DEVOPS LAB MANUAL

Experiment 1: Introduction to Maven and gradle:

Overview of Build Automation Tools, key differences between Maven and Gradle, Installation and Setup.

Build Automation Tools:

1. Apache Maven

- Config file: `pom.xml` (XML-based)
- Philosophy: Convention over configuration
- Dependency management: Built-in via Maven Central
- Ideal for: Java projects, enterprise applications

Pros:

- Widely used and standardized
- Strong IDE support (e.g., IntelliJ, Eclipse)

Cons:

- XML can be verbose
- Less flexible for custom workflows

2. Gradle

- Config file: build.gradle (Groovy) or build.gradle.kts (Kotlin DSL)
- Philosophy: Flexible, modern build system
- Dependency management: Maven/Ivy compatible
- Ideal for: Java, Kotlin, Android, multi-language builds

Pros:

- Faster builds with daemon and incremental support
- Highly customizable scripting

Cons:

- More complex to learn initially

3. Apache Ant

- Config file: `build.xml` (XML-based)
- Philosophy: Scripted automation
- Dependency management: None built-in (use Ivy)
- Ideal for: Older/legacy projects

Pros:

- Flexible for custom build tasks

Cons:

- Outdated, verbose, manual dependency management

4. Make / GNU Make

- Config file: Makefile
- Philosophy: Rule-based build process
- Used in: C/C++, low-level or cross-platform projects

Pros:

- Simple and powerful
- Works well in Unix environments

Cons:

- Not native to Java
- Not dependency-aware like Gradle/Maven

5. Bazel (by Google)

- Config files: `BUILD`, `WORKSPACE`
- Philosophy: Fast, scalable builds for monorepos
- Used in: Google-scale codebases

Pros:

- Excellent caching, parallelism
- Language-agnostic

Cons:

- Complex to set up
- Less community support than Maven/Gradle

Key differences between Maven and Gradle:

Feature	Maven	Gradle
Language	XML	Groovy/Kotlin DSL
Performance	Slower	Faster (incremental)
Flexibility	Limited	High
Learning Curve	Easier for beginners	Steeper for beginners
Android Support	Limited	First-class

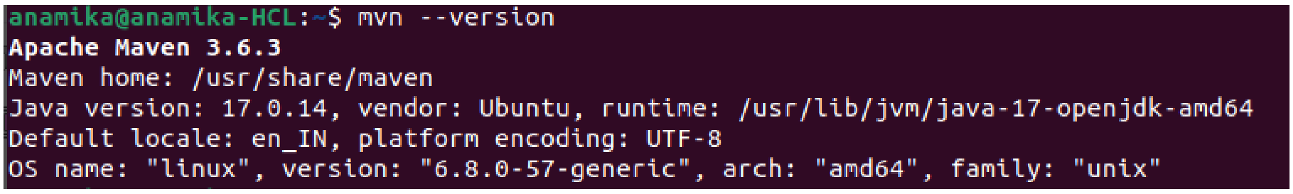
Maven Installation:

(Note: better to use java 17 version)
Update java:
sudo apt update
sudo apt install openjdk-17-jdk
java –version

sudo apt update
sudo apt install maven -y
mvn --version

Note: if need latest version
wget https://downloads.apache.org/maven/maven-3/3.9.6/binaries/apache-maven-3.9.6-bin.tar.gz

Result:



```
anamika@anamika-HCL:~$ mvn --version
Apache Maven 3.6.3
Maven home: /usr/share/maven
Java version: 17.0.14, vendor: Ubuntu, runtime: /usr/lib/jvm/java-17-openjdk-amd64
Default locale: en_IN, platform encoding: UTF-8
OS name: "linux", version: "6.8.0-57-generic", arch: "amd64", family: "unix"
```

Gradle installation:
Ensure your system has Java Development Kit (JDK) installed. Gradle requires Java 8 or later.
Check if Java is installed:

(Note: better to use java 17 version)
Update java:
sudo apt update
sudo apt install openjdk-17-jdk
java --version

If not installed, install OpenJDK:
sudo apt update && sudo apt install openjdk-11-jdk -y

Gradle installation steps:
option 1:
sudo apt update && sudo apt install gradle -y

(Note: Better to prefer option 2)
option 2:
Download the latest Gradle binary:
wget https://services.gradle.org/distributions/gradle-8.6-bin.zip

Extract and move it to **/opt/gradle/**:

sudo mkdir /opt/gradle

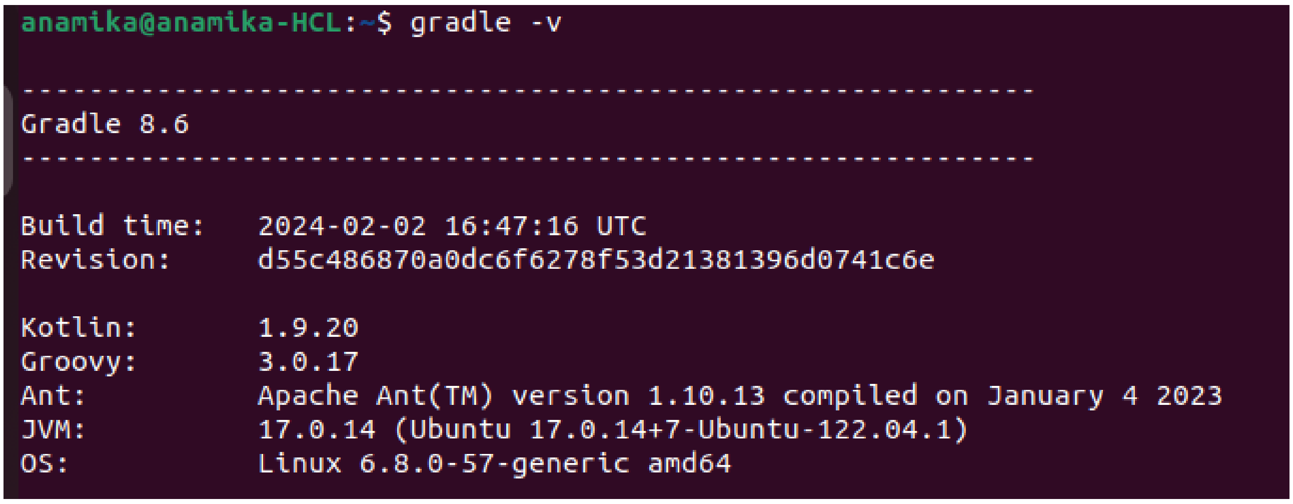
sudo unzip gradle-8.6-bin.zip -d /opt/gradle

sudo ln -s /opt/gradle/gradle-8.6 /opt/gradle/latest

Set up environment variables:
echo "export PATH=/opt/gradle/latest/bin:\$PATH" | sudo tee /etc/profile.d/gradle.sh
source /etc/profile.d/gradle.sh

Verify:
gradle -v

Result:



```
anamika@anamika-HCL:~$ gradle -v

------------------------------------------------------------
Gradle 8.6
------------------------------------------------------------

Build time:   2024-02-02 16:47:16 UTC
Revision:     d55c486870a0dc6f6278f53d21381396d0741c6e

Kotlin:       1.9.20
Groovy:       3.0.17
Ant:          Apache Ant(TM) version 1.10.13 compiled on January 4 2023
JVM:          17.0.14 (Ubuntu 17.0.14+7-Ubuntu-122.04.1)
OS:           Linux 6.8.0-57-generic amd64
```

Experiment 2: Working with Maven:
Creating a Maven Project, Understanding the POM file, Dependency management and plugins.

Create a New Maven Project:

Use the following command to generate a new Maven project:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

navigate to project directory;
```
cd my-app
```

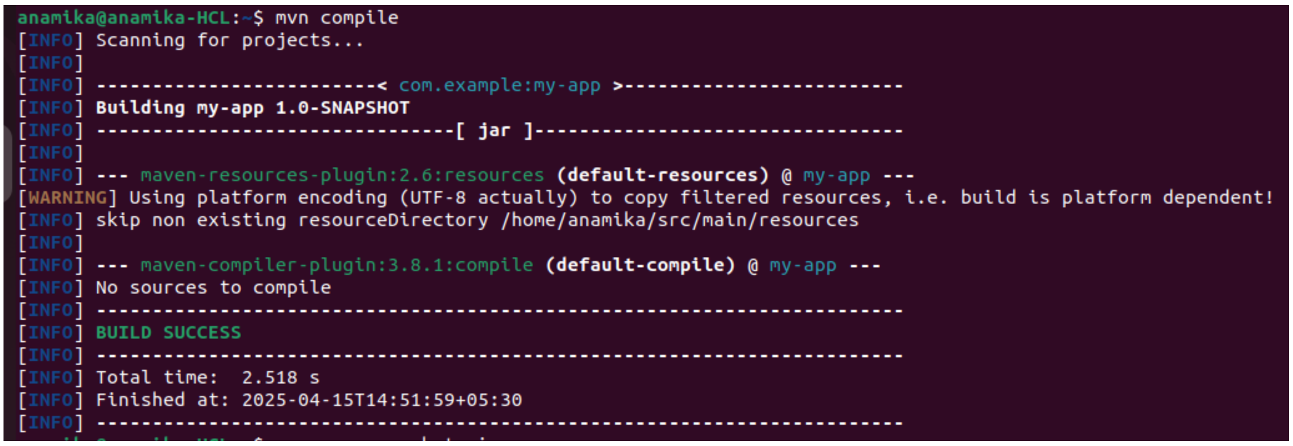
Create pom file ;
```
vi pom.xml
```

code:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://.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.example</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>my-app</name>
    <description>A simple Maven project</description>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Example: JUnit 5 for testing -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.8.1</version>
            <scope>test</scope>
        </dependency>

