar.sh status
/usr/bin/java
SonarQube is running (PID).
```

```
And you can access it at:
👉 http://<your-EC2-public-IP>:9000
Default username and password -> admin / admin
Generate token
Go to My Account → Security
Under Generate Tokens:
Name: jenkins-sonar
Type: User Token (or "Project Analysis Token" if available)
Click On Generate
Copy the token
```

2.4. Nexus Repository Manager: The Artifacts Librarian

A CI/CD pipeline produces build artifacts—the compiled, packaged, and versioned application. Nexus Repository Manager (or a similar tool like Artifactory) is a dedicated repository for storing and managing these artifacts.

- **Centralized Storage:** It provides a single, reliable source of truth for all build artifacts. Instead of deploying a file from a temporary Jenkins workspace, the pipeline deploys a specific, versioned artifact from Nexus.
- **Traceability and Rollbacks:** Every artifact (e.g., `ecommerce-site-v1.2.5.war`) is immutable. This creates an auditable history of all releases and makes rolling back to a previous stable version a trivial and reliable operation.
- **Dependency Proxying:** Nexus can proxy and cache public repositories like Maven Central. This speeds up builds (dependencies are downloaded from the local network) and insulates the build process from public repository outages.

```
INSTALLATION COMMANDS:

sudo yum install -y java-1.8.0
cd /opt
wget https://download.sonatype.com/nexus/3/nexus-3.85.0-03-linux-x86_64.tar
tar -xvf nexus-3.85.0-03-linux-x86_64.tar.gz
ll

sudo mv nexus-3.85.0-03 nexus
sudo adduser nexus
sudo passwd nexus
sudo chown -R nexus:nexus nexus
sudo chown -R nexus:nexus sonatype-work

sudo vi nexus/bin/nexus
modify this line
run_as_user="nexus"
sudo -u nexus ./nexus/bin/nexus start
```

```
And you can access it at:
👉 http://<your-EC2-public-IP>:8081
```

Create Repositories

1. Open the Repositories Page
2. In the left sidebar, click “Administration” → “Repository” →“Repositories”
3. Click the “Create repository” button (top-right)

A. Create Maven Release Repository

1. Click “Create repository”

2. Choose maven2 (hosted)

3. Enter details:

a. Name: `maven-release`

b. Version Policy: `Release`

c. Layout Policy: `Strict`

4. Keep “Blob store” as `default`

5. Scroll down → click “Create repository”

B. Create Maven Snapshot Repository

1. Again click “Create repository”
2. Choose maven2 (hosted)
3. Enter:
   - Name: `maven-snapshots`
   - Version Policy: `Snapshot`
   - Layout Policy: `Strict`

4. Click Create repository

```
#Edit pom.xml in your code
#For enquiries please checkout my github account

<distributionManagement>
  <repository>
    <id>nexus</id> <!-- must match Jenkins credentialsId -->
    <url>http://98.80.122.184:8081/repository/maven-releases/</url>
  </repository>
  <snapshotRepository>
    <id>nexus</id>
    <url>http://98.80.122.184:8081/repository/maven-snapshots/</url>
  </snapshotRepository>
</distributionManagement>
```

2.5. Apache Tomcat: The Production Stage

Apache Tomcat is a widely used open-source implementation of the Java Servlet, JavaServer Pages, and WebSocket technologies. For our e-commerce project, it serves as the web server and servlet container that runs the final application.

- **Application Host:** Its primary role is to host the Java web application packaged in the `.war` file.
- **Execution Environment:** It manages the lifecycle of servlets, handles HTTP requests from users browsing the e-commerce site, and routes them to the appropriate application logic.
- **Remote Deployment:** Tomcat includes a Manager App that allows for remote deployment of web applications. Jenkins plugins leverage this capability to automate the deployment of the `.war` file from Nexus directly onto the Tomcat server.

