1. Play with Jenkins

Getting Started
1. Attempt the test only if your network is stable. Avoid using mobile internet.
2. Run the following command in the terminal once the **jenkins.war** file is visible under **/projects/challenge** to start Jenkins.
   export _JAVA_OPTIONS=-Xmx2048m && /usr/lib/jvm/java-11-openjdk-amd64/bin/java -jar jenkins.war --httpPort=8000
3. Launch Jenkins web URL in a new tab.
Note: If you get an error as "Error: Invalid or corrupt jar file jenkins.war" or "File or directory does not exist" then:

- Remove the existing war file **$rm jenkins.war**
- Download the jenkins war file **$wget https://get.jenkins.io/war-stable/2.375.1/jenkins.war**
- Execute the command **export _JAVA_OPTIONS=-Xmx2048m && /usr/lib/jvm/java-11-openjdk-amd64/bin/java -jar jenkins.war --httpPort=8000** and try launching Jenkins again.

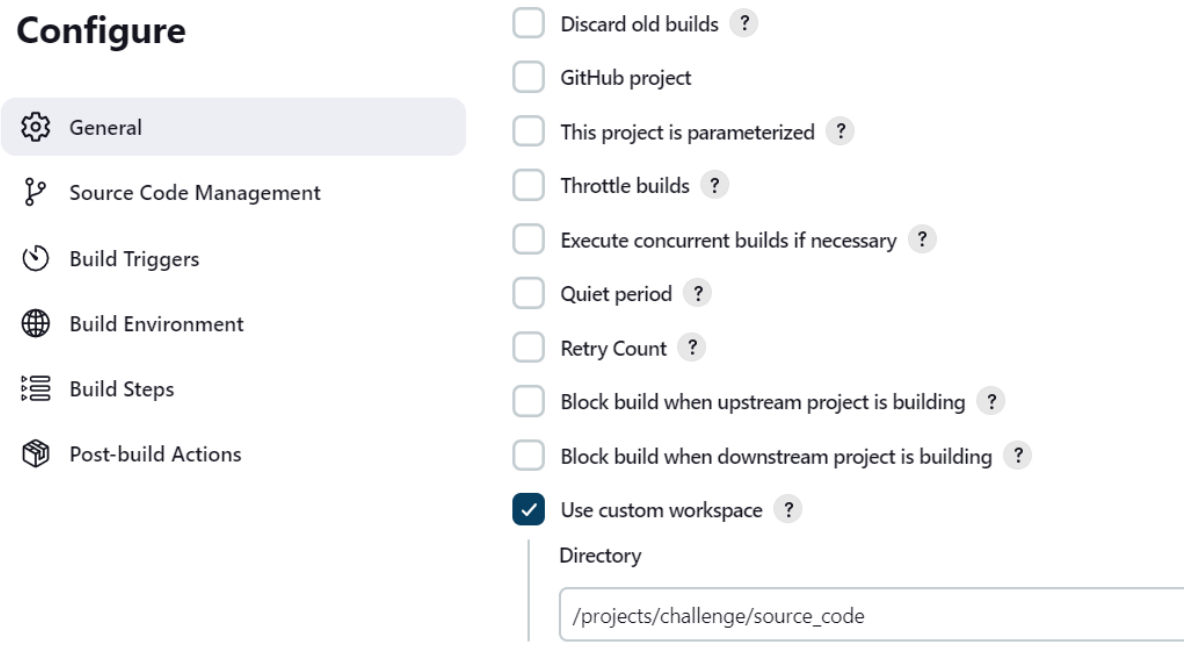
4. Select **Run Tests** to check your result.