        <!-- Example: Gson for JSON processing -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.9</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Compiler Plugin -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>

            <!-- Surefire Plugin for running unit tests -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M7</version>
            </plugin>
             <!-- Assembly Plugin for creating an executable JAR -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.4.0</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

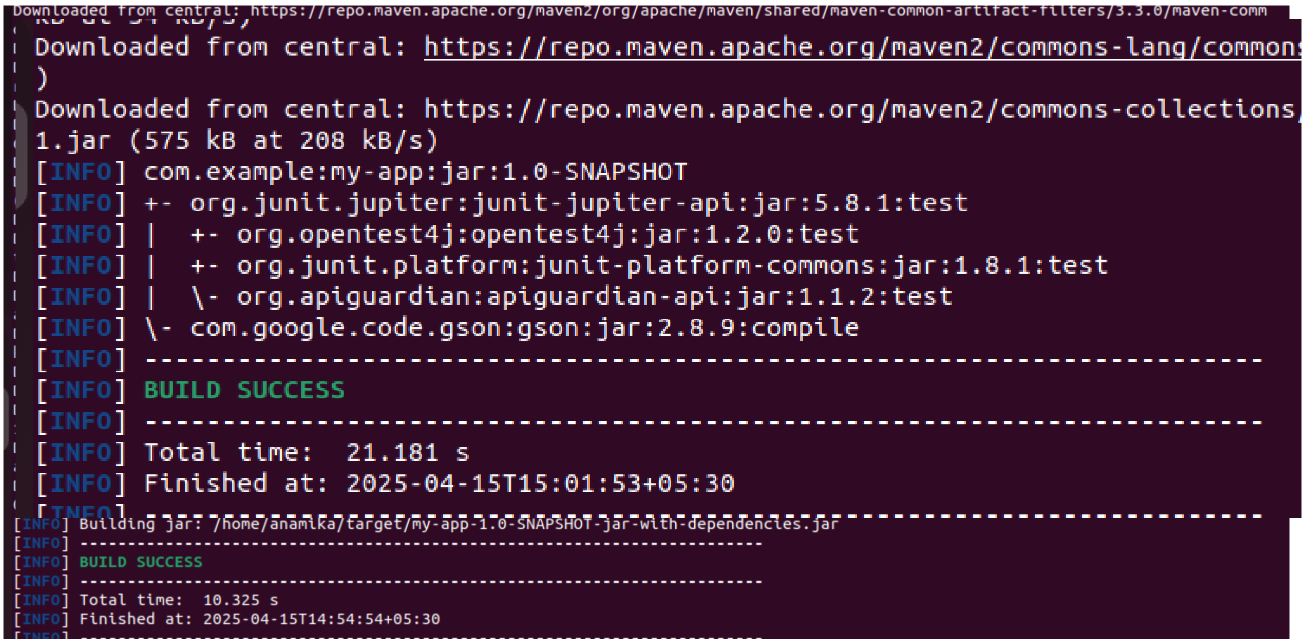
Result:

mvn compile



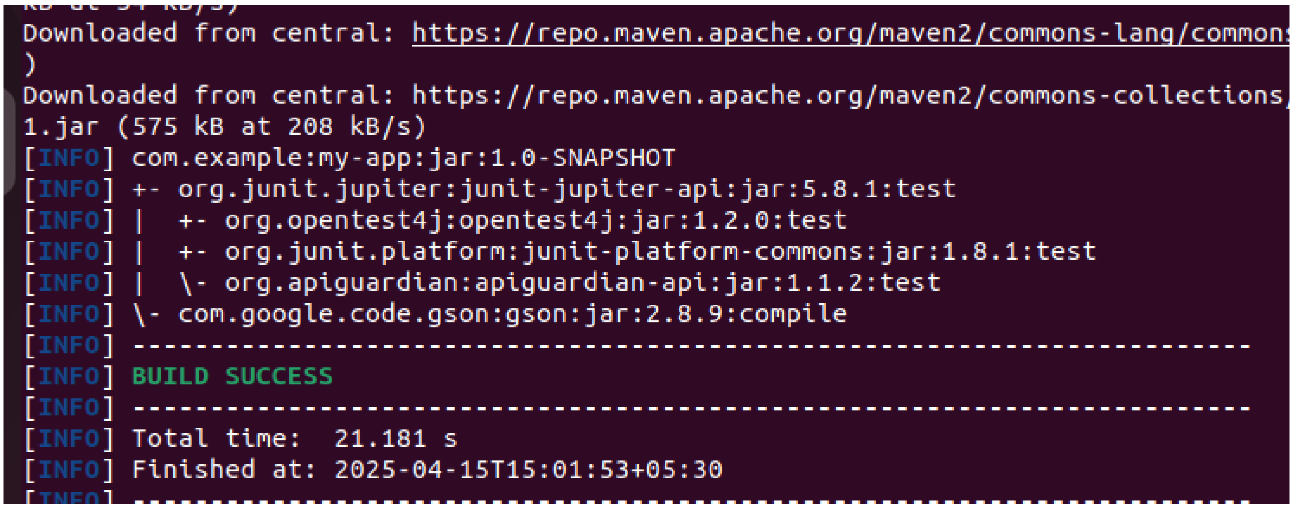
```
anamika@anamika-HCL:~$ mvn compile
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< com.example:my-app >-------------------------
[INFO] Building my-app 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ my-app ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/anamika/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ my-app ---
[INFO] No sources to compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.518 s
[INFO] Finished at: 2025-04-15T14:51:59+05:30
[INFO] ------------------------------------------------------------------------
```

mvn package



```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-common-artifact-filters/3.3.0/maven-comm
Downloaded from central: https://repo.maven.apache.org/maven2/commons-lang/common
)
Downloaded from central: https://repo.maven.apache.org/maven2/commons-collections
1.jar (575 kB at 208 kB/s)
[INFO] com.example:my-app:jar:1.0-SNAPSHOT
[INFO] +- org.junit.jupiter:junit-jupiter-api:jar:5.8.1:test
[INFO] |  +- org.opentest4j:opentest4j:jar:1.2.0:test
[INFO] |  +- org.junit.platform:junit-platform-commons:jar:1.8.1:test
[INFO] |  \- org.apiguardian:apiguardian-api:jar:1.1.2:test
[INFO] \- com.google.code.gson:gson:jar:2.8.9:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  21.181 s
[INFO] Finished at: 2025-04-15T15:01:53+05:30
[INFO] Building jar: /home/anamika/target/my-app-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.325 s
[INFO] Finished at: 2025-04-15T14:54:54+05:30
```

mvn dependency:tree



```
Downloaded from central: https://repo.maven.apache.org/maven2/commons-lang/common
)
Downloaded from central: https://repo.maven.apache.org/maven2/commons-collections
1.jar (575 kB at 208 kB/s)
[INFO] com.example:my-app:jar:1.0-SNAPSHOT
[INFO] +- org.junit.jupiter:junit-jupiter-api:jar:5.8.1:test
[INFO] |  +- org.opentest4j:opentest4j:jar:1.2.0:test
[INFO] |  +- org.junit.platform:junit-platform-commons:jar:1.8.1:test
[INFO] |  \- org.apiguardian:apiguardian-api:jar:1.1.2:test
[INFO] \- com.google.code.gson:gson:jar:2.8.9:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  21.181 s
[INFO] Finished at: 2025-04-15T15:01:53+05:30
```

Key Sections of pom.xml

(i) <modelVersion>

Defines the POM model version. Currently, it's always 4.0.0.

(ii) Project Coordinates

These uniquely identify a Maven project:

- <groupId> → The organization or project name (e.g., com.example).
- <artifactId> → The project name (e.g., my-app).
- <version> → The version of the project (e.g., 1.0-SNAPSHOT).

(iii) <packaging>

Defines the output type:

- jar (default) → For Java libraries.
- war → For web applications.
- pom → For parent projects.

(iV)<dependencies>

Manages project dependencies. Each dependency has:

- <groupId> → Organization or vendor.
- <artifactId> → The name of the library.
- <version> → Specific version.
- <scope> (optional) → Defines where it's used (compile, test, provided, etc.).

