JENKINS

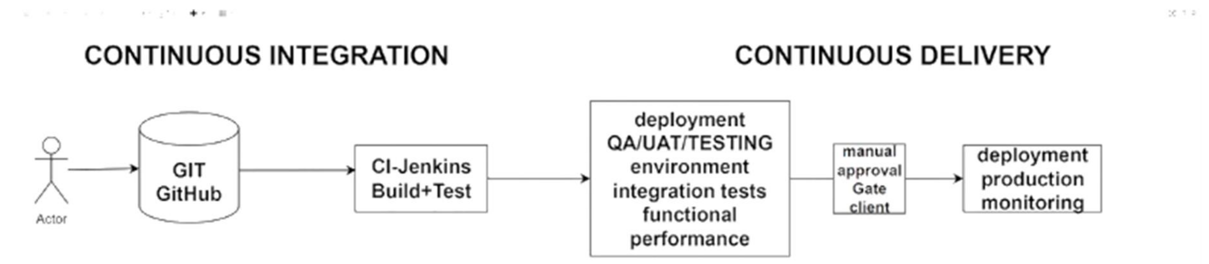
Jenkins is an open-source automation server that originated as a fork of the Hudson project in 2011. It quickly gained popularity due to its flexibility, extensive plugin ecosystem, and ability to automate a range of software development tasks.

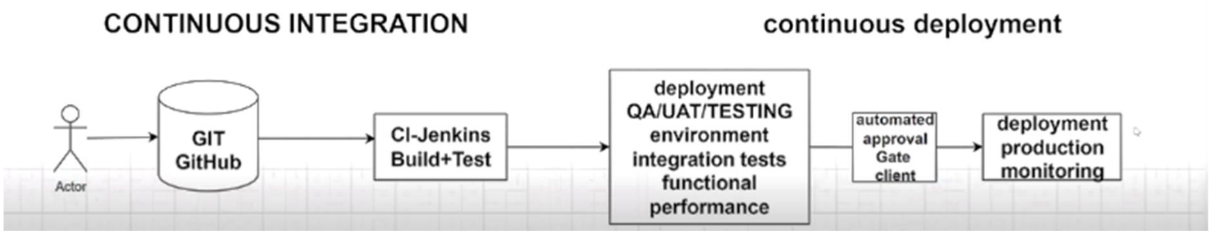
Continuous Integration (CI):

- **Build Automation**: Jenkins automates building software whenever code changes occur, allowing developers to catch issues early.
- **Automated Testing**: Jenkins runs automated tests as part of CI, providing immediate feedback on code quality.
- **Code Quality Analysis**: Integrates with tools to ensure code meets quality standards.
- **Notification and Reporting**: Jenkins notifies developers of build outcomes and provides detailed reports.

Continuous Delivery (CD):

- **Automated Deployments**: Jenkins supports deploying code to various environments after testing.
- **Pipelines**: Jenkins pipelines allow defining complex deployment workflows, including approvals and rollbacks.
- **Integration with Deployment Tools**: Jenkins works with Docker, Kubernetes, and cloud providers to automate deployments.
- **Artifact Management**: Jenkins handles versioned storage and management of built artifacts.


CONTINUOUS INTEGRATION
CONTINUOUS DELIVERY
Actor
GIT GitHub
CI-Jenkins Build+Test
deployment QA/UAT/TESTING environment integration tests functional performance
manual approval Gate client
deployment production monitoring


CONTINUOUS INTEGRATION
continuous deployment
Actor
GIT GitHub
CI-Jenkins Build+Test
deployment QA/UAT/TESTING environment integration tests functional performance
automated approval Gate client
deployment production monitoring

Installation of Jenkins

Jenkins can be installed in multiple platforms such as:

- Linux – Hosted by AWS
- Windows – Local system
- MacOS - Local system
- Solaris

Jenkins installation steps can be gotten from this official Jenkins documentation link https://www.jenkins.io/doc/book/installing/

Jenkins Editions:

- Community Edition = CE – Free edition
- Enterprise Edition = EE - CloudBees Jenkins

Jenkins Installation (CE) And Setup in AWS EC2 Redhat Instnace.