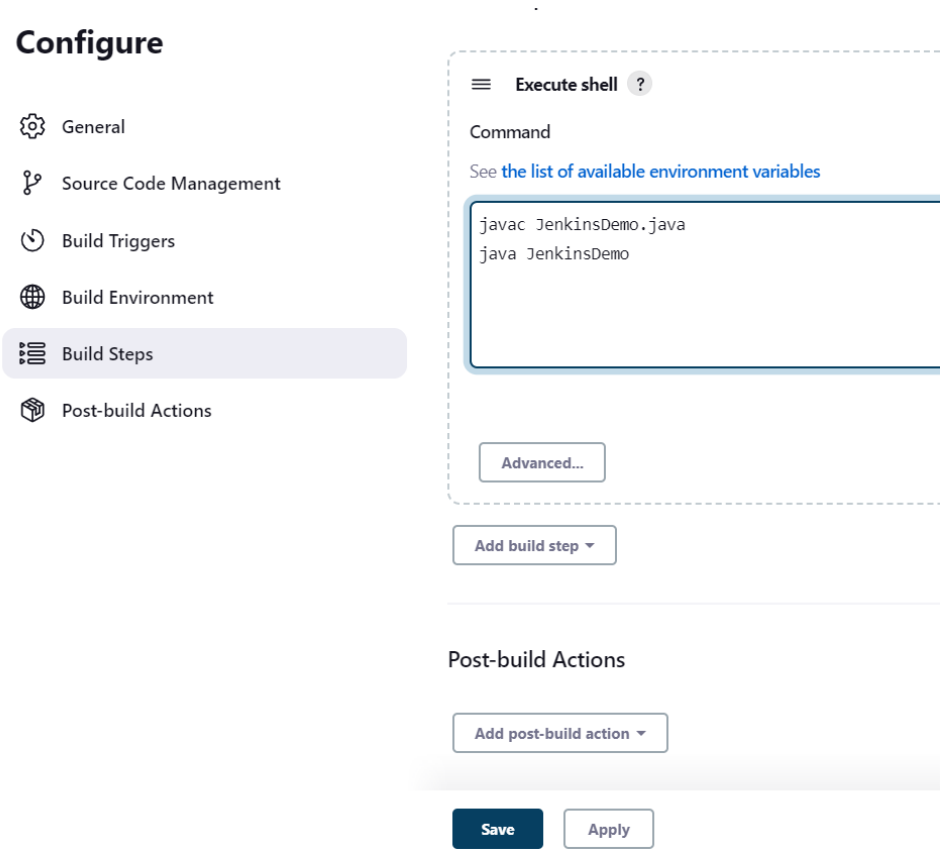
```
user@5aa6c7c4ae09:/projects/challenge$ ls
jenkins.war  logs  PROJECT_FILES_INSTRUCTIONS.md  source_code
user@5aa6c7c4ae09:/projects/challenge$ export _JAVA_OPTIONS=-Xmx1024m
user@5aa6c7c4ae09:/projects/challenge$ export _JAVA_OPTIONS=-Xmx1024m && /usr/lib/jvm/java-11-openjdk-amd64
/bin/java -jar jenkins.war --httpPort=8000
Picked up _JAVA_OPTIONS: -Xmx1024m
Running from: /projects/challenge/jenkins.war
webroot: $user.home/.jenkins
2023-04-20 16:21:00.049+0000 [id=1]     INFO    winstone.Logger#logInternal: Beginning extraction from war
file
2023-04-20 16:21:01.860+0000 [id=1]     WARNING o.e.j.s.handler.ContextHandler#setContextPath: Empty contex
tPath
2023-04-20 16:21:01.987+0000 [id=1]     INFO    org.eclipse.jetty.server.Server#doStart: jetty-10.0.12; bui
lt: 2022-09-14T01:54:40.076Z; git: 408d0139887e27a57b54ed52e2d92a36731a7e88; jvm 11.0.14+9-Ubuntu-0ubuntu2.
16.04
2023-04-20 16:21:02.728+0000 [id=1]     INFO    o.e.j.w.StandardDescriptorProcessor#visitServlet: NO JSP Su
pport for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
2023-04-20 16:21:02.876+0000 [id=1]     INFO    o.e.j.s.s.DefaultSessionIdManager#doStart: Session workerNa
```