(v) <build>

Defines the build process, including:

- Plugins (e.g., exec-maven-plugin to run Java apps).
- Custom configurations.
- Running Maven Goals

Some useful Maven commands:

```
mvn compile                                      # Compiles the code
mvn test                                         # Runs tests
mvn package                                      # Packages the project into a JAR/WAR
mvn clean install                                # Cleans and installs dependencies
mvn exec:java -Dexec.mainClass="com.example.App" # Runs Java class
```

The pom.xml is the heart of a Maven project, controlling dependencies, build configurations, and plugins. Understanding its structure helps manage Java projects efficiently.

What Are Maven Plugins?

Maven plugins extend its functionality by enabling tasks such as compiling code, running tests, packaging applications, and deploying them. There are two types of plugins:

1. Build Plugins → Run during different build phases (e.g., maven-compiler-plugin for compilation).
2. Reporting Plugins → Generate reports (e.g., maven-surefire-plugin for test reports).

Experiment 3: Working with Gradle:

Setting up a gradle project, understanding build scripts(groovy and Kotlin DSL), Dependency management and Task Automation.

(Note: better to use java 17 version)
Update java:

```
sudo apt update
sudo apt install openjdk-17-jdk
java --version
```

(Note: Once gradle installation done, go with below steps)

Create a New Gradle Project:

Run the following command to generate a new Gradle project:

```
gradle init
```

You will be prompted to select a project type:

- **Application**: For Java/Kotlin projects that create executables.
- **Library**: For reusable Java/Kotlin libraries.
- **Basic**: A simple Gradle setup without predefined conventions.

Example selection:

- Select **"application"** for a Java project.
- Choose **"Java"** as the language.
- Pick a **build script DSL**: either **Groovy** (default) or **Kotlin**.
- Set the package name (optional).

Alternatively, create a simple Java application project directly:

```
gradle init --type java-application
```

code:

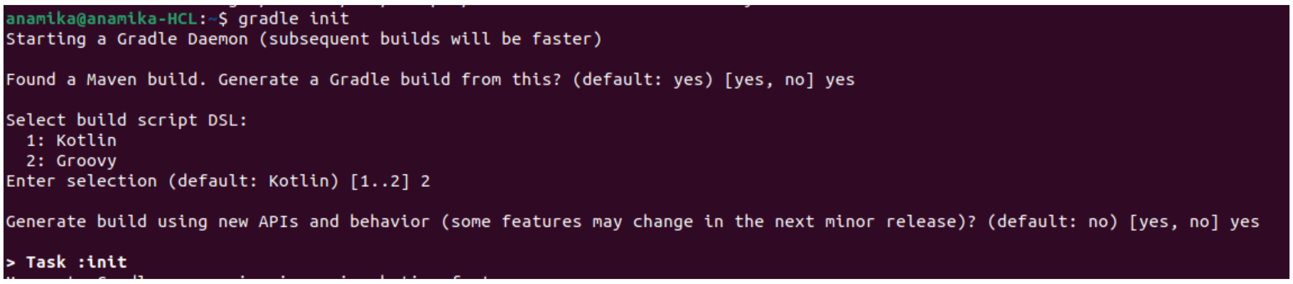
build.gradle ->

```
plugins {
   id 'java'
   id 'application'
}
group = 'com.example'
version = '1.0.0'
```

```
java {
    sourceCompatibility = JavaVersion.VERSION_17
    targetCompatibility = JavaVersion.VERSION_17
}
application {
    mainClass = 'com.example.Main'
}
repositories {
    mavenCentral()
}
dependencies {
    // Use latest stable JUnit for testing
    testImplementation 'org.junit.jupiter:junit-jupiter:5.9.2'

    // Example dependency: Gson for JSON parsing
    implementation 'com.google.code.gson:gson:2.10'
}
tasks.withType(JavaCompile).configureEach {
    options.encoding = 'UTF-8'
}
test {
    useJUnitPlatform()
}
```

Result:

gradle init

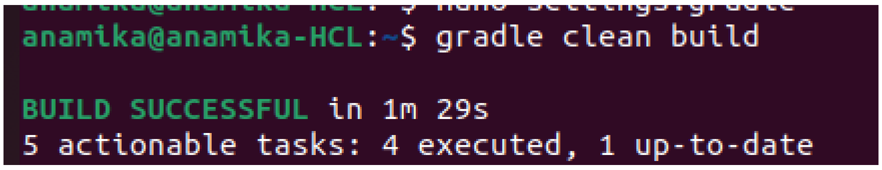


```
anamika@anamika-HCL:~$ gradle init
Starting a Gradle Daemon (subsequent builds will be faster)

Found a Maven build. Generate a Gradle build from this? (default: yes) [yes, no] yes

Select build script DSL:
  1: Kotlin
  2: Groovy
Enter selection (default: Kotlin) [1..2] 2

Generate build using new APIs and behavior (some features may change in the next minor release)? (default: no) [yes, no] yes

> Task :init
```

gradle build

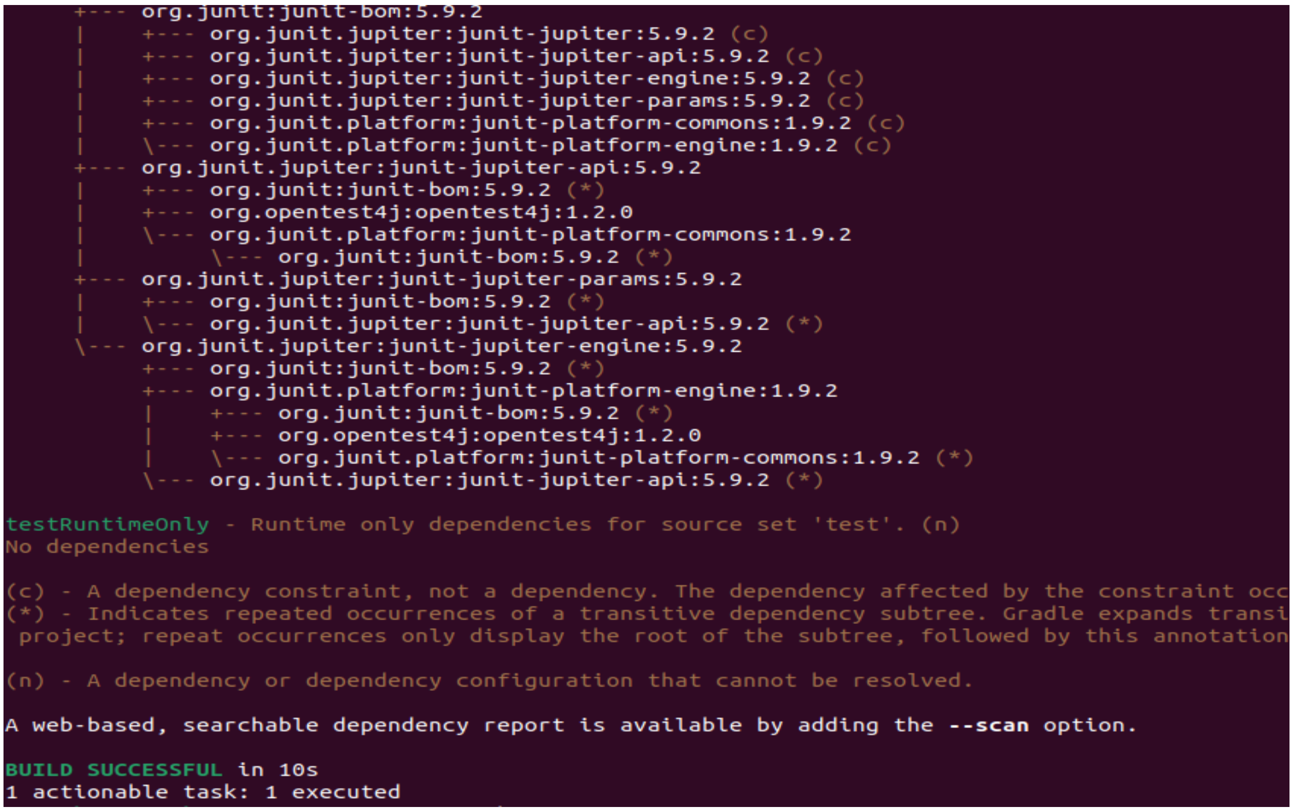
or

gradle clean build



```
anamika@anamika-HCL:~$ gradle clean build

BUILD SUCCESSFUL in 1m 29s
5 actionable tasks: 4 executed, 1 up-to-date
```

gradle dependencies



```
     +--- org.junit:junit-bom:5.9.2
     |    +--- org.junit.jupiter:junit-jupiter:5.9.2 (c)
     |    +--- org.junit.jupiter:junit-jupiter-api:5.9.2 (c)
     |    +--- org.junit.jupiter:junit-jupiter-engine:5.9.2 (c)
     |    +--- org.junit.jupiter:junit-jupiter-params:5.9.2 (c)
     |    +--- org.junit.platform:junit-platform-commons:1.9.2 (c)
     |    \--- org.junit.platform:junit-platform-engine:1.9.2 (c)
     +--- org.junit.jupiter:junit-jupiter-api:5.9.2
     |    +--- org.junit:junit-bom:5.9.2 (*)
     |    +--- org.opentest4j:opentest4j:1.2.0
     |    \--- org.junit.platform:junit-platform-commons:1.9.2
     |         \--- org.junit:junit-bom:5.9.2 (*)
     +--- org.junit.jupiter:junit-jupiter-params:5.9.2
     |    +--- org.junit:junit-bom:5.9.2 (*)
     |    \--- org.junit.jupiter:junit-jupiter-api:5.9.2 (*)
     \--- org.junit.jupiter:junit-jupiter-engine:5.9.2
          +--- org.junit:junit-bom:5.9.2 (*)
          +--- org.junit.platform:junit-platform-engine:1.9.2
          |    +--- org.junit:junit-bom:5.9.2 (*)
          |    +--- org.opentest4j:opentest4j:1.2.0
          |    \--- org.junit.platform:junit-platform-commons:1.9.2 (*)
          \--- org.junit.jupiter:junit-jupiter-api:5.9.2 (*)

testRuntimeOnly - Runtime only dependencies for source set 'test'. (n)
No dependencies

(c) - A dependency constraint, not a dependency. The dependency affected by the constraint occ
(*) - Indicates repeated occurrences of a transitive dependency subtree. Gradle expands transi
 project; repeat occurrences only display the root of the subtree, followed by this annotation

(n) - A dependency or dependency configuration that cannot be resolved.

A web-based, searchable dependency report is available by adding the --scan option.

BUILD SUCCESSFUL in 10s
1 actionable task: 1 executed
```

