Creating Your First CI Pipeline: Full Architectural Design and Execution Flow

1. Introduction to Continuous Integration

In modern software development, Continuous Integration (CI) is a fundamental practice that enables teams to deliver high-quality code more frequently and efficiently. Continuous Integration involves automatically building, testing, analyzing, and packaging code every time a change is committed to a shared repository. The primary objective is to detect integration issues early, ensure that code quality remains high, and reduce manual intervention in the software delivery lifecycle.

In this document, we will design and implement a fully functional Continuous Integration pipeline using industry-standard tools hosted in Amazon Web Services (AWS). The chosen tools for this pipeline include:

- **AWS EC2 Instances**
- **Jenkins**
- **Git**
- **Maven**
- **SonarQube**
- **Checkstyle**
- **Nexus**
- **Slack/Email**

By the end of this document, you will have a comprehensive understanding of how to set up, integrate, and automate these tools to achieve an efficient Continuous Integration pipeline that can scale as your team and projects grow.

2. High-Level Architectural Design

At a high level, the Continuous Integration pipeline will follow a series of stages, each designed to perform specific tasks that contribute to ensuring that code is functional, secure, and high-quality before it progresses to the next stage. Below is an overview of the architecture and flow:

- **AWS Infrastructure:** All components will be hosted on separate Amazon EC2 instances.
- **Developer Workflow:** Developers push code changes to Git repositories.
- **Jenkins Automation:** Jenkins polls the Git repository for changes or listens for webhooks.
- **Code Checkout:** Jenkins checks out the latest code.
- **Build and Unit Tests:** Maven is used to build the Java project and run unit tests.
- **Code Quality Analysis:** Checkstyle and SonarQube analyze the code for issues, quality metrics, and code smells.

- **Quality Gates:** SonarQube enforces code quality gates that determine whether the build can proceed.
- **Artifact Storage:** If successful, Maven uploads the built artifact to Nexus Repository.
- **Notifications:** Slack and Email notifications are sent at various stages to inform the team of successes or failures.
- **Continuous Feedback Loop:** The system is fully automated, providing real-time feedback to developers.

3. Detailed Execution Flow

3.1 Code Commit

The CI process is triggered by developers committing code changes to the central Git repository. This can either be GitHub, GitLab, Bitbucket, or a private Git server hosted within AWS.

Every time a commit is pushed:

- Jenkins detects the change (either via polling or webhook).
- A new build job is automatically triggered.

3.2 Build Process

3.2.1 Code Checkout

- Jenkins fetches the latest version of the code from the Git repository.
- The workspace is cleaned to avoid leftover files from previous builds, ensuring a fresh start.

3.2.2 Maven Build and Unit Testing

- Jenkins invokes Maven to build the Java application.
- Maven downloads all required dependencies.
- The application is compiled, and unit tests are executed.
- The results of the unit tests are collected.
- If any test fails, Jenkins halts the pipeline and notifies the team via Slack and/or email.

3.2.3 Initial Notification

- Jenkins sends a build status notification indicating whether the build and unit tests passed or failed.

3.3 Code Analysis Stage

3.3.1 Static Code Analysis with Checkstyle

- Checkstyle analyzes the code for formatting issues, naming conventions, and coding standards violations.
- Results are aggregated and formatted for reporting.

3.3.2 SonarQube Code Quality Gate

- Jenkins sends the compiled code and Checkstyle reports to SonarQube for deeper analysis.
- SonarQube scans for code smells, bugs, vulnerabilities, duplication, and maintainability issues.
- SonarQube applies Quality Gates, which are pre-defined thresholds for acceptable code quality.
- If the Quality Gate is failed, Jenkins halts the pipeline and notifies the team.
- If the Quality Gate passes, the pipeline proceeds to the next stage.

3.4 Artifact Packaging and Storage

3.4.1 Maven Packaging

- Maven packages the application into a deployable artifact, typically a JAR or WAR file for Java applications.

3.4.2 Nexus Artifact Repository

- Jenkins uploads the artifact to Nexus Repository.
- Nexus serves as a centralized artifact repository for versioned builds, which can later be promoted to staging or production environments.
- If the upload fails, Jenkins sends failure notifications.

3.5 Notifications and Continuous Feedback

At every major stage in the pipeline:

- Build success/failure
- Unit test success/failure
- Quality gate pass/fail
- Artifact upload success/failure

Notifications are sent automatically to:

- Slack channels for real-time collaboration.
- Email addresses for asynchronous updates.

This ensures that all team members are continuously aware of the current state of the codebase and pipeline execution.