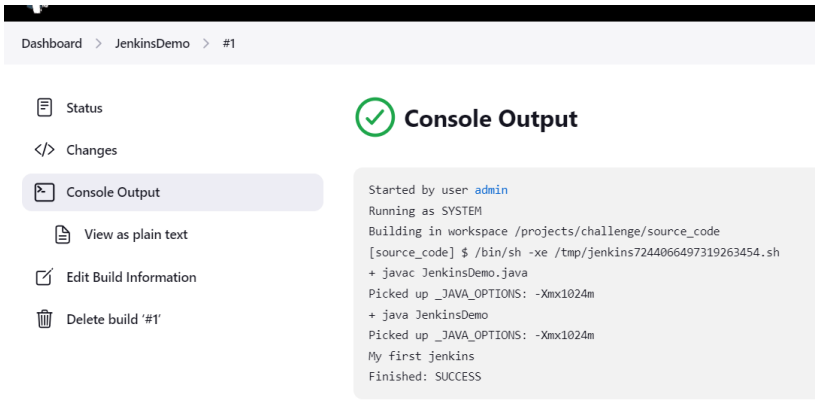
Setup Jenkins .

Configuring Jenkins

- Now, unlock Jenkins using the **Administrator password** in the specified location and continue with install suggested plugins.
- When you find the **Create First Admin User** page, click on **continue as admin** and then click **save and finish**.
- Now, your Jenkins setup is complete and click on **start using Jenkins**.
- Create a new job named **JenkinsDemo**.
- Proceed with the **Freestyle project**.
- A source code directory named **source_code** is given in the path **/projects/challenge/**
- Configure the **custom workspace** to invoke the **source_code** directory .
- Add build step to Execute a shell with **javac JenkinsDemo.java** and **java JenkinsDemo**.
- Now save and start building your job.
- Once done, check your result in the console output of your build. ***Hurray!! you are done if your output is*** **Finished: SUCCESS**




Configure
General
Source Code Management
Build Triggers
Build Environment
Build Steps
Post-build Actions
Execute shell
Command
See the list of available environment variables
javac JenkinsDemo.java
java JenkinsDemo
Advanced...
Add build step
Post-build Actions
Add post-build action
Save
Apply


Dashboard > JenkinsDemo > #1
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete build '#1'
Console Output
Started by user admin
Running as SYSTEM
Building in workspace /projects/challenge/source_code
[source_code] $ /bin/sh -xe /tmp/jenkins7244066497319263454.sh
+ javac JenkinsDemo.java
Picked up _JAVA_OPTIONS: -Xmx1024m
+ java JenkinsDemo
Picked up _JAVA_OPTIONS: -Xmx1024m
My first jenkins
Finished: SUCCESS

1. Generate JUnit Test Report with Maven project

Getting Started
1. Once the **jenkins.war** file is visible under /projects/challenge path, start Jenkins by clicking on **Run -> Run**.
2. Launch Jenkins in a new tab for better visualization by clicking on **Run -> Open preview**.
3. Login to Jenkins with username as **admin** and fetch password from the path $ **cat /home/user/.jenkins/secrets/initialAdminPassword**
4. Click on **Run Tests** to know your result.
5. Once you complete the challenge, click **Submit**.

Note:
If you face "**Error: Invalid or corrupt jar file jenkins.war**" or "**File or directory does not exist**" error, then perform the following:
- Remove the existing Jenkins war file $ **rm jenkins.war**
- Download the jenkins war file $ **wget https://get.jenkins.io/war-stable/2.375.1/jenkins.war**
- Once downloaded, click on **Run -> Run** or Execute the following command $ **export _JAVA_OPTIONS=-Xmx2048m && /usr/lib/jvm/java-11-openjdk-amd64/bin/java -jar jenkins.war --httpPort=8000**
- Launch Jenkins again.