If Main.java doesnot exist, create it as below

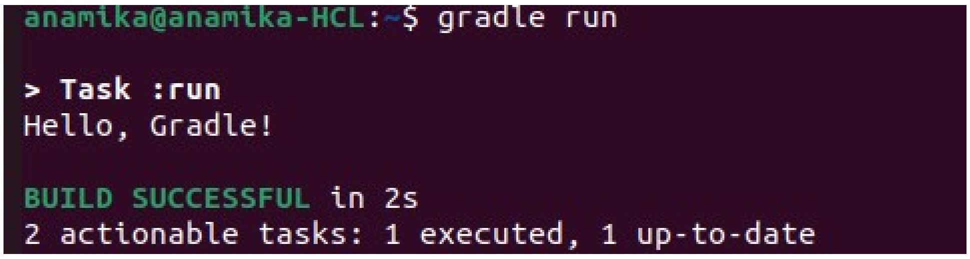
```
mkdir -p src/main/java/com/example

vi src/main/java/com/example/Main.java
```

code:

```
package com.example;


public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, Gradle!");
    }
}
```

save and exit then run the project(gradle run)

```
gradle run
```



```
anamika@anamika-HCL:~$ gradle run

> Task :run
Hello, Gradle!

BUILD SUCCESSFUL in 2s
2 actionable tasks: 1 executed, 1 up-to-date
```

Task Automation in Gradle:

Gradle tasks automate processes like compilation, testing, packaging, and deployment. You can use built-in tasks or create custom tasks to streamline your workflow.

1. Running Built-in Tasks

Gradle provides many predefined tasks. Some common ones are:

Command	Description
gradle tasks	Lists available tasks.
gradle build	Compiles, tests, and packages the project.
gradle clean	Deletes the build/ directory.
gradle test	Runs unit tests.
gradle run	Runs the application (if using application plugin).
gradle dependencies	Lists all dependencies.

2. Creating Custom Tasks

You can define custom tasks in build.gradle:

a) Simple Task

```
tasks.register('hello') {
    doLast {
        println 'Hello, Gradle!'
    }
}
```

Run it with:

```
gradle hello
```

b) Task with Multiple Actions

```
tasks.register('greet') {
    doFirst {
        println 'Starting the task...'
    }
    doLast {
        println 'Hello, World!'
    }
}
```

doFirst runs before the main action.

doLast runs after the main action.

3. …

5

Experiment 4: Practical Exercise:

Build and run a java application with Maven, Migrate the same Application to Gradle

(Note: As in experiment 2 build maven project as below , after building migrate the same to gradle)

Create a New Maven Project:

Use the following command to generate a new Maven project:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

navigate to project directory;

```
cd my-app
```

Create pom file ;

```
vi pom.xml
```

code:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://.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.example</groupId>
    <artifactId>my-app</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>my-app</name>
    <description>A simple Maven project</description>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Example: JUnit 5 for testing -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.8.1</version>
            <scope>test</scope>
        </dependency>
        <!-- Example: Gson for JSON processing -->
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.9</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <!-- Compiler Plugin -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>

            <!-- Surefire Plugin for running unit tests -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M7</version>
            </plugin>

            <!-- Assembly Plugin for creating an executable JAR -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>3.4.0</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <descriptorRefs>
                                <descriptorRef>jar-with-dependencies</descriptorRef>
                            </descriptorRefs>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

build:
```
mvn compile
```

Convert Maven to Gradle (automatically)

Gradle provides an automatic way to convert a Maven project:

```
cd my-app

gradle init --type pom
```

This generates a Gradle build script based on pom.xml.

Experiment 5: Introduction to Jenkins
What is Jenkins?, installing jenkins on local or cloud environment, configuaring jenkins for first use

Jenkins:
Jenkins is a continuous integration tool or automation tool.

Step 1: Install Java (Jenkins Requires Java)

Jenkins requires Java 11 or newer.

sudo apt install fontconfig openjdk-17-jre -y

Step 2: Add Jenkins Repository and Install Jenkins

wget -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

Note: if got error go with curl installation, after that install jenkins

sudo apt update

sudo apt install curl -y

curl -fsSL https://pkg.jenkins.io/debian/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 binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update

sudo apt install jenkins -y

jenkins --version

Step 3: Start and Enable Jenkins Service

sudo systemctl enable --now jenkins

sudo systemctl status jenkins

If Jenkins is not running, start it manually:

sudo systemctl start jenkins

Step 4: Open Firewall for Jenkins (If Required)

sudo ufw allow 8080

sudo ufw enable

sudo ufw status

Step 5: Access Jenkins Web Interface

Open a browser and go to:

http://localhost:8080

To get the initial admin password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

note:

it may look like this below

(03eed19f8f7244488e0949e2b4c12766)

paste it in jenkins dashboard

Step 7:

Jenkins dashboard:http://localhost:8080

It will open jenkins dashboard

Experiment 6: Continuous Intergration with jenkins:
Setting up a CI pipeline, Integrating jenkins with Maven/Gradle, Running automated builds and tests.

Step 1:
go to git dashboard
create account , if already created use same account
a) create new repo name it as “jenkins-ci-pipeline”

Go to terminal:
Update System Packages:--

```
sudo apt update && sudo apt upgrade -y
sudo apt install git -y
git  --version
git init
git config --global user.name "namratha"
git config --global user.email "nammu4324@gmail.com"
git add .
git commit -m "adding pom.xml"
git branch exp6
git checkout exp6
git push –all
git remote -v
git remote add master https://github.com/namrathank/jenkins-ci-pipeline.git
git push –all
git push -u origin --all
```

Step 2: Install Java (Jenkins Requires Java)

Jenkins requires Java 11 or newer.

```
sudo apt install fontconfig openjdk-17-jre -y
```

Step 3: Add Jenkins Repository and Install Jenkins

```
wget -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null
```

