- **Git**

Step 1: Setting up Jenkins

1. **Download Jenkins:**

   - If you haven't installed Jenkins, download it from [Jenkins official website](https://www.jenkins.io/download/).

2. **Run Jenkins:**

   - Navigate to your downloads folder where `jenkins.war` is located and execute the following command:

   ```sh
   java -jar jenkins.war
   ```

3. **Access Jenkins UI:**

   - Open your browser and go to `http://localhost:8080`.

   - Enter the **initial admin password** found in the `jenkins.log` file.

   - Install the recommended plugins.

   - Create an admin user and complete the setup.

Step 2: Installing Required Plugins

To ensure Jenkins can work with **Docker and Maven**, follow these steps:

1. **Go to Jenkins Dashboard** → `Manage Jenkins` → `Manage Plugins`.

2. **Under the "Available" tab**, search for:

   - **Docker Pipeline Plugin**

   - **Maven Integration Plugin**

3. Install both plugins and restart Jenkins.

Step 3: Adding JDK and Maven in Jenkins

Jenkins requires **Java and Maven** to build the project.

1. **Go to Jenkins Dashboard** → `Manage Jenkins` → `Global Tool Configuration`.
2. **Add JDK:**
   - Scroll to **JDK** → Click `Add JDK`.
   - Uncheck "Install Automatically."
   - Set the **JDK Name** and specify the Java installation path (e.g., `/usr/lib/jvm/java-11-openjdk`).
3. **Add Maven:**
   - Scroll to **Maven** → Click `Add Maven`.
   - Uncheck "Install Automatically."
   - Set the **Maven Name** and specify the installation path (e.g., `/usr/share/maven`).

Click **Save**.

Step 4: Creating a New Jenkins Job

1. **Go to Jenkins Dashboard** → Click on `"New Item"`.
2. Enter a **name** for your job.
3. Select **"Pipeline"** and click `"OK"`.
4. Scroll down to **Pipeline Definition**.
5. Choose **"Pipeline script"** and enter the following script.

Step 5: Adding the Pipeline Script

Copy and paste the following **Groovy** script into the pipeline configuration.

```groovy
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                // Clone the GitHub repository
                git branch: 'main', url: 'https://github.com/sonam-niit/springproject.git'

                // Run Maven Build
                bat "./mvnw compile"

                echo 'Building the Project with Maven'
            }
        }

        stage('Test') {
            steps {
                // Run Tests
                bat "./mvnw test"

                echo 'Testing the Project with Maven'
            }
        }

        stage('Package') {
            steps {
                // Package the application
                bat "./mvnw package"
```

```
            echo 'Packaging the Project'
        }
    }

    stage('Containerize') {
        steps {
            // Build the Docker image
            bat "docker build -t myapp ."

            echo 'Containerizing the Application'
        }
    }

    stage('Deploy') {
        steps {
            script {
                // Check if the container is running
                def containerRunning = bat(script: 'docker ps -q -f name=sbapp', returnStdout:
true).trim()
                if (containerRunning.isInteger()) {
                    // Stop and remove the container
                    bat "docker stop sbapp"
                    bat "docker rm sbapp"
                }
            }

            // Run the new Docker container
            bat "docker run -d --name sbapp -p 9092:8082 myapp"

            echo 'Deploying the Application'
        }
```

```
        }
    }
}
```

Explanation of the Pipeline Stages:

1. **Build**: Clones the GitHub repository and compiles the code.
2. **Test**: Runs unit tests.
3. **Package**: Packages the Spring Boot application into a `.jar` file.
4. **Containerize**: Builds a Docker image.
5. **Deploy**: Stops any running container, removes it, and starts a new one.

Click **Save**.

Step 6: Running the Jenkins Job

1. **Click on "Build Now"** to start the job.
2. Monitor the build process in the **Console Output**.
3. Once the build is successful, check if the **Docker container is running**:

```sh
docker ps
```

Step 7: Deploying the Application

1. Open a browser and navigate to:

```
```

http://localhost:9092/api/product/5678

```

2. If the application is running successfully, you should see the expected output.

---

## **Step 8: Automating Deployment on Git Changes**

Jenkins can automatically detect changes in the Git repository and trigger a build.

1. **Go to Jenkins Dashboard** → Click on your Job.
2. Click **"Configure"**.
3. Under **"Build Triggers"**, check **"Poll SCM"**.
4. Enter the schedule:

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

This tells Jenkins to check for updates every **5 minutes**.

5. Click **Save**.

Now, whenever you push changes to GitHub, Jenkins will **automatically build and deploy** the new version.
```