Configuring Jenkins
1. Configure Maven in the Global tool configuration.
2. Create a new job named **MavenDemo**.
3. Proceed with the Freestyle project.
4. A Maven Java project named **mavenproject** is given in the **/projects/challenge** path.
5. Configure the custom workspace to invoke the **mavenproject** directory.
6. Add build step to Invoke top-level Maven targets.
7. Set the Goals to **Clean, Compile,** and **Test**.
8. Save and build your job.
9. Once done, check your result in the console output of your build.

Dashboard > Manage Jenkins > Tools

List of Maven installations on this system

Add Maven

Maven

Name

maven

Required

Install automatically

Install from Apache

Version

3.9.1

Add Installer

Add Maven

Save Apply

Dashboard > MavenDemo > Configuration

Configure

- General
- Source Code Management
- Build Triggers
- Build Environment
- Build Steps
- Post-build Actions

[Plain text] Preview

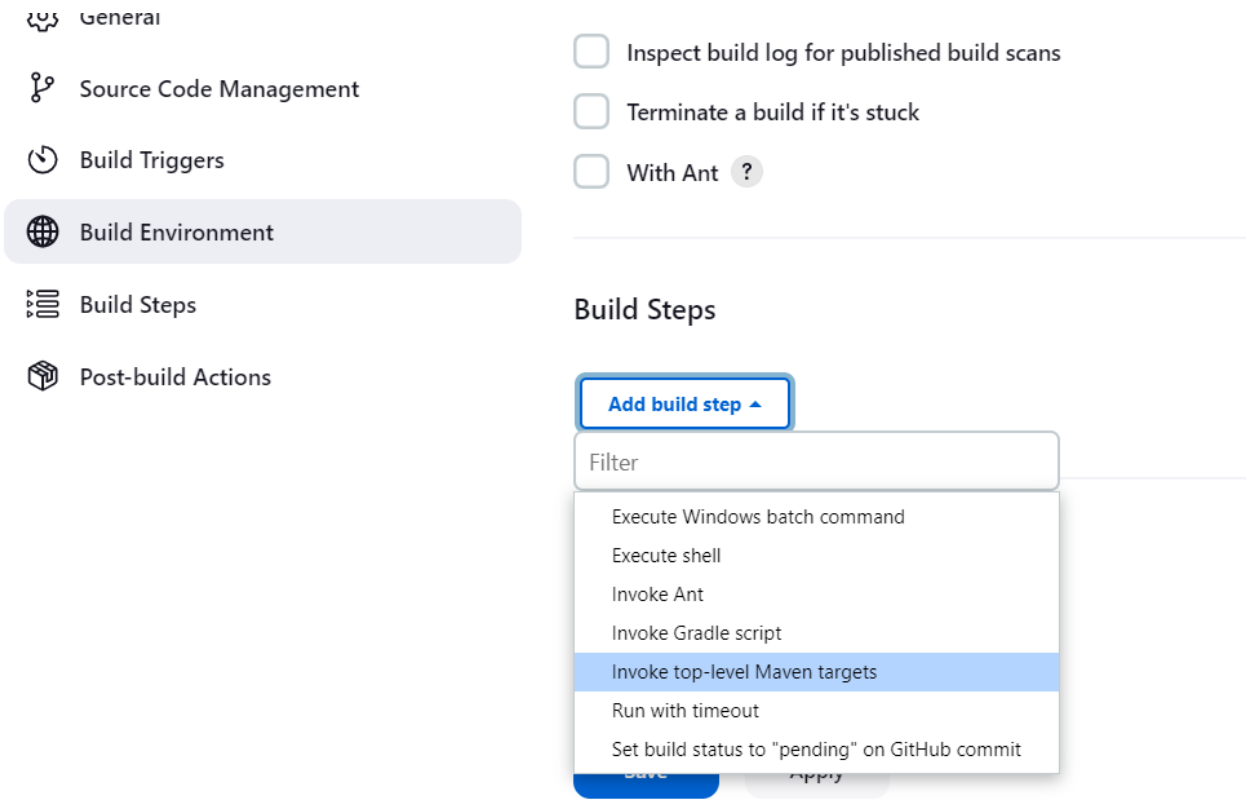
- [ ] Discard old builds
- [ ] GitHub project
- [ ] This project is parameterized
- [ ] Throttle builds
- [ ] Execute concurrent builds if necessary

