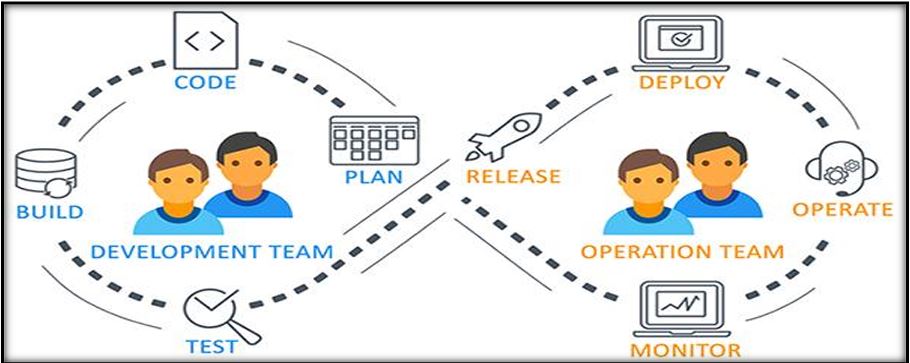
**DEVSECOPS CICD SETUP**

# 1. What is DevOps?

DevOps, when explained in simple terms can be stated as combining the engineers team and the operations team to work together in an automated environment and also in a repeatable way. Working this way will help in getting the things done faster. Development and operation teams are entirely two different teams and it has been always impossible for them to work together. DevOps is a transformative set of practices designed to automate and integrate workflows between software development and IT operations teams. The primary goal is to accelerate the build, test, and release process, ensuring that software is delivered faster and more reliably.

A key visual representation of DevOps is the infinity loop, illustrating the continuous and interconnected nature of its lifecycle phases. While the phases may seem sequential, the loop highlights the constant need for collaboration, iteration, and improvement at every stage of the process.



## 1.1 Purpose

The success in agile software development strongly depends on the team’s ability to quickly roll out features to the end users and continuously improve the software incorporating the feedback from the end users. Shorter the cycle, better the user satisfaction would be. An efficient CI-CD would be the key to achieving quick turnarounds.

The purpose of this document is to provide all the necessary details of the DevOps processes to get an understanding and to get trained on them to carry out day to day operations.

## 1.2 Intended Audience

The Intended audience for this document is as follows:

* Business Process Owners of C-DAC Noida
* TL/PL/GC/HOD of C-DAC Noida

# 2. DevOps Objectives

The primary objectives of DevOps are as follows –

1. **Foster a Collaborative Environment**

The main theory behind DevOps is to combine development and operations to create a unilateral team that focuses on delivering common objectives.

1. **Impose End-to-End Responsibility**

In the traditional software development model, developers and operations had separate roles. But in DevOps, both groups work as a team that is fully accountable for the application from beginning to end. One of the core principles of DevOps is the control and responsibility of services from "concept to grave."

1. **Encourage Continuous Improvement**

Teams must continuously adapt to changing circumstances, whether that may be the emergence of new technology, customer needs, or changes in rules and regulations. DevOps places a strong focus on continuous improvement to optimize performance, cost and speed of delivery.

1. **Automate (Almost) Everything**

To strive for continuous improvement with high cycle rates and the ability to immediately respond to customer feedback, teams must utilize automated processes.

1. **Focus on Customer Needs**

DevOps requires teams to act like a lean startup that can innovate continuously, pivot when a strategy is no longer working, and invest in features to deliver customer satisfaction. DevOps teams must have one finger on the pulse to consistently meet the needs of ever-changing consumer demands. The data gathered from the automated processes must be constantly reviewed to ensure performance targets are met.

1. **Embrace Failure and Learn from it**

To fully embrace DevOps, team must change their attitude towards failure. By accepting failure, they can foster a “climate for learning” which will positively impact organizational culture.

1. **Unite Teams – and Expertise**

DevOps teams are required to be involved at every stage of the software development lifecycle, from planning, building, deployment, feedback and improvement. This requires a cross-functional team where each member is well-rounded and has a balanced set of skills.

# 3. DevOps Lifecycle

The DevOps lifecycle consists of following phases

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Phase** | **Description** |
| 1 | Plan | Use agile practices to break work into smaller, manageable tasks |
| 2 | Build | Version control and automation for the development build process |
| 3 | Continuous Integration and Deployment | Merge code changes frequently into the main branch |
| 4 | Monitor and Alert | Monitor the development, integration, deployment, and operations process |
| 5 | Operate | Maintain, monitor, and troubleshoot applications in production |
| 6 | Continuous Feedback | Evaluate each release and generate reports to improve future releases |

# 4. CI-CD Approach



**CI/CD Approaches: Streamlining Software Delivery**

Continuous Integration and Continuous Delivery/Deployment (CI/CD) are foundational practices in DevOps that automate and streamline the software development lifecycle. The following approaches highlight how CI/CD can be implemented to enhance efficiency, quality, and speed in software delivery:

**4.1. Continuous Integration (CI):**

* **Definition**: CI focuses on integrating code changes from multiple developers into a shared repository several times a day.
* **Key Practices**:
  + Automated builds and tests are triggered with every code commit.
  + Early detection of integration issues, minimizing "integration hell."
  + Use of version control systems like Git and CI tools like Jenkins, Travis CI, or GitHub Actions.
* **Benefits**: Improved code quality, reduced integration risks, and faster feedback for developers.

