**DevOps BootCamp**

**Prerequisite**1) Java 17 or Above  
2) Git  
3) GitHub Account  
4) Maven 3.9.9 (Build Tool)  
5) SonarQube (sonarqube-25.4.0.105899)  
6) Jenkins (Latest Version)  
7) VS Code IDE or any other equivalent  
Note: Workshop will be conducted using Windows OS Machine; however, Students can use Linux/CentOS based on their convenience

**Workshop Part-1**

**Run the Application in Local System Using Maven**

**JAVA Installation and Configuration (7 Minutes)**

Install Java 17  
<https://www.openlogic.com/openjdk-downloads>

Download Zipped Version or MSI version(*preferred*)  
*Installing in the default directory(program files) is preferred, as sometimes installing in other location does not work sometimes*

Step 1: Setting JAVA-HOME in Windows

To set the JRE\_HOME or JAVA\_HOME variable:

1. Locate your Java installation directory

C:\jdk-17

You can also type where java at the command prompt.

**Add Java to PATH**

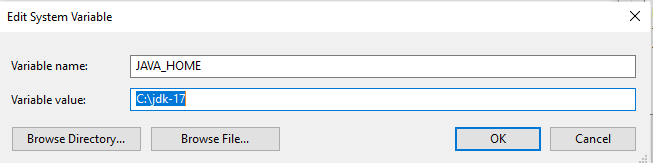
Press Win + S → Type **Environment Variables** → Click **Edit the system environment variables**

In the **System Properties** window, click **Environment Variables**

Under **System variables**, find and select the variable Path, then click **Edit**

Click **New** and add: C:\jdk-17\bin

1. Do one of the following:  
   **Windows 7** – Right click **My Computer** and select **Properties** > **Advanced**  
   **Windows 8** – Go to **Control Panel** > **System** > **Advanced System Settings  
   Windows 10**– Search for **Environment Variables** then select **Edit the system environment variables**
2. Click the **Environment Variables** button.
3. Under **System Variables**, click **New**.
4. In the **Variable Name** field, enter either:
   * JAVA\_HOME if you installed the JDK (Java Development Kit)  
     or
   * JRE\_HOME if you installed the JRE (Java Runtime Environment)
5. In the **Variable Value** field, enter your JDK or JRE installation path.



1. Click **OK**and **Apply Changes** as prompted

You'll need to close and re-open any command windows that were open before you made these changes, as there's no way to reload environment variables from an active command prompt. If the changes don't take effect after reopening the command window, restart Windows.

Step 2: Set the JAVA\_HOME variable via the command line (Optional)

If you would prefer to set the JAVA\_HOME (or JRE\_HOME) variable via the command line:

1. Open Command Prompt (make sure you Run as administrator so you're able to add a system environment variable).
2. Set the value of the environment variable to your JDK (or JRE) installation path as follows:

setx /m JAVA\_HOME "C:\jdk-17"

Restart Command Prompt to reload the environment variables then use the following command to check the it's been added correctly.

1. echo %JAVA\_HOME%

You should see the path to your JDK (or JRE) installation.

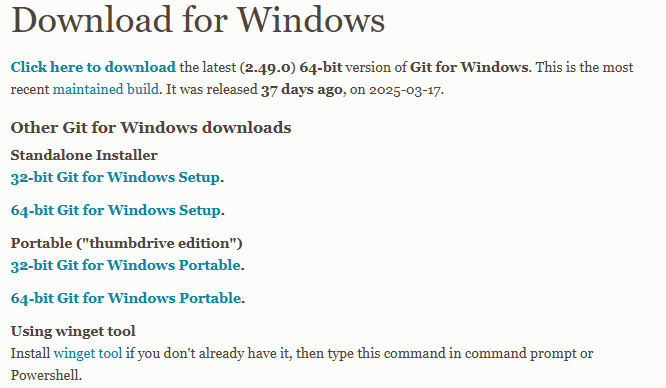
Open a CMD prompt and type java –version => you should see the details w.r.t java installation which justifies JAVA\_HOME is configured successfully

**GIT Installation and Working with Remote Code (5 Minutes)**

Step 1: Download Git

<https://git-scm.com/downloads>





Run the Installer (Choose Default Options)

Open Git CMD from Start Menu and type in git to check if GIT is installed successfully or not

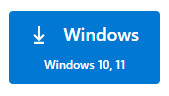
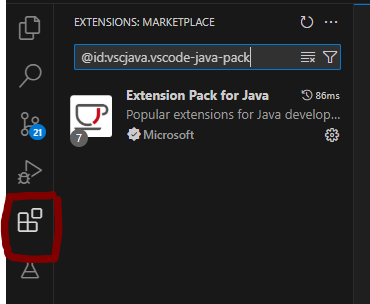
Step 2: Fork the remote Github repo (https://github.com/mahakud1/rest-api-crud-example.git)  
Forking a Git repository, particularly on platforms like GitHub or GitLab, is a workflow that involves creating a copy of an existing repository in your own account. You can then make changes to your forked copy. Follow the below mentioned steps .  
  
