JENKINS

THE

COMPLETE

TUTORIAL:

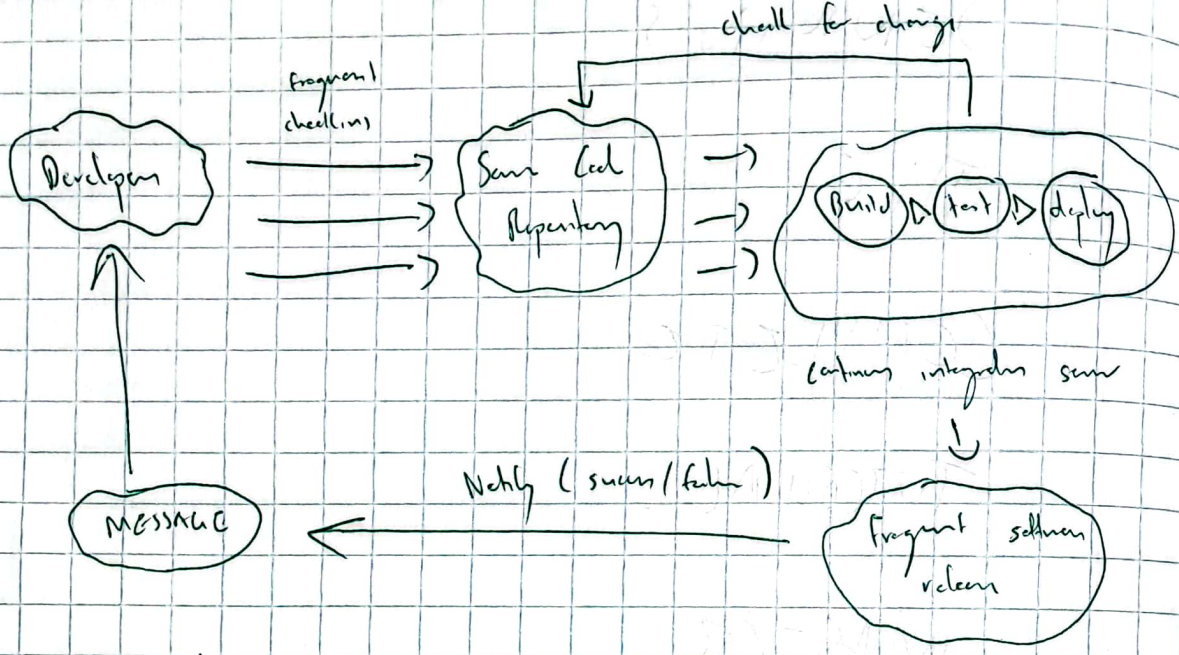
MASTER

CICD AND

DEVOPS

Section 1: Course Introduction

Introduction to continuous integration



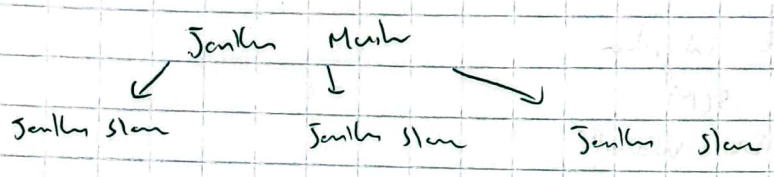
Continuous Integration: practice of integrating changes from different developers in the team into a mainline as early as possible. Each integration can then be verified by an automated build and automated tests.

Continuous Delivery: practice of keeping your codebase deployable at any point.

Continuous Deployment: keeping your application deployable at any point or even automatically releasing to a test or production environment.