Prerequisite

- AWS Account.
- Create Redhat EC2 t2.medium Instance with 4GB RAM.
- Create Security Group and open Required ports. 8080 Jenkins port, ..etc
- Attach Security Group to EC2 Instance.

➢ Install java openJDK 1.8+ for SonarQube version 7.8

➢ Install Java JDK 1.8+ as Jenkins pre-requisite

➢ Install other softwares - git, unzip and wget

```
sudo hostnamectl set-hostname jenkins
sudo yum upgrade
sudo yum  install unzip wget tree git -y
sudo yum install java-11-openjdk -y
```

➢ Add Jenkins Repository and key

```
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
```

➢ Install Jenkins

```
sudo yum -y install jenkins  --nobest
```

➢ Start Jenkins service and verify Jenkins is running

```
sudo systemctl daemon-reload
sudo systemctl start jenkins
sudo systemctl enable jenkins
sudo systemctl status jenkins
```

- Access Jenkins from the browser

```
public-ip:8080
curl ifconfig.co
```

- Get jenkins initial admin password

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

OR

Install Jenkins using the script

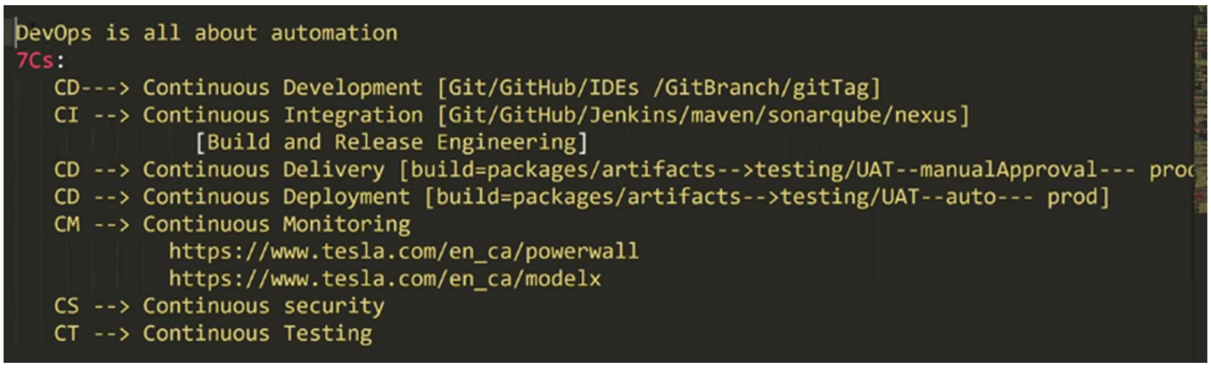
```
#!/bin/bash
# Installing Jenkins on RHEL 7/8, CentOS 7/8 or Amazon Linux OS
# You can execute this script as user-data when launching your
EC2 VM.
sudo timedatectl set-timezone America/New_York
sudo hostnamectl set-hostname Jenkins
sudo yum install wget -y
sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
sudo      rpm      --import      https://pkg.jenkins.io/redhat-
stable/jenkins.io-2023.key
sudo yum upgrade
```

```
# Add required dependencies for the jenkins package
sudo yum install java-11-openjdk -y
sudo yum -y install jenkins  --nobest
sudo systemctl daemon-reload
sudo systemctl enable jenkins
sudo systemctl start jenkins
sudo systemctl status jenkins
echo "End of jenkins installation"
sudo su - ec2-user
```

How is Jenkins managed?