**a) GitHub/GitLab:** Navigate to the repository you want to fork on the platform (GitHub, GitLab, etc.).

**b) Fork Button:** Look for the "Fork" button, usually located on the main repository page.

**c) Forking:** Clicking this button will create a copy of the repository in your account.

**d) Cloning:** You can then clone the forked repository to your local machine to make changes.

**VS Code Java Editor (10 Mins)**  
Download VSCode => <https://code.visualstudio.com/download>, Select

  
Install the Extension Pack for Java, as shown below, this will help in compiling and running the application  


Import the Java Project which was cloned in the previous step or Create a Java Project

Open terminal inside VSCode and type ‘java -version’ it should show the java version which was installed already, which means the installation is successful.

Troubleshoot: if it does not show the java version, then restart VSCode

**How to push the changes to Remote GIT Repo**

Type the following commands in the VS Code Terminal one after the other   
git config --global user.email "srinivas.mahakud@gmail.com"

git config --global user.name "Srinivas Mahakud"

git status (will show which files were changed in local as compared to the remote)

git add (specify the file which showed in the above line)

git add . (add all files from the current location)  
  
$ git add -A, $ git add –all

git commit -m "refactored"

git push -u -f origin main (remember not master in this case)

**MAVEN Installation and Configuration (10 Mins)**

<https://srinivas-mahakud.medium.com/configure-maven-on-windows-b1a89da970f9>

Maven Life Cycle

<https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>

**Run the Application in Local System Using Maven**

You can follow these steps or alternatively you can refer the readme.md file

Open terminal in VSCode  
Go to the path where pom file is there.

Type the command *mvn clean package*

Upon successful execution of the above command, type *mvn spring-boot:run* in the terminal, this will start the application.

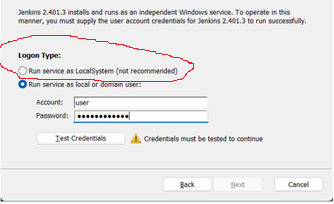
Open a browser, and type in [http://localhost:8080/swagger-ui/index.html](http://localhost:8080/swagger-ui/index.html/) to see the application end points  
Troubleshoot: If after typing the above URL, nothing gets shown. Then Remove the trailing / — some versions of Swagger UI don’t handle that well.

**Workshop Part-2**

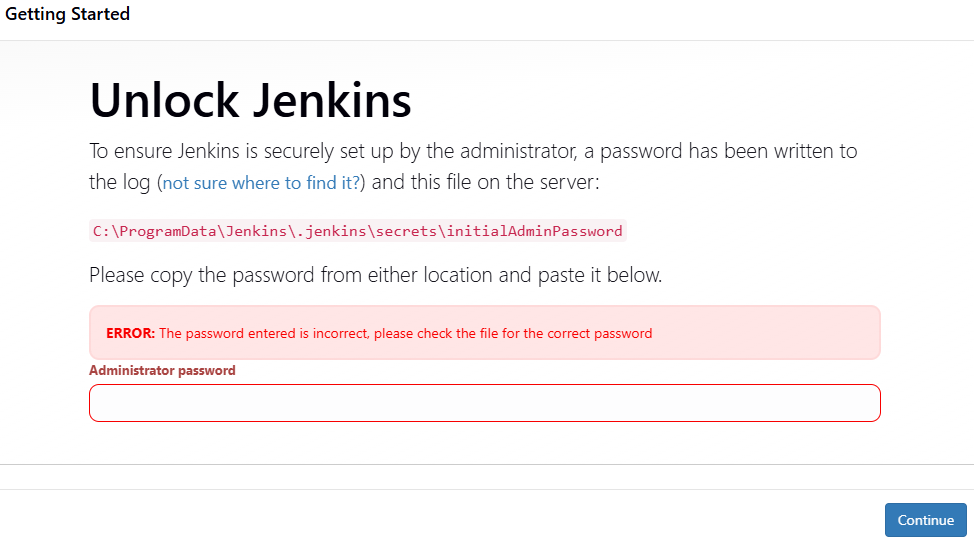
**CI/CD using Jenkins**

**JENKINS INSTALLATION (10 Mins) - Outdated**

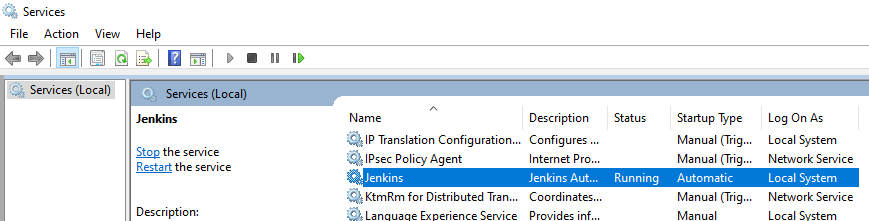
<https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable/>

Jenkins Installation Steps in Windows  
<https://www.jenkins.io/doc/book/installing/windows/>   
During installation, when you come across the following screen,   


Then, choose the highlighted option.