4. Why This Architecture?

This architecture is designed to meet several core objectives:

- **Automation:** Minimize human intervention.
- **Speed:** Quickly detect and address issues.

- **Quality Assurance:** Enforce code quality standards before deployment.
- **Traceability:** Maintain records of builds and artifacts.
- **Scalability:** Easily expand as more projects or teams join.

5. Step-by-Step System Setup on AWS

5.1 AWS Account Preparation

- Log in to your AWS Account.
- Generate AWS Key Pairs to SSH into EC2 instances using .pem format for Bash login
- Create dedicated Security Groups for (You can decide to use a default VPC or create your own)
    - Jenkins
        - Allow Inbound traffic for port 8080, 22 setting the source as your own IP
        - Allow inbound traffic from port 80 for sonarQube quality gates report
    - Nexus
        - Allow Inbound traffic for port 8081, 22 setting the source as your own IP
        - Allow Inbound traffic for port 8080 for Jenkins server to upload artifacts source being Jenkins SG
    - SonarQube
        - Allow Inbound traffic for port 80, as it will be accessed via Nginx
        - Allow Inbound traffic for port 8080 for Jenkins server to upload reports source being Jenkins SG

5.2 Provision EC2 Instances

Launch EC2 instances for with the following userdata scripts.

Use appropriate instance types based on load (`t3.medium` or higher for SonarQube, t2.small for jenkins and t3.medium for nexus).

- Jenkins Serve
    - #!/bin/bash
    - sudo apt update
    - sudo apt install openjdk-11-jdk -y
    - sudo apt install maven -y
    - curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
    - /usr/share/keyrings/jenkins-keyring.asc > /dev/null

- 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
- sudo apt-get update
- sudo apt-get install jenkins -y
- ###

Nexus Repository

```
#!/bin/bash
yum install java-1.8.0-openjdk.x86_64 wget -y
mkdir -p /opt/nexus/
mkdir -p /tmp/nexus/
cd /tmp/nexus/
NEXUSURL="https://download.sonatype.com/nexus/3/latest-unix.tar.gz"
wget $NEXUSURL -O nexus.tar.gz
sleep 10
EXTOUT=`tar xzvf nexus.tar.gz`
NEXUSDIR=`echo $EXTOUT | cut -d '/' -f1`
sleep 5
rm -rf /tmp/nexus/nexus.tar.gz
cp -r /tmp/nexus/* /opt/nexus/
sleep 5
useradd nexus
chown -R nexus.nexus /opt/nexus
cat <<EOT>> /etc/systemd/system/nexus.service
[Unit]
Description=nexus service
After=network.target
[Service]
Type=forking
LimitNOFILE=65536
ExecStart=/opt/nexus/$NEXUSDIR/bin/nexus start
```

```
ExecStop=/opt/nexus/$NEXUSDIR/bin/nexus stop
User=nexus
Restart=on-abort
[Install]
WantedBy=multi-user.target
EOT
echo 'run_as_user="nexus"' > /opt/nexus/$NEXUSDIR/bin/nexus.rc
systemctl daemon-reload
systemctl start nexus
```