- Jenkins is managed by a Jenkins user created by default during installation.
- This is for security measure.
- Jenkins shouldn't be managed as root user

NB: DevOps is all about automation and Jenkins help in achieving this automation through 7C:



```
DevOps is all about automation
7Cs:
   CD---> Continuous Development [Git/GitHub/IDEs /GitBranch/gitTag]
   CI --> Continuous Integration [Git/GitHub/Jenkins/maven/sonarqube/nexus]
              [Build and Release Engineering]
   CD --> Continuous Delivery [build=packages/artifacts-->testing/UAT--manualApproval--- proc
   CD --> Continuous Deployment [build=packages/artifacts-->testing/UAT--auto--- prod]
   CM --> Continuous Monitoring
            https://www.tesla.com/en_ca/powerwall
            https://www.tesla.com/en_ca/modelx
   CS --> Continuous security
   CT --> Continuous Testing
```

Project: Managing a client application called Tesla using Jenkins

- ✓ It's a java web application
- ✓ The application has been created and shared in a Github repository by developers
- ✓ Developers continuous commit/push codes to the project repository in Github using git branching strategy.

Jenkins-Github integration:

To integrate your GitHub repository with Jenkins and set up a Freestyle project that pulls the repository, while also adding the Jenkins user to the **sudoers** file so Jenkins can perform tasks requiring elevated privileges, follow these steps:

1. **Set Up GitHub Integration**: Make sure Jenkins has the GitHub plugin installed. If not, install it from the Jenkins Plugin Manager.
2. **Create a GitHub Personal Access Token**:
   - Go to GitHub, and log in to your account.
   - Navigate to **Settings > Developer settings > Personal access tokens**.
   - Click **Generate new token**.
   - Provide a name for the token and select the scopes you need (e.g., **repo** for full repository access).
   - Once created, copy the token as you'll need it for authentication in Jenkins
3. **Configure GitHub Credentials in Jenkins**:
   - In Jenkins, go to **Manage Jenkins > Manage Credentials**.
   - In the left menu, click **Jenkins** (or another appropriate domain if you have one).
   - Click **Add Credentials**.
   - For **Kind**, select **Username and Password**
   - In the **Secret** field, paste the GitHub personal access token you copied earlier.
   - Provide a **Description** and **ID** for your credentials.
   - Click **OK** to save the credentials.
4. Add Jenkins User to the sudoers File:
   - Use visudo to safely edit the sudoers file:

```
sudo visudo
```

- Add the line to allow the Jenkins user (jenkins) to run commands without a password prompt:

```
jenkins ALL=(ALL) NOPASSWD: ALL
```

- Save the file and exit the editor.
- Restart Jenkins to apply the changes:

```
sudo systemctl restart jenkins
```

5. **Create a Freestyle Project**:
   - In Jenkins, go to **New Item**.
   - Give your project a name (tesla) and choose **Freestyle project**, then click **OK**.
   - **Configure the Source Code Management**:
   - In the project configuration, find the section **Source Code Management**.
   - Select **Git**.
   - Enter the repository URL (e.g., **https://github.com/fewaitconsulting/maven-web-app**).
   - In the credential's dropdown, choose the GitHub credentials you added earlier.
   - Specify the branch to build, if necessary (default is **master** now called **main**).
6. Save the Project:
   - Once you have configured your project, click **Save**.
   - Test the Configuration:
7. Manually trigger the build (click Build Now) to test the configuration.
8. Verify that Jenkins pulls the repository and completes the build successfully.

   Once your freestyle project is set up and running, Jenkins will pull the Tesla repository from GitHub.