Open a browser and type <http://localhost:9080/>, you will get this screen.  
Go to C:\Program Files\Jenkins\jenkins.err and you should be able to see the password  
  
Jenkins Installation Steps in Linux  
<https://www.jenkins.io/doc/book/installing/linux/>

Stop/Start Jenkins using the Service Window



Access Jenkins Dashboard <http://localhost:9080/> (port could differ based on configuration)

**JENKINS INSTALLATION (10 Mins)**

Go to the Jenkins Installation Section in the following link.

<https://srinivas-mahakud.medium.com/sonarqube-and-jenkins-integration-in-windows-machine-65e86913e1d6>

**Jenkins and SonarQube Integration**

Follow the same link as mentioned above   
<https://srinivas-mahakud.medium.com/sonarqube-and-jenkins-integration-in-windows-machine-65e86913e1d6>

**SonarQube and Maven Integration (optional)**

1. Download SonarQube sonarqube.org
2. We need to have Java Installed as a Prerequisite
3. Download SonarQube(free version) zipped file and then, unzip in your system
4. Run SonarQube

D:\Softwares\sonarqube-25.4.0.105899\bin\windows-x86-64>StartSonar.bat

You should be able to see the following message

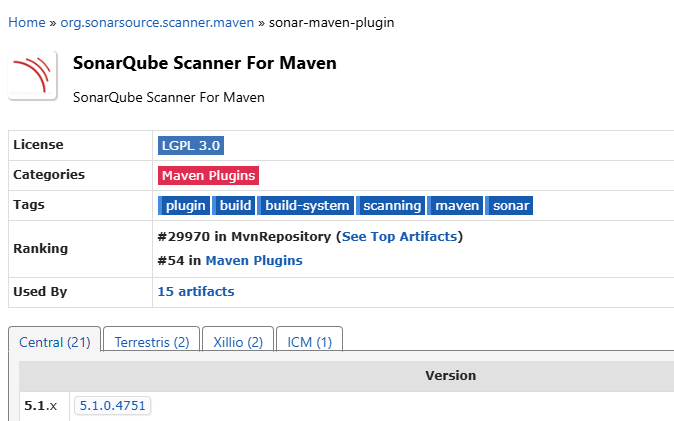


1. Access SonarQube from browser using <http://localhost:9000/> (login: admin/admin). Change the password after login for the first time
2. We need to integrate sonarqube into our maven POM.xml file, for code quality check.

Search for ‘maven sonarqube integration’ in google

<https://docs.sonarsource.com/sonarqube-server/10.4/analyzing-source-code/scanners/sonarscanner-for-maven/>

*Go to MVNRepository and we need to search for SonarQube Scanner for Maven*

*and we can find.* Add the following in pom.xml, after dependencies tag.

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.sonarsource.scanner.maven</groupId>

<artifactId>sonar-maven-plugin</artifactId>

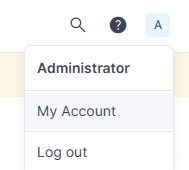
<version>5.1.0.4751</version>

</plugin>

</plugins>

</pluginManagement>

</build>

Go to sonarqube dashboard(localhost:9000)  


Go to security and generate a token, copy the token to your notepad

1. Open the command prompt, go to your project directory and the following command, this command has the token



*mvn sonar:sonar -Dsonar.host.url=http://localhost:9000 -Dsonar.token=squ\_3472525f5a6c94e062398cbe145c126081b05624*

1. Once the build complete, we can go to sonarqube dashboard and click on project and check the build details, if there are any code quality issues, then it will highlight

**Troubleshooting**

**#Issue1: Jenkins** **is failing to fetch the branch (likely master) from your GitHub repository. This happens if:**

1. **The default branch is not master**, and you haven’t explicitly specified the branch.
2. **The repository URL is correct, but there are no commits in the specified branch**, or the branch does not exist.
3. **You don’t have access permissions**, though in this case it's a public repo, so less likely.