SonarQube Server

```
#!/bin/bash
cp /etc/sysctl.conf /root/sysctl.conf_backup
cat <<EOT> /etc/sysctl.conf
vm.max_map_count=262144
fs.file-max=65536
ulimit -n 65536
ulimit -u 4096
EOT
cp /etc/security/limits.conf /root/sec_limit.conf_backup
cat <<EOT> /etc/security/limits.conf
sonarqube   -   nofile   65536
sonarqube   -   nproc    409
EOT

sudo apt-get update -y
sudo apt-get install openjdk-11-jdk -y
sudo update-alternatives --config java
java -version
sudo apt update
wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - | sudo apt-key add -
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'
```

```
sudo apt install postgresql postgresql-contrib -y
#sudo -u postgres psql -c "SELECT version();"
sudo systemctl enable postgresql.service
sudo systemctl start  postgresql.service
sudo echo "postgres:admin123" | chpasswd
runuser -l postgres -c "createuser sonar"
sudo -i -u postgres psql -c "ALTER USER sonar WITH ENCRYPTED PASSWORD 'admin123';"
sudo -i -u postgres psql -c "CREATE DATABASE sonarqube OWNER sonar;"
sudo -i -u postgres psql -c "GRANT ALL PRIVILEGES ON DATABASE sonarqube to sonar;"
systemctl restart  postgresql
#systemctl status -l   postgresql
netstat -tulpena | grep postgres
sudo mkdir -p /sonarqube/
cd /sonarqube/
sudo curl -O https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-8.3.0.34182.zip
sudo apt-get install zip -y
sudo unzip -o sonarqube-8.3.0.34182.zip -d /opt/
sudo mv /opt/sonarqube-8.3.0.34182/ /opt/sonarqube
sudo groupadd sonar
sudo useradd -c "SonarQube - User" -d /opt/sonarqube/ -g sonar sonar
sudo chown sonar:sonar /opt/sonarqube/ -R
cp /opt/sonarqube/conf/sonar.properties /root/sonar.properties_backup
cat <<EOT> /opt/sonarqube/conf/sonar.properties
sonar.jdbc.username=sonar
sonar.jdbc.password=admin123
sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube
sonar.web.host=0.0.0.0
sonar.web.port=9000
sonar.web.javaAdditionalOpts=-server
sonar.search.javaOpts=-Xmx512m -Xms512m -XX:+HeapDumpOnOutOfMemoryError
sonar.log.level=INFO
sonar.path.logs=logs
```

```
EOT
cat <<EOT> /etc/systemd/system/sonarqube.service
[Unit]
Description=SonarQube service
After=syslog.target network.target
[Service]
Type=forking
ExecStart=/opt/sonarqube/bin/linux-x86-64/sonar.sh start
ExecStop=/opt/sonarqube/bin/linux-x86-64/sonar.sh stop
User=sonar
Group=sonar
Restart=always

LimitNOFILE=65536
LimitNPROC=4096
[Install]
WantedBy=multi-user.target
EOT
systemctl daemon-reload
systemctl enable sonarqube.service
#systemctl start sonarqube.service
#systemctl status -l sonarqube.service
apt-get install nginx -y
rm -rf /etc/nginx/sites-enabled/default
rm -rf /etc/nginx/sites-available/default
cat <<EOT> /etc/nginx/sites-available/sonarqube
server{
    listen      80;
    server_name sonarqube.groophy.in;
    access_log  /var/log/nginx/sonar.access.log;
    error_log   /var/log/nginx/sonar.error.log;
    proxy_buffers 16 64k;
```

```
    proxy_buffer_size 128k;
    location / {
        proxy_pass  http://127.0.0.1:9000;
        proxy_next_upstream error timeout invalid_header http_500 http_502 http_503 http_504;
        proxy_redirect off;

        proxy_set_header    Host            \$host;
        proxy_set_header    X-Real-IP       \$remote_addr;
        proxy_set_header    X-Forwarded-For \$proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Proto http;
    }
}
EOT
ln -s /etc/nginx/sites-available/sonarqube /etc/nginx/sites-enabled/sonarqube
systemctl enable nginx.service
#systemctl restart nginx.service
sudo ufw allow 80,9000,9001/tcp
echo "System reboot in 30 sec"
sleep 30
reboot
```

5.3 Jenkins Post Installation

Launch the Jenkins Server by copying the given external IP address and pasting it on your browser.

Once runnig you will be prompted for the server's password. SSH into the server via your the pem file and the IP address.

For Example

```
ssh -i  my_key.pem ubuntu@10.0.0.0
```

Become the root user by:

```
sudo -i
```

Then cat the path given in the login page to get the password.

Install suggested plugins Initially and any additional plugin will be installed later.

Create a User and Save.

Before building any job setup Nexus repository cause the Jenkins server will need access for the job.

- Install necessary plugins:
    - Git plugin
    - Maven Integration
    - Checkstyle plugin
    - SonarQube Scanner plugin
    - Slack Notification plugin
- Set up Jenkins credentials for:
    - Git access
    - Nexus authentication
    - SonarQube integration

5.4 Nexus Repository Configuration

The Initial launch is almost similar to that of Jenkins, access via external port, get the password by ssh into the nexus server copy and paste the password via the path provided.

The username is admin

- Click Repository in from the left tab
- Create 3 main repositories:
    - maven-hosted for uploading artifacts
    - maven-proxy for storing dependencies and add the url for downlaoding the dependencies
    - maven-group for grouping both the repos created.

5.5 SonarQube Configuration

- Install SonarQube and configure a PostgreSQL database backend.
- Create a new project token for Jenkins to authenticate.
- Set up Quality Gates and coding rules.
- Integrate Checkstyle plugin into SonarQube.

5.6 Jenkins Pipeline Configuration

5.6.1 Build Job

- Create a freestyle initially.(You can create a second job and copy every configuration from the first job).

  Note: By Using freestyle you can create a chain of downstream independent jobs.

- Configure source control repository by giving its URL and the branch which should have your code and POM file
- Give credentials
- In the Build section choose “Invoke top-level maven target” add “install” and “Dskiptest” under Goal. This only builds and skips the test code.
- Under the settings, add the settings file name not the path. Inorder for the system to utilise the set repos in Nexus.
- Make sure to pass the variable names under the properties section.