Jenkins Maven Integration (To build the code)

1. **Ensure Maven Is Installed on the Jenkins Server**:
    - Go to Manage Jenkins > Global Tool Configuration and install maven.
2. **Configure Maven in Jenkins**:
    - In Jenkins, go to **Manage Jenkins > Global Tool Configuration**.
    - Scroll down to **Maven installations** and add a new Maven installation if not already configured.
    - Provide a name for the installation and specify the Maven home directory.
    - If Jenkins will be downloading Maven automatically, check the box to **Install automatically** and specify the desired version.
    - Save the configuration.
3. **Configure Your Freestyle Project**:
    - Open the Freestyle project you created earlier for your Tesla app.
    - In the **Build** section, add a build step of type **Invoke top-level Maven targets**.
    - Specify the **Maven installation** you configured earlier.
    - In the **Goals** field, enter the Maven goals you want to execute. For example, **package** for a typical build and test process.
4. **Save the Project**:
    - Save the changes to your project configuration.
5. **Test the Configuration**:
    - Manually trigger a build to test the configuration.
    - Jenkins will use Maven to build your Tesla app according to the specified goals.

Jenkins – Sonarqube integration (For code quality analysis)

- Open port 9000 in the security group for sonarqube (if not yet done so).
- Make sure sonarqube server is running
- Switch to sonar user

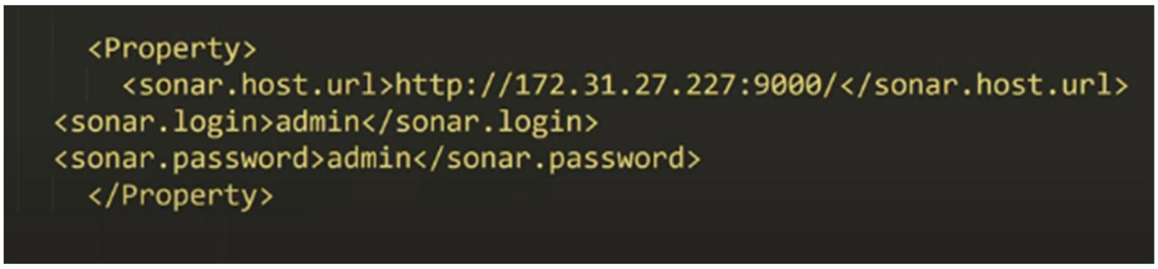
```
sudo su – sonar
```

- Sonarqube service must be running

```
sh /opt/sonarqube/bin/linux-x86-64/sonar.sh start
sh /opt/sonarqube/bin/linux-x86-64/sonar.sh status
```

- In the “property” tag in the project file (pom.xml), add sonarqube login credentials

NB: Check ip of the server and if the two servers (Jenkins and sonarqube) are in the same environment, use private ip address instead of public



```
    <Property>
      <sonar.host.url>http://172.31.27.227:9000/</sonar.host.url>
  <sonar.login>admin</sonar.login>
  <sonar.password>admin</sonar.password>
    </Property>
```

Configure Your Freestyle Project:

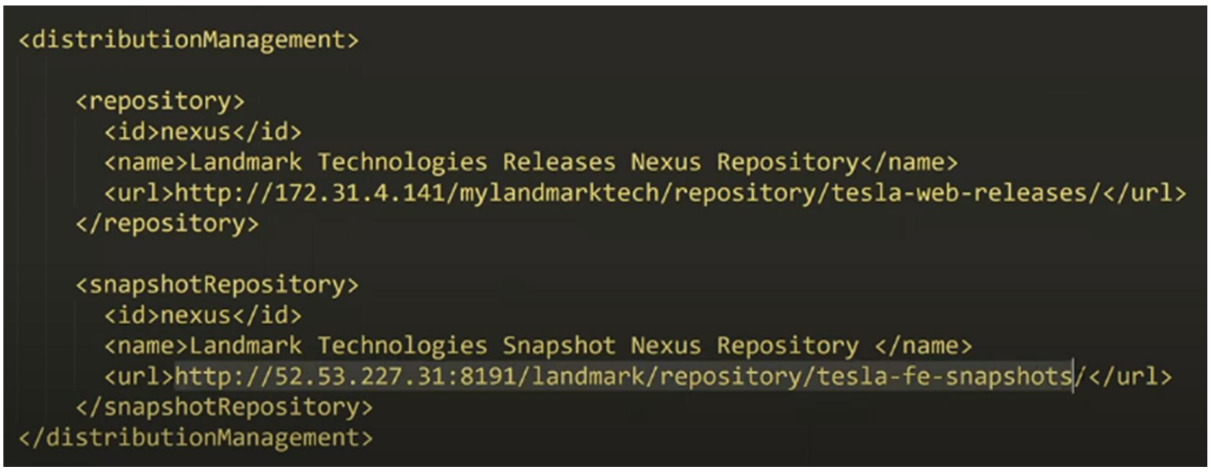
- Open the Freestyle project you created earlier for your Tesla app.
- In the Build section, add a build step of type Invoke top-level Maven targets.
- Specify the Maven installation you configured earlier.
- In the Goals field, enter the Maven goals you want to execute.

```
sonar:sonar
```

