# Continuous Integration using Jenkins

## Project

**Problem Statement:** You are working as a DevOps Engineer in a company named Sanders & Fresco Pvt ltd. You have been asked by your manager to create a Maven Project using Jenkins and build a war file of that project. As a proof of concept, you have been given a web application to build. And once done with building the war file, deploy it over the tomcat server.

**Steps to solve:**

• Open Jenkins and create a Maven project using it.

• You will have to create the following jobs, which are as follow:

o Compile

o Code Review

o Unit test

o Package

o Deploy

You have to complete your job within the given time frame.

**Solution:**

**Based on the project requirements, here's a detailed step-by-step process to create a Maven project in Jenkins, build a WAR file, and deploy it to Tomcat:**

**1. Prerequisites Setup**

Before starting in Jenkins:

* Install Java JDK on Jenkins server
* Install Maven on Jenkins server
* Install Tomcat server (for deployment)
* Ensure Jenkins has these plugins installed:
  + Maven Integration plugin
  + Deploy to container plugin
  + Any SCM plugin (Git/SVN) if using version control

**2. Create a New Maven Project in Jenkins**

1. Log in to Jenkins and click "New Item"
2. Enter a name for your project (e.g., "MavenWebApp")
3. Select "Maven project" and click OK

**3. Configure Source Code Management**

1. In the project configuration:
   * Under "Source Code Management":
     + Select your SCM (Git or Subversion)
     + Enter repository URL
     + Add credentials if needed

**4. Configure Build Triggers (Optional)**

Set up when builds should execute:

* Poll SCM (schedule regular checks)
* Webhooks (trigger on code push)
* Manual build (default)

**5. Create the Required Build Jobs**

**A. Compile Job**

1. In the "Build" section:
   * Add build step: "Invoke top-level Maven targets"
   * Goals: compile
   * This compiles Java source code

**B. Code Review Job**

1. Add another build step or create a separate job:
   * Use tools like SonarQube or Checkstyle
   * Example Maven goal: sonar:sonar
   * Or use dedicated plugins for code analysis

**C. Unit Test Job**

1. Add build step:
   * Goals: test
   * This runs all unit tests in the project

**D. Package Job**

1. Add build step:
   * Goals: package
   * This creates the WAR file in target/ directory

**E. Deploy Job**

1. Add post-build action:
   * Select "Deploy war/ear to a container"
   * Configure:
     + WAR/EAR files: \*\*/\*.war (or exact path)
     + Context path: /yourapp (optional)
     + Containers: Select Tomcat version
     + Enter Tomcat manager credentials:
       - URL: http://tomcat-server:8080/manager/text
       - Username/password (configured in Tomcat's tomcat-users.xml)

**6. Save and Run the Project**

1. Click "Save" to store your configuration
2. Click "Build Now" to execute the entire pipeline

**7. Verify Deployment**

1. Check Tomcat server:
   * Access http://tomcat-server:8080/manager/html
   * Verify your application appears in the list
2. Access your application:
   * Typically at http://tomcat-server:8080/yourapp