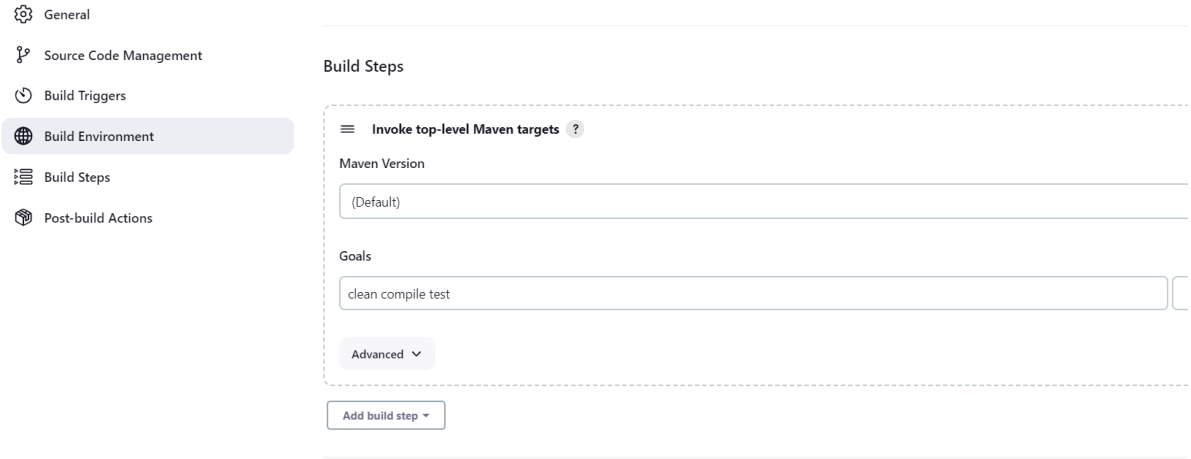
Advanced Edited

- [ ] Quiet period
- [ ] Retry Count
- [ ] Block build when upstream project is building
- [ ] Block build when downstream project is building
- [x] Use custom workspace

Directory

/projects/challenge/mavenproject






```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.maven</groupId>
  <artifactId>mavendemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13-beta-2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

This is an example of a dependency block in a Maven project's pom.xml file. The dependencies block is used to list all the external libraries or modules that the project depends on, along with their versions and scopes.

In this example, the project has a dependency on the JUnit testing framework. The groupId identifies the organization that created the library, the artifactId specifies the name of the library, and the version specifies the version of the library to use. The scope specifies the scope of the dependency, which in this case is test, indicating that the JUnit library is only needed for testing purposes and should not be included in the final build.

When the project is built, Maven will automatically download the JUnit library and its dependencies from a remote repository, and make them available to the project's code. This allows the code to use JUnit for testing without having to manually download and manage the library.

This is an example of a pom.xml file for a Maven project. The pom.xml file is the heart of a Maven project, as it contains all the information required to build and configure the project.

In this example, the project element is the root element of the pom.xml file, and it contains child elements that define various aspects of the project, such as the project's groupId, artifactId, and version.

The modelVersion element specifies the version of the POM schema to use, and the xmlns and xmlns:xsi attributes specify the XML namespace for the POM schema and the XML schema instance namespace, respectively.

The groupId element specifies the unique identifier for the project's group or organization, the artifactId element specifies the unique identifier for the project's artifact or module, and the version element specifies the version of the project.

The dependencies element contains child dependency elements that list the project's external dependencies, in this case a dependency on JUnit. The groupId, artifactId, and version elements within the dependency element specify the details of the dependency, and the scope element specifies the scope of the dependency, in this case, test.

When the project is built, Maven will read the pom.xml file and use the information it contains to download the necessary dependencies, compile the source code, and package the project into a deployable format.