**4.2. Continuous Delivery (CD):**

* **Definition**: Continuous Delivery extends CI by automating the release process, ensuring that software is always in a deployable state.
* **Key Practices**:
  + Deployments are prepared for staging environments after passing automated tests.
  + Manual approval can be incorporated for production releases.
  + Emphasis on automation tools like Azure DevOps, Spinnaker, or GitLab CI/CD.
* **Benefits**: Reliable and repeatable deployment processes, reduced human error, and quicker time-to-market.

**4.3. Continuous Deployment:**

* **Definition**: Continuous Deployment takes Continuous Delivery a step further by automating the entire release process, deploying changes to production without manual intervention.
* **Key Practices**:
  + Every successful build and test in the pipeline is automatically deployed to production.
  + Strong emphasis on comprehensive testing and monitoring to ensure stability.
  + Use of container orchestration tools like Kubernetes or ECS for seamless deployments.
* **Benefits**: Faster updates to users, reduced lead time for fixes and features, and a fully automated delivery pipeline.

**4.4. Choosing the Right CI/CD Approach:**

The choice between Continuous Delivery and Continuous Deployment depends on the organization’s risk tolerance, regulatory requirements, and customer expectations. For example:

* **Continuous Delivery** suits organizations requiring manual approval or working in highly regulated industries.
* **Continuous Deployment** is ideal for companies prioritizing speed and operating in competitive, fast-moving markets.

**4.5. Tools Supporting CI/CD Approaches:**

1. **Version Control Tools**: Github, Gitlab
2. **CI Tools:** Jenkins
3. **Build Tools**: Maven, Gradle
4. **Artifact Tool**: Nexus
5. **Code Analysis Tools:** Sonarqube
6. **Container Tools:** Docker
7. **Image Registry:** Harbor
8. **Deployment Tools**: Docker, Kubernetes, Helm.
9. **Cluster Management**: Rancher
10. **Monitoring**: Prometheus & Grafana

# 5. Benefits Of Implementing DevOps Process

Implementing DevOps offers numerous advantages for organizations aiming to improve their software development and delivery processes. Here are some key benefits:

**1. Faster Delivery**

* DevOps speeds up software releases by automating processes like code integration, testing, and deployment, enabling teams to deliver updates and new features more quickly.
* Quick feedback loops help identify and fix issues faster.

**2. Better Team Collaboration**

* DevOps promotes closer collaboration between development and operations teams, breaking down barriers and improving communication.
* Everyone works together with shared goals, increasing accountability.

**3. Higher Software Quality**

* Automated testing ensures bugs are caught early, improving the reliability of the software.
* Monitoring tools help detect and fix issues before they affect users.

**4. Improved Efficiency**

* Automation reduces repetitive tasks, saving time and minimizing errors.
* Resources are used more effectively, especially with practices like containerization.

**5. Greater Reliability**

* Consistent deployment methods ensure software runs smoothly across different environments.
* Monitoring and quick responses to incidents minimize downtime and disruptions.

**6. Scalability**

* DevOps supports scaling infrastructure and applications to handle changing demands easily.
* Integration with cloud platforms makes scaling faster and more efficient.

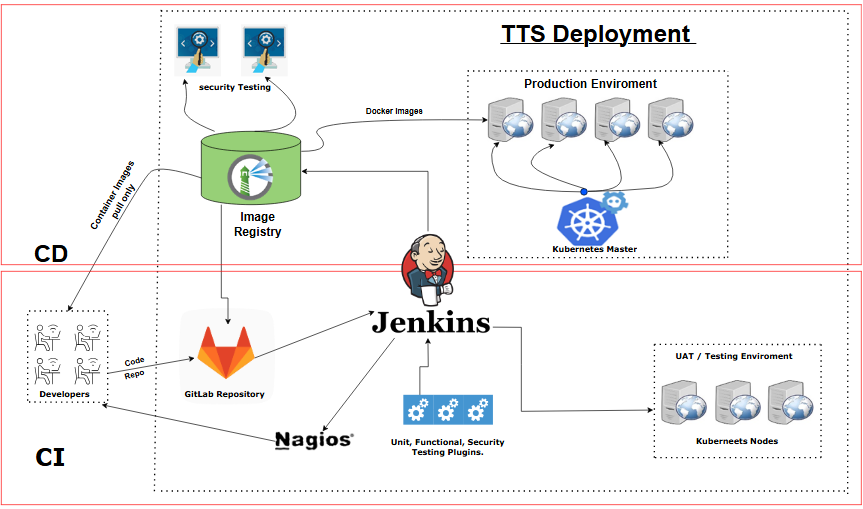
**7. Cost Savings**

* Automation reduces manual effort, optimizing resources and lowering costs.
* Early issue detection prevents expensive failures and downtime.