```
Installation commands:

sudo yum update -y
sudo yum install java-17 -y
wget https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.111/bin/apache-tomcat-9.
tar -xvzf apache-tomcat-9.0.111.tar.gz
ll
mv apache-tomcat-9.0.111.tar.gz tomcat

cd /opt/tomcat/bin
sudo chmod +x *.sh
```

```
sudo ./startup.sh

http://<your-ec2-public-ip>:8080
cd /opt/tomcat/webapps/manager/META-INF/
vim context.xml
************comment this two lines***********
    <!--
    <Valve className="org.apache.catalina.valves.RemoteAddrValve"
           allow="127\.\d+\.\d+\.\d+|::1" />
    -->
Do the same for:
/opt/tomcat/webapps/host-manager/META-INF/context.xml

cd /opt/tomcat/conf/
vim tomcat-users.xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="tomcat" password="tomcat" roles="manager-gui,
  manager-script, manager-status"/>
</tomcat-users>

vim servers.xml
connector port 8081

cd bin/
sudo ./shutdown.sh
sudo ./startup.sh

http://<your-ec2-public-ip>:8080/manager/html
Username: tomcat
Password: tomcat
```

3. From Commit to Production: A Step-by-Step Walkthrough

Let's trace the journey of a new feature—for instance, a "Dynamic Pricing Module"—from a developer's machine to the live e-commerce website.

Step	Action	Tools Involved	Detailed Description & Rationale
1. Code Push	Developer commits and pushes code to a Git feature branch.	Git, Developer IDE	The developer finalizes the code for the new module. A `git push` command sends the changes to the central Git server (e.g., GitHub, GitLab). This is the event that triggers the entire automated pipeline.
2. Pipeline Trigger	Jenkins detects the new commit via a webhook.	Jenkins, Git	The Git server sends a webhook notification to Jenkins. Jenkins identifies the corresponding pipeline job, checks out the latest source code into a clean workspace on a build agent, and starts executing the stages defined in the `Jenkinsfile`.
3. Code Analysis	Jenkins executes the SonarQube analysis stage.	Jenkins, Maven, SonarQube	Jenkins runs the command `mvn sonar:sonar`. The Maven SonarQube scanner plugin analyzes the code, collects metrics, and sends the report to the SonarQube server. The pipeline then pauses and queries SonarQube for the Quality Gate status associated with this analysis.
4. Quality Gate Check	Jenkins evaluates the SonarQube Quality Gate result.	Jenkins, SonarQube	**If the gate passes** (e.g., no new critical issues, sufficient test coverage), the pipeline proceeds to the next stage. **If the gate fails**, the pipeline is aborted. Jenkins marks the build as 'FAILED' and sends a notification to the developer, who must fix the issues before trying again. This prevents technical debt from accumulating.

Step	Action	Tools Involved	Detailed Description & Rationale
5. Build & Test	Jenkins executes the build and packaging stage.	Jenkins, Maven	With quality assured, Jenkins runs `mvn clean package`. This command cleans previous build outputs, compiles the source code, runs all unit tests (a second layer of validation), and finally packages the application into a versioned WAR file, such as `ecommerce-site-1.3.0.war`.
6. Artifact Archiving	Jenkins uploads the WAR file to Nexus.	Jenkins, Nexus	Using a Nexus plugin, Jenkins pushes the newly created `ecommerce-site-1.3.0.war` to a "releases" repository in Nexus. This artifact is now the immutable, "golden" version of this build, ready for deployment and permanently archived for auditing and potential rollbacks.
7. Production Deployment	Jenkins deploys the artifact from Nexus to Tomcat.	Jenkins, Nexus, Tomcat	The final stage begins. Jenkins connects to the Tomcat server. Crucially, it instructs the deployment plugin to pull `ecommerce-site-1.3.0.war` **directly from Nexus**, not from its own workspace. The artifact is deployed to Tomcat's `webapps` directory. Tomcat automatically unpacks and starts the new version of the application. The "Dynamic Pricing Module" is now live.

4. Advanced Considerations and Best Practices

Building the pipeline is only the first step. Optimizing and securing it ensures long-term stability and efficiency.

4.1. Pipeline as Code: The `Jenkinsfile`

Hard-coding pipeline logic in the Jenkins UI (Freestyle projects) is brittle and not scalable. The modern approach is to define the pipeline in a `Jenkinsfile`.

Benefits:

- **Versioning:** The pipeline definition lives in source control, just like your application code. Changes are tracked and auditable.
- **Collaboration:** Teams can collaborate on the pipeline definition through pull requests.
- **Disaster Recovery:** If the Jenkins server fails, the pipeline can be instantly recreated from the `Jenkinsfile` in source control.