**✅ Solution:**

Update your git step in the Fetch Code stage to explicitly specify the correct branch name. Your GitHub repo uses main, **not master**.

**🔧 Fix:**

Change this:

git 'https://github.com/mahakud1/rest-api-crud-example.git'

To:

git branch: 'main', url: 'https://github.com/mahakud1/rest-api-crud-example.git'

**#Issue2: SonarQube Not Starting**Open Task Manager → Look for java.exe or similar processes linked to Elasticsearch/SonarQube → End them if no SonarQube is running.  
  
Delete the file named: node.lock, under D:\Softwares\sonarqube-25.4.0.105899\data\es8\ and Start Sonar again   
  
Check the exact reason for Elasticsearch failure in:

* D:\Softwares\sonarqube-25.4.0.105899\logs\es.log
* D:\Softwares\sonarqube-25.4.0.105899\logs\sonar.log

Restart your machine and then Start SonarQube Server instance  
  
**#Issue3: Code not Building after changes in VSCode**make sure save the changes before running mvn clean package command

**#Issue4: ran the command in project directory, but the sonarqube scanning with maven is not happening properly and its throwing some issues**

mvn sonar:sonar -Dsonar.host.url=localhost:9000 -Dsonar.token=sqp\_dd15754adc4655d5f796388fa137e791c5476524

**#Issue5: Failed to query server version: GET http://localhost:9000/api/v2/analysis/version failed with HTTP 401.**

**Please check the property sonar.token or the environment variable SONAR\_TOKEN.**

**Ensure to add SONAR\_TOKEN as a variable name which needs to be mapped to Sonarqube Token, Jenkins look for this variable.**

**Workshop Part-3**

**CI/CD using Jenkins Pipeline**Go to Jenkins Dashboard => New Item => Select ‘Pipeline’ => Give a name to Pipeline

Click on ‘Configure’ on the left panel.  
Go to Pipeline Section, select ‘Pipeline Script’ from the drop down.

Paste the following groovy script.  
node {

def mvnHome

stage('Preparation') {

// Checkout the main branch from GitHub

checkout([$class: 'GitSCM',

branches: [[name: '\*/main']],

userRemoteConfigs: [[url: 'https://github.com/mahakud1/rest-api-crud-example.git']]

])

// tool 'maven' corresponds to the name what is being configured in Manage Jenkins -> Tools-> Maven

mvnHome = tool 'maven'

}

stage('Build') {

// Run the maven build

withEnv(["MVN\_HOME=$mvnHome"]) {

if (isUnix()) {

sh '"$MVN\_HOME/bin/mvn" -Dmaven.test.failure.ignore clean package'

} else {

bat(/"%MVN\_HOME%\bin\mvn" -Dmaven.test.failure.ignore clean package/)

}

}

}

stage('SonarQube Analysis') {

// Inject SONAR\_TOKEN securely

withCredentials([string(credentialsId: 'sonarqube', variable: 'SONAR\_TOKEN')]) {

withEnv(["MVN\_HOME=$mvnHome"]) {

if (isUnix()) {

sh '"$MVN\_HOME/bin/mvn" sonar:sonar -Dsonar.login=$SONAR\_TOKEN -Dsonar.host.url=http://localhost:9000'

} else {

bat "\"%MVN\_HOME%\\bin\\mvn\" sonar:sonar -Dsonar.login=%SONAR\_TOKEN% -Dsonar.host.url=http://localhost:9000"

}

}

}

}

stage('Results') {

junit '\*\*/target/surefire-reports/TEST-\*.xml'

archiveArtifacts 'target/\*.jar'

}

}

**Click Save and Apply, Then, Build the Pipeline and check the console**

**Deploy JAR to a Remote Server via SSH/SCP**

**Prerequisites**

1. Jenkins is installed and working
2. Your JAR file is built and available in the target/ folder
3. You have access to the remote server (IP, username, SSH key or password)
4. SSH is enabled on the remote server
5. Jenkins has the SSH Agent plugin or Publish Over SSH plugin installed

**Using Publish Over SSH (Freestyle Job)**

**Step 1**: Configure SSH Server in Jenkins

* Go to Manage Jenkins → Configure System
* Find the "Publish over SSH" section
* Add a new SSH server:
  + Name: RemoteServer
  + Hostname: your.server.ip
  + Username: deployuser
  + Remote Directory: /home/deployuser/app
  + Add SSH Key or Password
* Test connection

**Step 2**: Add Post-build Action in Job

* In your Jenkins Job:
  + Click Configure
  + Under "Post-build Actions" → "Send build artifacts over SSH"
  + Set:
    - Source files: target/\*.jar
    - Remove prefix: target
    - Remote directory: app
    - Exec command: (optional) java -jar app/rest-api-crud-example.jar