Note: if got error go with curl installation, after that install jenkins

```
sudo apt update

 sudo apt install curl -y

 curl -fsSL https://pkg.jenkins.io/debian/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 binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt update

sudo apt install jenkins -y

jenkins --version

Step 4: Start and Enable Jenkins Service

sudo systemctl enable --now jenkins

sudo systemctl status jenkins

If Jenkins is not running, start it manually:

sudo systemctl start jenkins

Step 5: Open Firewall for Jenkins (If Required)

sudo ufw allow 8080

sudo ufw enable

sudo ufw status

Step 6: Access Jenkins Web Interface

Open a browser and go to:

http://localhost:8080

To get the initial admin password:

sudo cat /var/lib/jenkins/secrets/initialAdminPassword

note:

it may look like this below

(03eed19f8f7244488e0949e2b4c12766)

paste it in jenkins dashboard

Step 7:

Jenkins dashboard:http://localhost:8080

Go to jenkins dashboard and do the belolw configuration

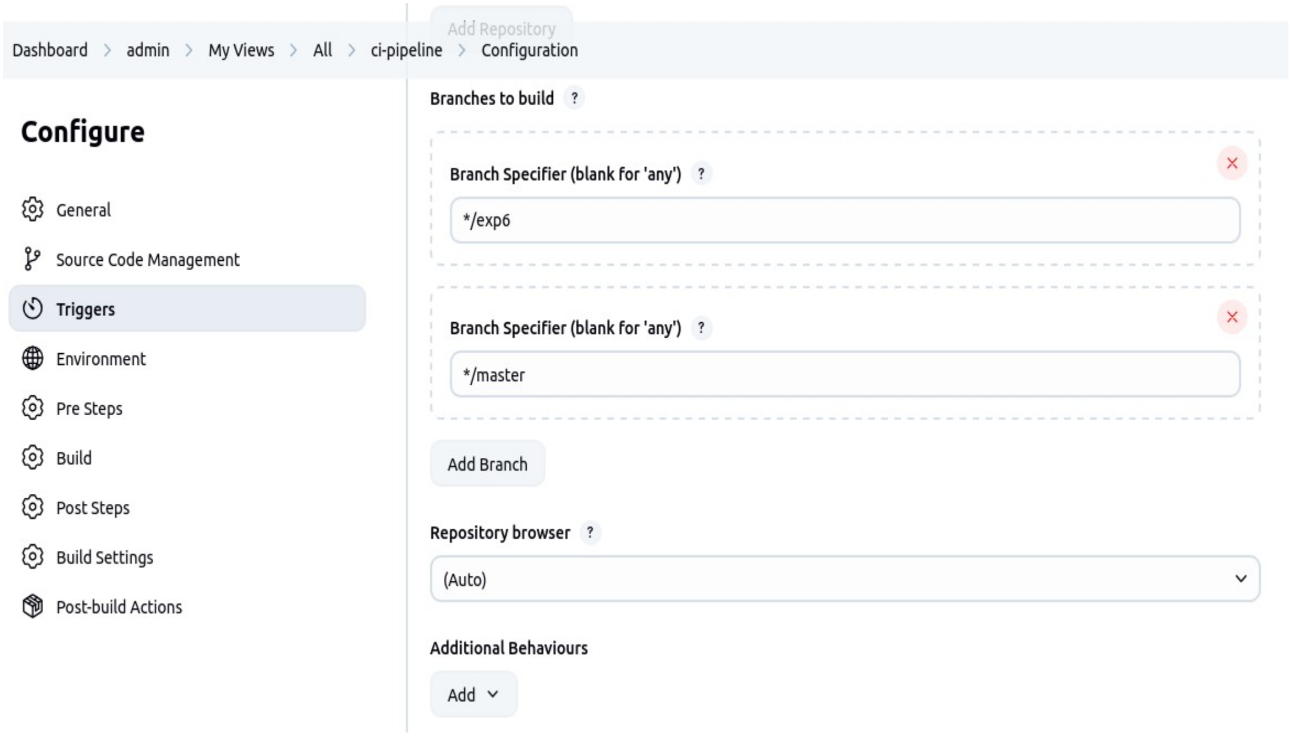
go to manage jenkin -> system configuration -> add plugin

plugins to be add:

a) Maven Integration

b) junit

c) git



Dashboard > admin > My Views > All > ci-pipeline > Configuration

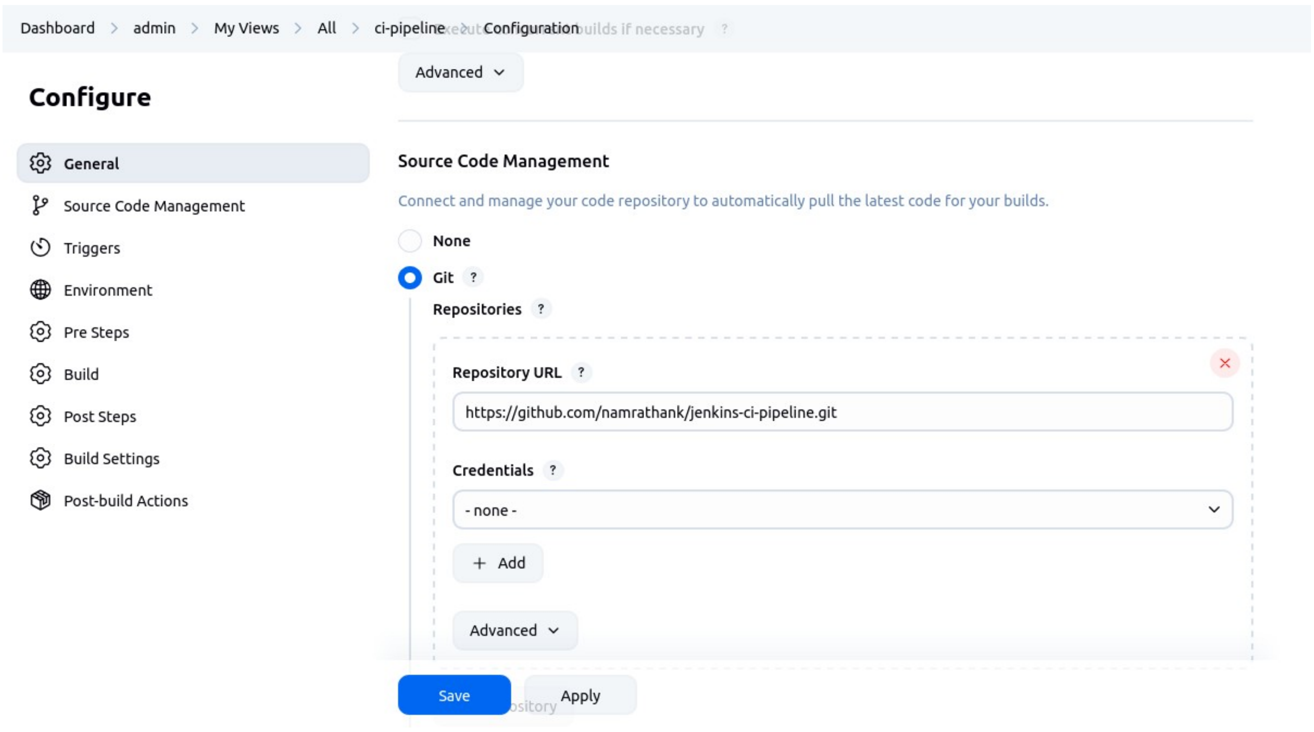
Branches to build

Branch Specifier (blank for 'any'): */exp6

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Additional Behaviours



Dashboard > admin > My Views > All > ci-pipeline > Configuration

Source Code Management

Connect and manage your code repository to automatically pull the latest code for your builds.

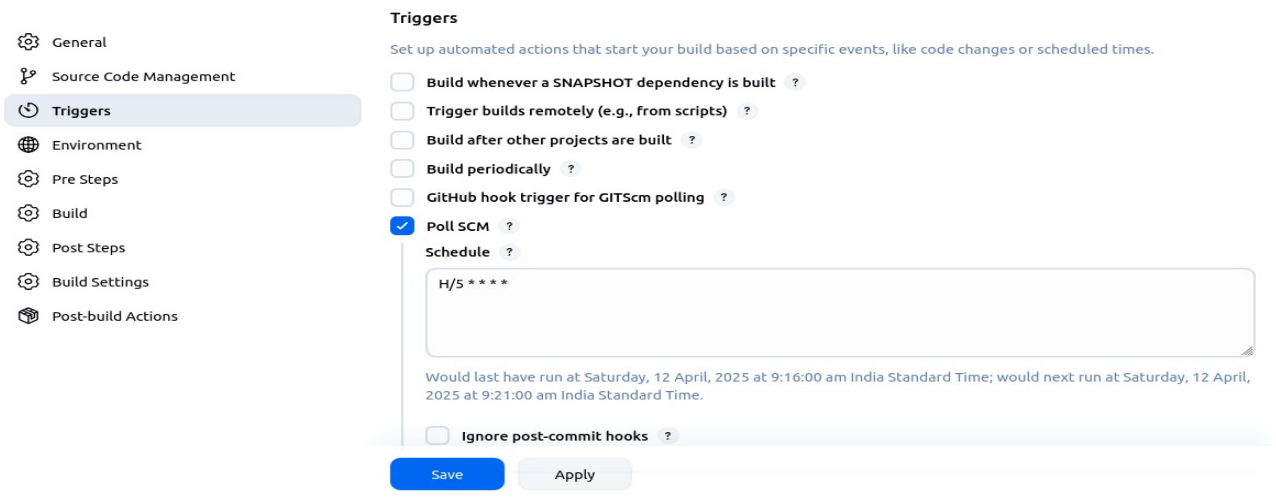
None [✓ Git]

Repositories

Repository URL: https://github.com/namrathank/jenkins-ci-pipeline.git

Credentials: - none -

Copy and paste repository url from your github



Triggers

Set up automated actions that start your build based on specific events, like code changes or scheduled times.

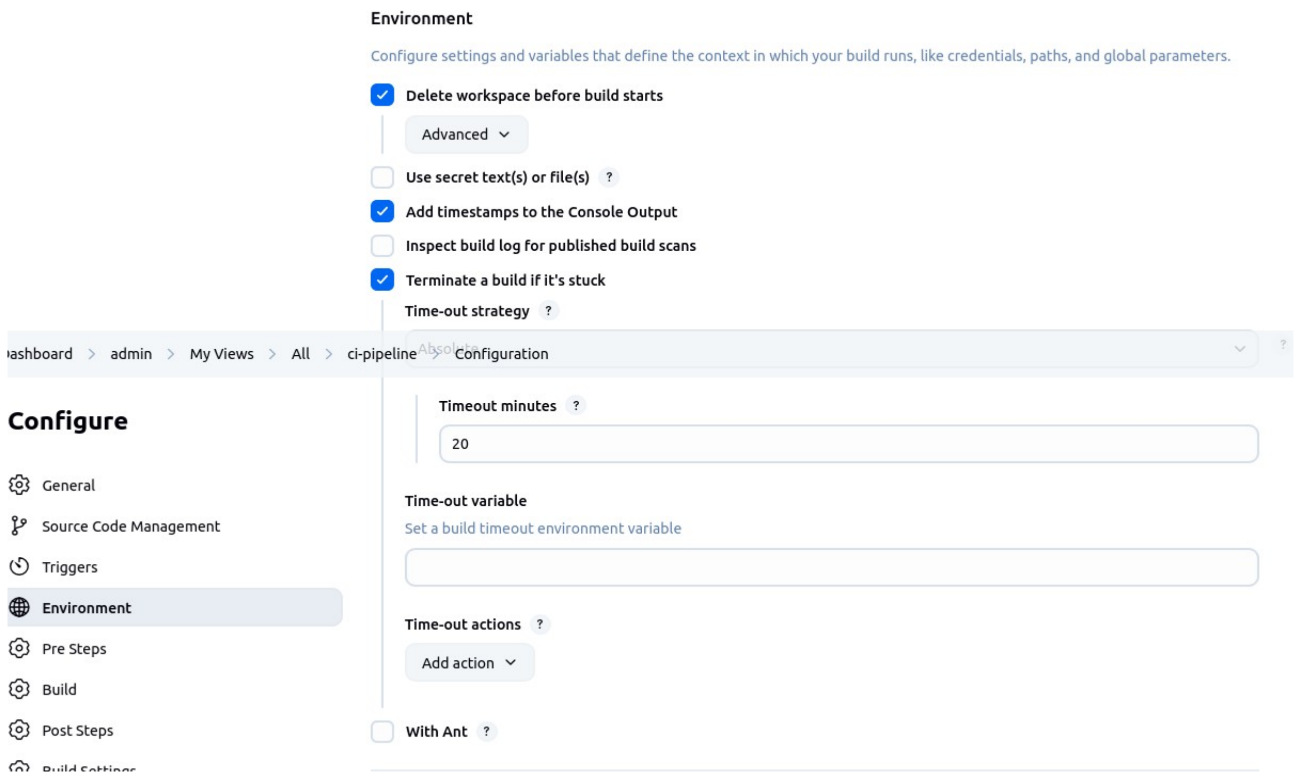
- [ ] Build whenever a SNAPSHOT dependency is built
- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [ ] Build periodically
- [ ] GitHub hook trigger for GITScm polling
- [✓] Poll SCM

Schedule