```
pipeline {
    agent any

    tools {
        jdk 'jdk-8'
        maven 'Maven'
    }

    stages {
        stage('Clone SCM') {
            steps {
                echo 'Cloning SCM repository'
                git branch: 'main', url: 'https://github.com/kadivetikavya/mind
            }
        }

        stage('SonarQube Analysis') {
            steps {
                echo 'Scanning project with SonarQube...'
                sh '''
                    mvn clean verify sonar:sonar \
                    -Dsonar.projectKey=myproject \
                    -Dsonar.host.url=http://98.80.122.184:9000/ \
                    -Dsonar.login=squ_70101041208a736ad4c60d07adca4cb4327c3d28
                '''
```

```
        }
    }

    stage('Build Artifact') {
        steps {
            echo ' Building artifact with Maven'
            sh 'mvn clean install'
        }
    }

    stage('Publish to Nexus') {
        steps {
            echo 'Deploying artifact to Nexus...'
            withCredentials([usernamePassword(credentialsId: 'nexus', usern
                sh """
                    mvn -s /var/lib/jenkins/.m2/settings.xml clean deploy
                """
            }
        }
    }

    stage('Deploy to Tomcat') {
        steps {
            echo 'Deploying WAR to Tomcat webserver'
            deploy adapters: [
                tomcat9(
                    credentialsId: 'tomcat',
                    path: '',
                    url: 'http://98.80.122.184:8091/'
                )
            ],
            contextPath: 'project-2',
            war: '**/*.war'
        }
    }
}

post {
    success {
        echo '✅ Deployment successful!'
    }
    failure {
```

```
            echo '❌ Deployment failed — check console logs.'
        }
    }
}
```

4.2. Security Throughout the Pipeline (DevSecOps)

Security must be integrated into every stage, not treated as an afterthought.

- **Credential Management:** Never hard-code passwords or API keys. Use the Jenkins Credentials Manager to securely store and inject secrets (like Nexus and Tomcat credentials) into the pipeline.
- **SAST and DAST:** SonarQube provides Static Application Security Testing (SAST). Consider adding Dynamic Application Security Testing (DAST) tools (like OWASP ZAP) in a staging environment to scan the running application for vulnerabilities.
- **Dependency Scanning:** Use tools like OWASP Dependency-Check or Nexus Lifecycle to scan project dependencies for known vulnerabilities (CVEs). This can be another stage in your pipeline.

4.3. Environment Separation and Promotion

Direct deployment to production is risky. A mature pipeline includes multiple environments.

- **Development -> Staging -> Production:** The pipeline should first deploy to a 'staging' environment that mirrors production.
- **Automated Testing in Staging:** After deploying to staging, run a suite of automated integration tests, performance tests, and user acceptance tests (UAT).
- **Manual Gate for Production:** The final deployment to production can be gated by a manual approval step in Jenkins (e.g., `input 'Deploy to Production?'`), giving a release manager final say.

5. Conclusion: From Automation to Business Value

By integrating Jenkins, SonarQube, Maven, Nexus, and Tomcat, we have constructed more than just an automated workflow; we have built a strategic asset. This CI/CD pipeline transforms the software development lifecycle for an e-commerce platform from a

high-risk, manual process into a predictable, efficient, and quality-driven engine for innovation.

The immediate benefits are clear: faster delivery of features, higher code quality, enhanced security, and greater stability. Developers receive instant feedback, operations gain reliable and repeatable deployments, and the business can respond to market demands with unprecedented agility. This robust foundation enables teams to focus on what truly matters: building features that delight customers and drive growth.