Introduction to Jenkins

Jenkins Distributed Architecture :



Jenkins master : scheduling and dispatching jobs to slaves, monitor the slaves, record and present job results, execute some jobs directly

Jenkins slave : hearth all request from the Jenkins master instance. Each slave is an independent entity running on its own OS

Getting started with Jenkins

A few Jenkins terms :

- job/project : runnable task that configured in Jenkins
- node : each machine that is part of jenkins env
- executer : thread or slot for execution of jobs
- build : result after executing a job
- plugin : software that extend, core functionality of Jenkins

Section 11: Continuous Integration

Jenkins integration with Git using poll SCM

2 ways of integration:
- Poll SCM
- GitHub Webhook

Poll SCM vs Webhook

Poll SCM:
- Resource intensive
- Wait for build
- Safer option

Webhook
- Not costly
- Instantaneous Build
- Security concern

SonarQube

- Open source code analysis tool
- detailed code analysis report

Code Analysis with SonarQube

- add SonarQube dependency to pom.xml
- invoke sonar:sonar or lifecycle: clean test sonar:sonar
- use Jacoco plugin for code coverage

Integration of SonarQube with Jenkins

- In job -> configure -> Build : Add sonar: sonar after test

OR

- Add post build action -> select : SonarQube analysis with Maven

OR

- In job -> Build Environment -> Prepare QubeScanner environment
- In job -> Build -> Execute SonarQube Scanner
  ↓
  Task to run : scan
  ↓
  Analysis properties
  (+) Project Metadata
  sonar.projectKey :
  sonar.projectName :
  sonar.projectVersion :
  sonar.ProjectBaseDir :
  sonar.sources :
  sonar.java.binaries :

Email Notification

~~Build -> Execute Windows batch command~~

Post-Build Action : E-mail Notification

Using TestNG

PostSteps -> Publish TestNG Results

Jenkins Integration with Tomcat

- Post build → Deploy war/ear to container
- Container → Container
  ↳ Credentials : from tomcat config

Section 5 : Pipelines

Setting up Delivery Pipeline

① Build → ② Test → ③ Deploy

Build Trigger → Build after other project build → Pipeline Job

Setup pipeline view :

→ New view
→ Pipeline → Components - Add
→ Initial → Build
→ Final →
→ Enable start of new pipeline ☑
→ Enable restart ☑
→ Total build time ☑

Setup Build Pipeline

- Save as Delivery pipeline

Jenkins Pipeline

Declarative vs Scripted Pipeline → the 2 ways of coding a Jenkins pipeline

Declarative:
- recent feature in Jenkins and uses a simpler groovy syntax
- code is written locally in a file and is checked into SCM
- code written with a pipeline block

Scripted pipeline:
- traditional way writing Jenkins pipeline with a stricter groovy syntax
- directly on Jenkins instance
- code written with a node block

Pipeline Concepts:

Step: refers to the single task that has to be performed with a stage

Stage: defines a conceptually distinct subset of tasks

Pipeline: user-defined model of a CD pipeline

Node: machine which is part of the Jenkins environment.

Syntax Comparison:

Declarative:

```
pipeline {
    agent any
    stages {
        stage ('Build') {
            steps {
                // steps
            }
        }
        stage ('Test') {
            steps {
```

```
// will contain steps
    }
}
stage('Deploy') {
    steps {
        // steps
    }
}
}
```

Scripted:

```
node {
    stage('Build') {
        // steps
    }
    stage('Test') {
        // steps
    }
    stage('Deploy') {
        // steps
    }
}
```

Setting up scripted pipeline

- new job → Pipeline
- Pipeline → Definition: Pipeline script
- Script → Code script using format above

- Github + Maven

Setting up Declarative Pipeline

- in your git repo, have a file called Jenkinsfile with the code from syntax

- job → Pipeline
- Script → Pipeline Script from SCM
  - SCM → Git
  - Branch → Default → master

eg of user input to execute a stage:

```
stage ("Conditional Build") {
    steps {
        input ("Do you want to capture results?")
        junit '**/target/surefire-reports/TEST-*.xml'
        archive 'target/*.jar'
    }
}
```

- to get more syntax go to 'pipeline-syntax'

- to run steps in parallel

```
stage(—) {
    steps {
        parallel(
            )
    }
}
```