```
H/5 * * * *
```

Would last have run at Saturday, 12 April, 2025 at 9:16:00 am India Standard Time; would next run at Saturday, 12 April, 2025 at 9:21:00 am India Standard Time.

- [ ] Ignore post-commit hooks



Environment

Configure settings and variables that define the context in which your build runs, like credentials, paths, and global parameters.

- [✓] Delete workspace before build starts
- [ ] Use secret text(s) or file(s)
- [✓] Add timestamps to the Console Output
- [ ] Inspect build log for published build scans
- [✓] Terminate a build if it's stuck

Time-out strategy: Absolute

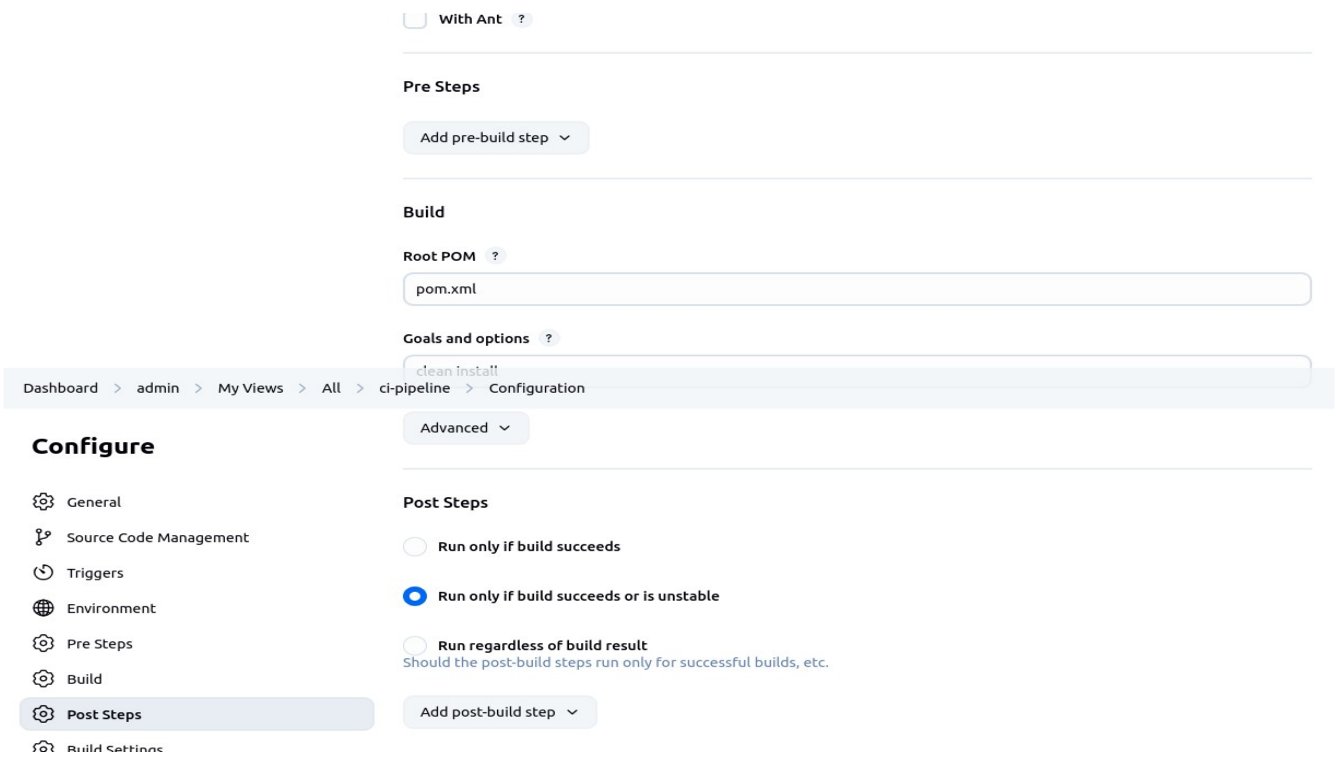
Timeout minutes: 20

Time-out variable

Set a build timeout environment variable

Time-out actions

- [ ] With Ant



Dashboard > admin > My Views > All > ci-pipeline > Configuration

With Ant

Pre Steps

Add pre-build step

Build

Root POM: pom.xml

Goals and options: clean install

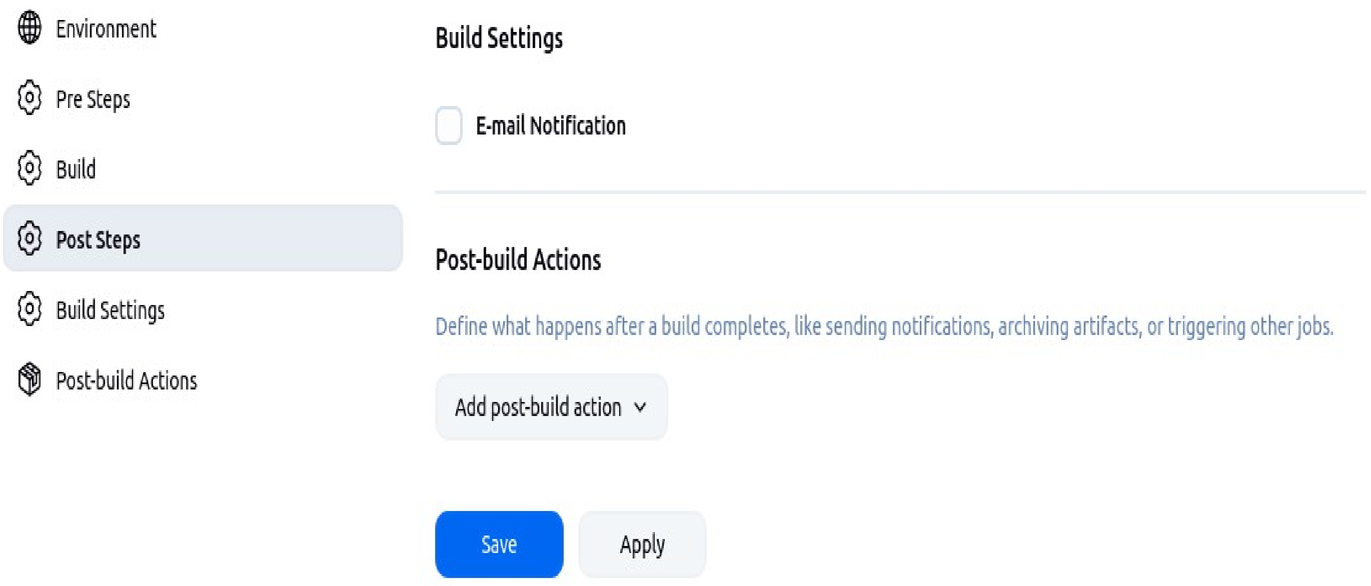
Advanced

Post Steps

Run only if build succeeds [✓ Run only if build succeeds or is unstable] Run regardless of build result

Should the post-build steps run only for successful builds, etc.

Add post-build step



Build Settings

E-mail Notification

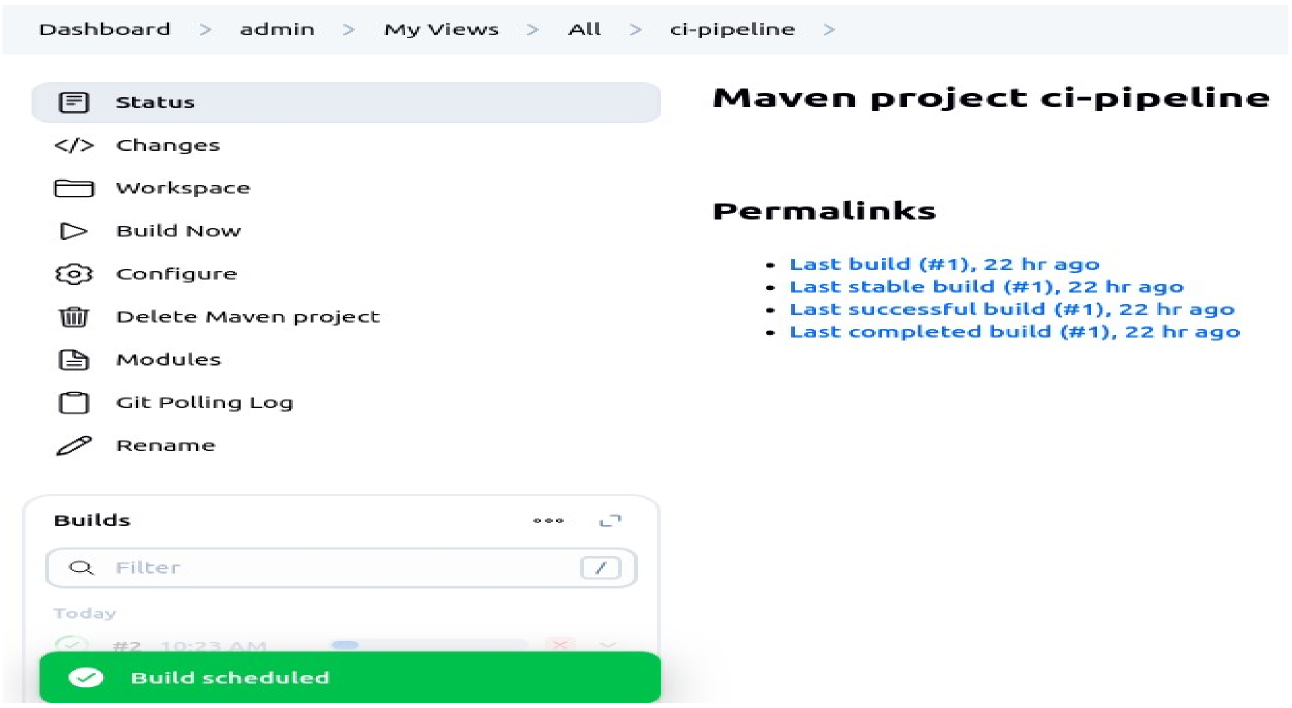
Post-build Actions