- Save the changes to your project configuration.
- Manually trigger a build to test the configuration.

Jenkins Nexus Integration (For uploading of built artifacts)

- Open required port 8081 in the security group for nexus.
- Create repos in nexus-UI to upload artifacts i.e. releases and snapshots repositories.
- Modify “distributionManagement” tag with nexus repos details in the pom.xml



```
<distributionManagement>

    <repository>
      <id>nexus</id>
      <name>Landmark Technologies Releases Nexus Repository</name>
      <url>http://172.31.4.141/mylandmarktech/repository/tesla-web-releases/</url>
    </repository>

    <snapshotRepository>
      <id>nexus</id>
      <name>Landmark Technologies Snapshot Nexus Repository </name>
      <url>http://52.53.227.31:8191/landmark/repository/tesla-fe-snapshots/</url>
    </snapshotRepository>
</distributionManagement>
```

Modify /maven/config/setting.xml in maven (Jenkins) server with login credential inside the <servers> tags.

```
sudo vi tools/hudson.tasks.Maven_MavenInstallation/maven3.8.6/conf/settings.xml
```

```
    <server>
      <id>nexus</id>
      <username>your-username</username>
      <password>your-password</password>
    </server>
```

- Configure Your Freestyle Project:
  - Open the Freestyle project you created earlier for your Tesla app.
  - In the Build section, add a build step of type Invoke top-level Maven targets.
  - Specify the Maven installation you configured earlier.

- In the Goals field, enter the Maven goals you want to execute.

```
deploy
```

- Save the changes to your project configuration.
- Manually trigger a build to test the configuration.

Jenkins - Tomcat Integration (For deployment of the java web application)

- Since both Tomcat and Jenkins use port 8080, we will change that of the tomcat to port 8177. This is done in the **conf/server.xml** file in the tomcat server.

```
<Connector port="8177" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

- Install "Deploy to container" plugin in the Jenkins UI

This plugin allows the deployment of the .war file to container after a successful build.

- Add a Tomcat user /tomcat9/conf/tomcat-users.xml

```
sudo vi /tomcat/conf/tomcat-users.xml
```

```
<user username="" password="" roles="manager-gui,admin-gui,manager-script"/>
```

- **Add Deploy to Container Build Step**:
  - Search and Install the **Deploy to container** plugin.
  - Click on **Post Build Actions** and select **Deploy war/ear to a container**.
  - Configure the build step with the necessary parameters:

    - **WAR/EAR files**: Specify the path to the WAR or EAR file to deploy. E.g. target/*war
    - **Context path**: Specify the context path under which the application should be deployed (e.g., **/tesla**) or allow it empty.
    - **Containers**: Select **Tomcat** with the correct version.
    - **Tomcat URL**: Enter the Tomcat server URL (e.g., **http://localhost:8177**).
    - **Credentials**: Select the credentials for authenticating with the Tomcat server.
    - **Tomcat version**: Select the Tomcat server configuration you added earlier.
- Click **Save** to apply the changes to the Jenkins job.
- Manually trigger a build of the Jenkins job you configured for deploying the web application.