  SNAP-REPO=

  NEXUS-USER=

  NEXUS-PASS=
- The values for these properties are the ones you set dutring the configuration of Nexus like the name of the maven-host, maven-proxy, maven-snapshot(SNAP-REPO), the user name and password you set during initial launch. The NEXUSIP= is your private IP in AWS for Nexus server.
- The Settings should look something like this:

  <?xml version="1.0" encoding="UTF-8"?>

<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd">

<servers>

<server>

<id>${SNAP-REPO}</id>

<username>${NEXUS-USER}</username>

<password>${NEXUS-PASS}</password>

</server>

<server>

<id>${RELEASE-REPO}</id>

<username>${NEXUS-USER}</username>

<password>${NEXUS-PASS}</password>

</server>

<server>

<id>${CENTRAL-REPO}</id>

```
    <username>${NEXUS-USER}</username>
    <password>${NEXUS-PASS}</password>
   </server>
   <server>
     <id>${NEXUS-GRP-REPO}</id>
    <username>${NEXUS-USER}</username>
    <password>${NEXUS-PASS}</password>
   </server>
  </servers>
  <mirrors>
   <mirror>
    <id>${CENTRAL-REPO}</id>
    <name>${CENTRAL-REPO}</name>
    <url>http://${NEXUSIP}:${NEXUSPORT}/repository/${NEXUS GRP REPO}/</url>
    <mirrorOf>*</mirrorOf>
   </mirror>
  </mirrors>
</settings>
```

Save everything and run build now.

5.6.2 Slack Notification Setup

You should already have an account setup on slack

Login to your account and then launch a workspace and then create new channel.

Visit api.slack.com, click on “Start Building” then click “Create APP”. Choose your workspace

From the left bar, click “Oauth and Permissisons” and scroll down to scopes and select “chat:write”

Scroll up and press “Install APP to workspace”

Copy the provided token.

Go to workspace and down in the text box add @name_of_APP and click “Enter ” then “Invite to channel”

Add the token from Slack to global configurations. Kind being Secret text

Go to Manage Jenkins→ Configure system→ Scroll to Slack if you don’t have make sure you install the plugin. Put a checkmark to “custom slack bot app user”

Go to the Job you built initially and scroll down to Post Actions and choose from the checkmarks what you would want to send out.

Click “Advanced” and add the token, and channel and workspace.

5.6.3 Checkstyle Integration

- Add Checkstyle plugin to Jenkins
- Add violation plugin to check the quality gate
- Configure Jenkins to parse Checkstyle reports and fail builds on violations.

  Note: This method is an alternative to the code analysis which is done inside jenkins
- Rememeber to set the maximim number of warning for which a build is considered a fail

5.6.4 SonarQube Scanner Setup

Launch SonarQube via the external IP,

Access it with default username and opassword as admin

Generate a Token

From the previously installled plugin SonarQube and SonarQube Gateway in Jenkins, Configure SonarQube

Add SonarQube scanner theough Global tool configuration in Jenkins

- Provide SonarQube Server URL and credentials in Jenkins and also the SoanrQube Gateway. The IP address is the private adddress

Since we are running freestyle projects, we also create a job for the codeAnalysis

Under the goal section add “install” Click on advanced just directly below goals and add execute SonarQube Scanners

Add the following as the analysis properties:

```
sonar.projectKey=vprofile

sonar.projectName=vprofile-repo

sonar.projectVersion=1.0

sonar.sources=src/

sonar.java.binaries=target/test-classes/com/visualpathit/account/controllerTest/

sonar.junit.reportsPath=target/surefire-reports/

sonar.jacoco.reportsPath=target/jacoco.exec

sonar.java.checkstyle.reportPaths=target/checkstyle-result.xml
```

5.6.5 Artifact Upload Job

- After successful build and quality checks, configure Nexus Artifact Uploader plugin.
- Since the artifact is already built, we can install copy artifact plugin to use it to upload the war file to nexus.
- Install “Zentimestamp plugin” for versioning your artifact

- Create a freestyle job for the upload and copy the content of the build job while specifying the file name like **/*.war
- Upload final artifacts to Nexus Repository.

6. Conclusion

By following this approach, you will have successfully built a robust, highly automated Continuous Integration pipeline that enforces code quality, facilitates collaboration, and prepares your software for reliable delivery. This system not only improves developer productivity but also ensures that technical debt is minimized, code standards are enforced, and errors are caught early before becoming production issues.

This is the foundation upon which professional DevOps teams build high-performing CI/CD pipelines used across industries today.