Define what happens after a build completes, like sending notifications, archiving artifacts, or triggering other jobs.

In build settings E-mail Notification is optional

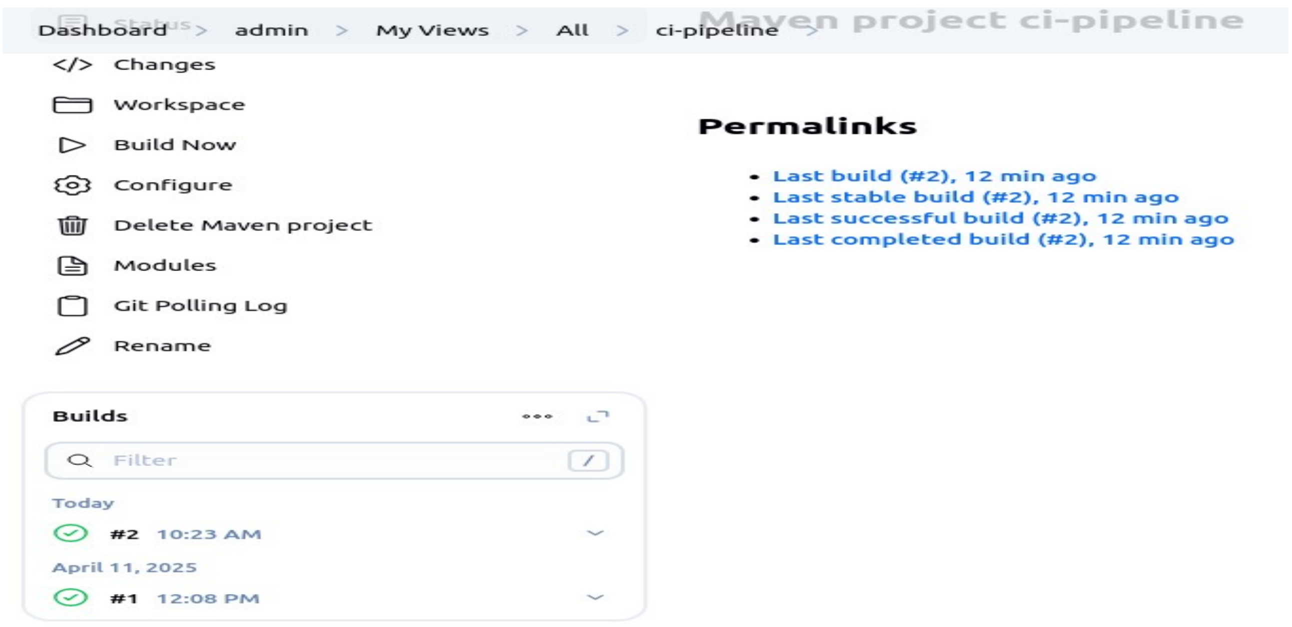
Save the configuration.

Once the all above configuration done build the CI pipeline by clicking the Build Now option as shown below

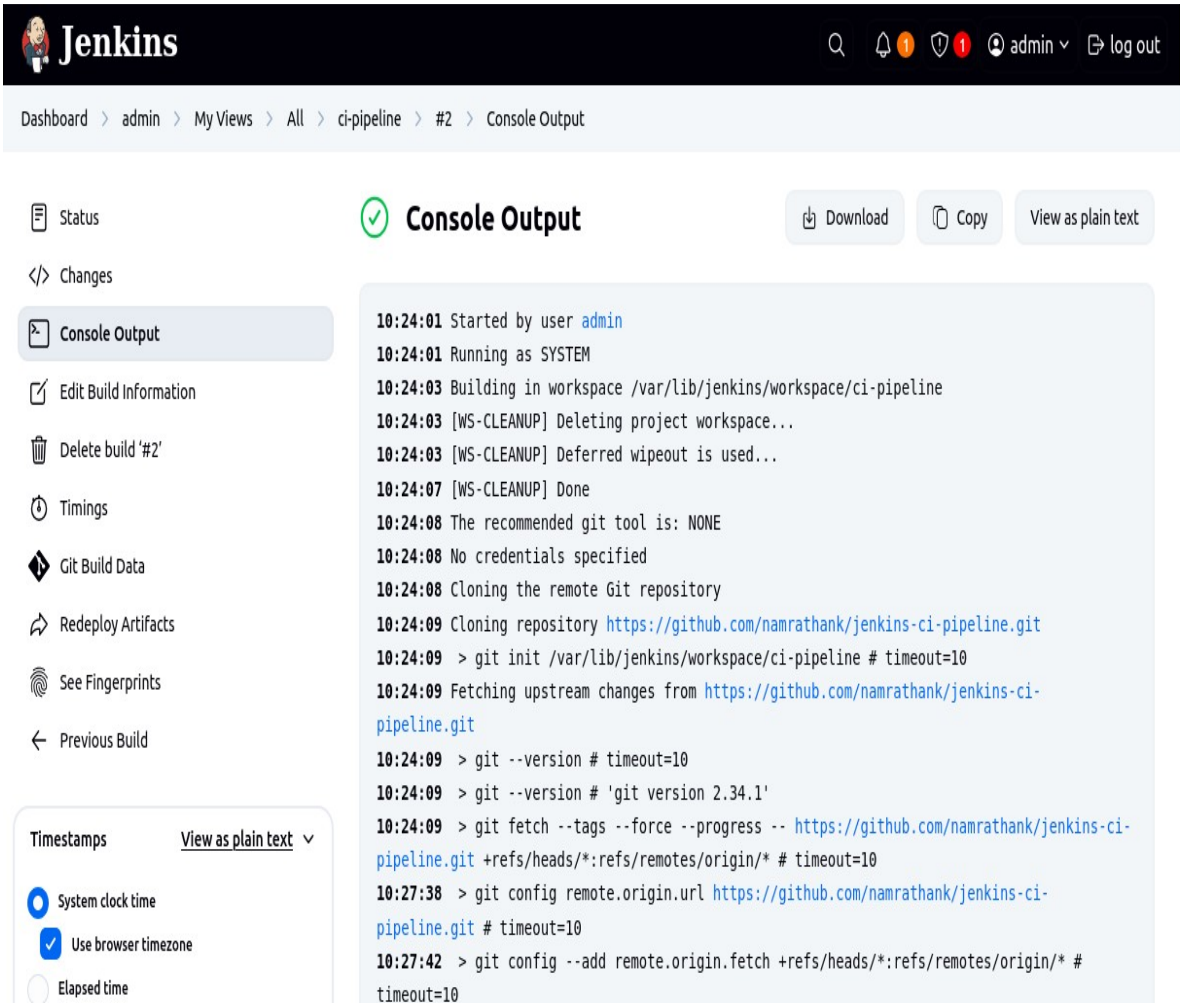

Dashboard > admin > My Views > All > ci-pipeline >
Status
Changes
Workspace
Build Now
Configure
Delete Maven project
Modules
Git Polling Log
Rename
Builds
Filter
Today
#2 10:23 AM
Build scheduled
Maven project ci-pipeline
Permalinks
Last build (#1), 22 hr ago
Last stable build (#1), 22 hr ago
Last successful build (#1), 22 hr ago
Last completed build (#1), 22 hr ago

Result will be the Console output:

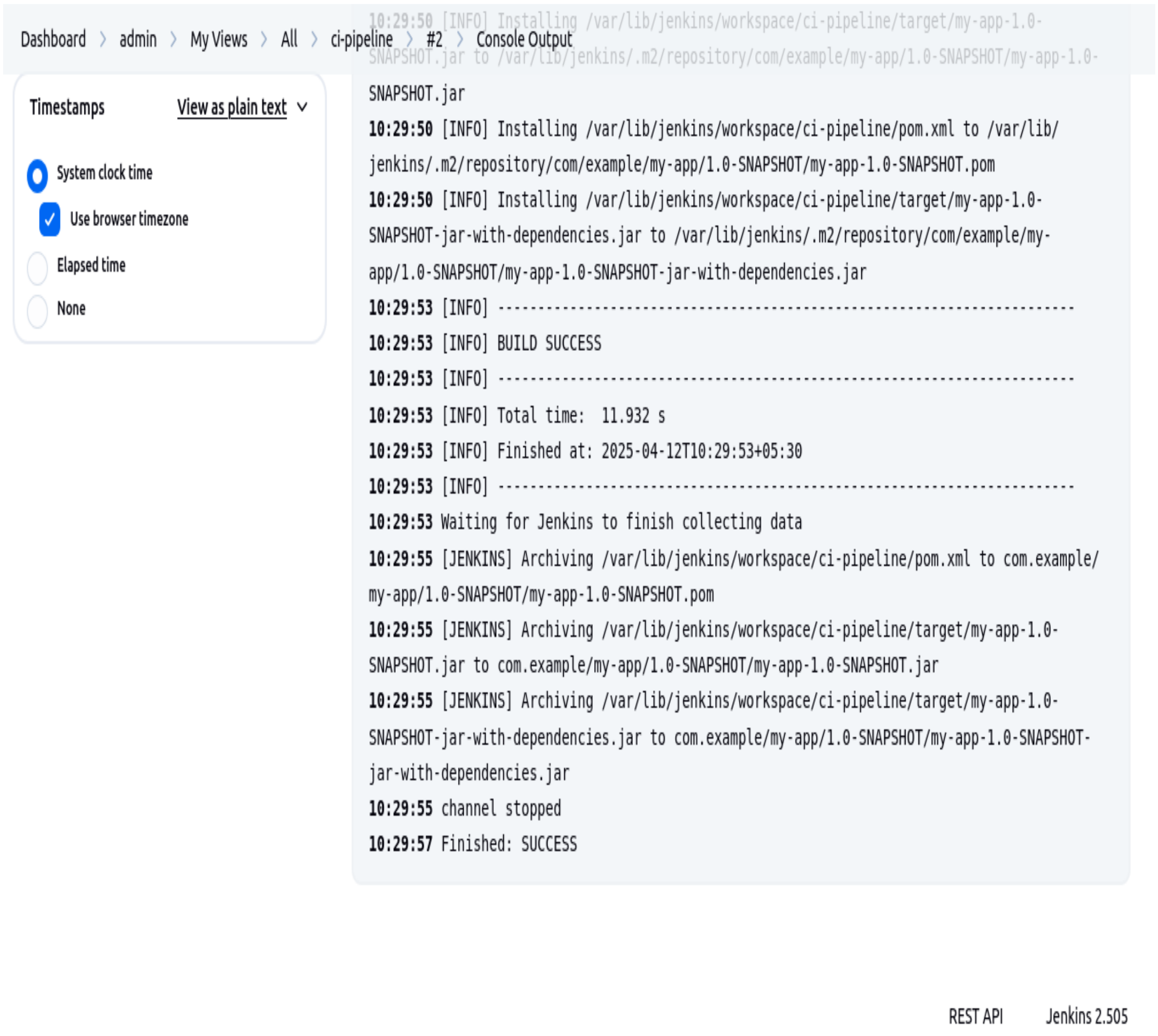
Once build done click on last build number as shown below: (Ex:- #2)


Dashboard > admin > My Views > All > ci-pipeline >
Maven project ci-pipeline
Changes
Workspace
Build Now
Configure
Delete Maven project
Modules
Git Polling Log
Rename
Builds
Filter
Today
#2 10:23 AM
April 11, 2025
#1 12:08 PM
Permalinks
Last build (#2), 12 min ago
Last stable build (#2), 12 min ago
Last successful build (#2), 12 min ago
Last completed build (#2), 12 min ago

Click on Console output to view result:



```
10:24:01 Started by user admin
10:24:01 Running as SYSTEM
10:24:03 Building in workspace /var/lib/jenkins/workspace/ci-pipeline
10:24:03 [WS-CLEANUP] Deleting project workspace...
10:24:03 [WS-CLEANUP] Deferred wipeout is used...
10:24:07 [WS-CLEANUP] Done
10:24:08 The recommended git tool is: NONE
10:24:08 No credentials specified
10:24:08 Cloning the remote Git repository
10:24:09 Cloning repository https://github.com/namrathank/jenkins-ci-pipeline.git
10:24:09  > git init /var/lib/jenkins/workspace/ci-pipeline # timeout=10
10:24:09 Fetching upstream changes from https://github.com/namrathank/jenkins-ci-
pipeline.git
10:24:09  > git --version # timeout=10
10:24:09  > git --version # 'git version 2.34.1'
10:24:09  > git fetch --tags --force --progress -- https://github.com/namrathank/jenkins-ci-
pipeline.git +refs/heads/*:refs/remotes/origin/* # timeout=10
10:27:38  > git config remote.origin.url https://github.com/namrathank/jenkins-ci-
pipeline.git # timeout=10
10:27:42  > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* #
timeout=10
```



```
10:29:50 [INFO] Installing /var/lib/jenkins/workspace/ci-pipeline/target/my-app-1.0-
SNAPSHOT.jar to /var/lib/jenkins/.m2/repository/com/example/my-app/1.0-SNAPSHOT/my-app-1.0-
SNAPSHOT.jar
10:29:50 [INFO] Installing /var/lib/jenkins/workspace/ci-pipeline/pom.xml to /var/lib/
jenkins/.m2/repository/com/example/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.pom
10:29:50 [INFO] Installing /var/lib/jenkins/workspace/ci-pipeline/target/my-app-1.0-
SNAPSHOT-jar-with-dependencies.jar to /var/lib/jenkins/.m2/repository/com/example/my-
app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT-jar-with-dependencies.jar
10:29:53 [INFO] ------------------------------------------------------------------------
10:29:53 [INFO] BUILD SUCCESS
10:29:53 [INFO] ------------------------------------------------------------------------
10:29:53 [INFO] Total time:  11.932 s
10:29:53 [INFO] Finished at: 2025-04-12T10:29:53+05:30
10:29:53 [INFO] ------------------------------------------------------------------------
10:29:53 Waiting for Jenkins to finish collecting data
10:29:55 [JENKINS] Archiving /var/lib/jenkins/workspace/ci-pipeline/pom.xml to com.example/
my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.pom
10:29:55 [JENKINS] Archiving /var/lib/jenkins/workspace/ci-pipeline/target/my-app-1.0-
SNAPSHOT.jar to com.example/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
10:29:55 [JENKINS] Archiving /var/lib/jenkins/workspace/ci-pipeline/target/my-app-1.0-
SNAPSHOT-jar-with-dependencies.jar to com.example/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT-
jar-with-dependencies.jar
10:29:55 channel stopped
10:29:57 Finished: SUCCESS
```

As above, build should get success.

This completes with CI pipeline in jenkins