**8. Enhanced Security**

* Security checks are built into the development process to catch vulnerabilities early.
* Automated tools enforce consistent security and compliance practices.

# 6. CI-CD Pipeline flow



# 1. Setup of CI Server

**1. Jenkins**

Jenkins is an open source automation server. It helps automate the parts of software development related to building, testing, and deploying, facilitating continuous integration, and continuous delivery.

**Java Installation**

#sudo apt update -y

#sudo apt install openjdk-17-jdk

#java --version

**Set the Java Path**

#vim .profile

JAVA\_HOME=/usr/lib/jvm/java-17-openjdk-amd64

PATH=$PATH:$JAVA\_HOME:$HOME/bin

**Jenkins Installation**

#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-get update

#sudo apt-get install jenkins -y

(Here we install latest Jenkins version 2.48)

**Firewall Configuration**

#sudo systemctl start ufw

#sudo systemctl enable ufw

#sudo ufw enable

**Open the necessary ports**

#sudo ufw allow 8080/tcp

#sudo ufw allow http

#sudo ufw reload

**Start Jenkins**

#sudo systemctl status jenkins.service

#sudo systemctl restart jenkins.service

Setup Jenkins to start at boot,

#sudo systemctl enable jenkins.service

**Access the Initial Root Password**

Open the file containing the initial root password:

#sudo vim /var/jenkins\_home/secrets/initialAdminPassword

Note the password provided in the file. This password is generated during installation and is needed for the first login.

**Accessing Jenkins**

By default jenkins runs at port 8080, You can access jenkins at

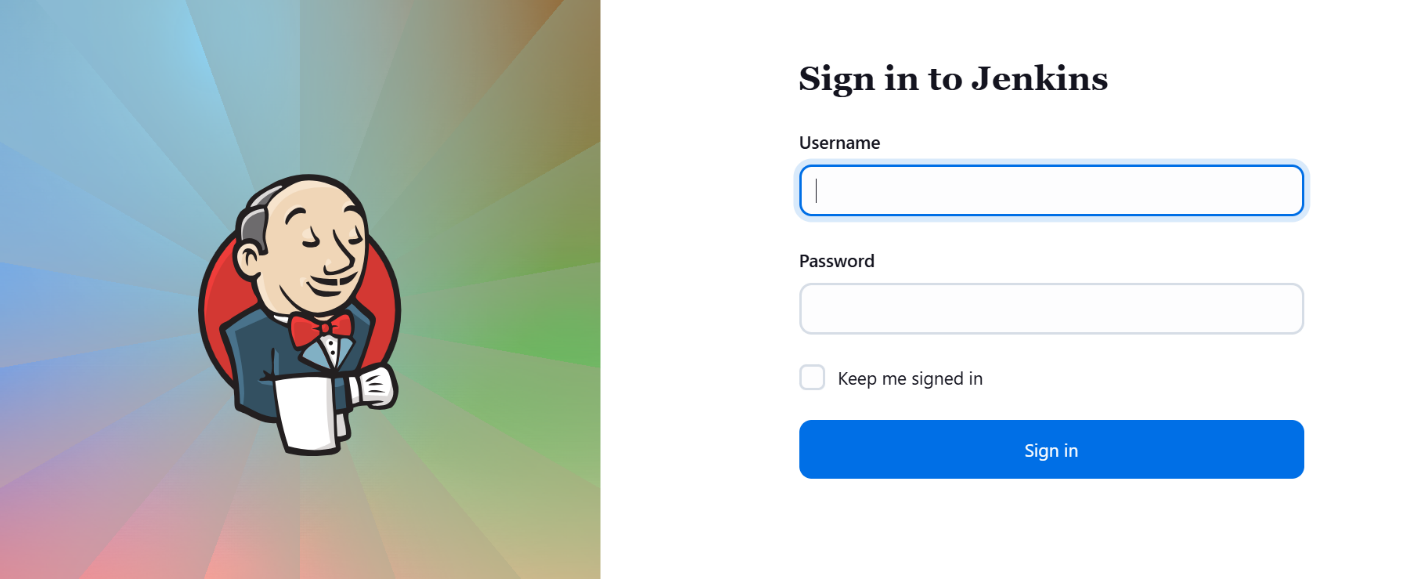
http://YOUR-SERVER-PUBLIC-IP:8080

Log in using the following credentials:

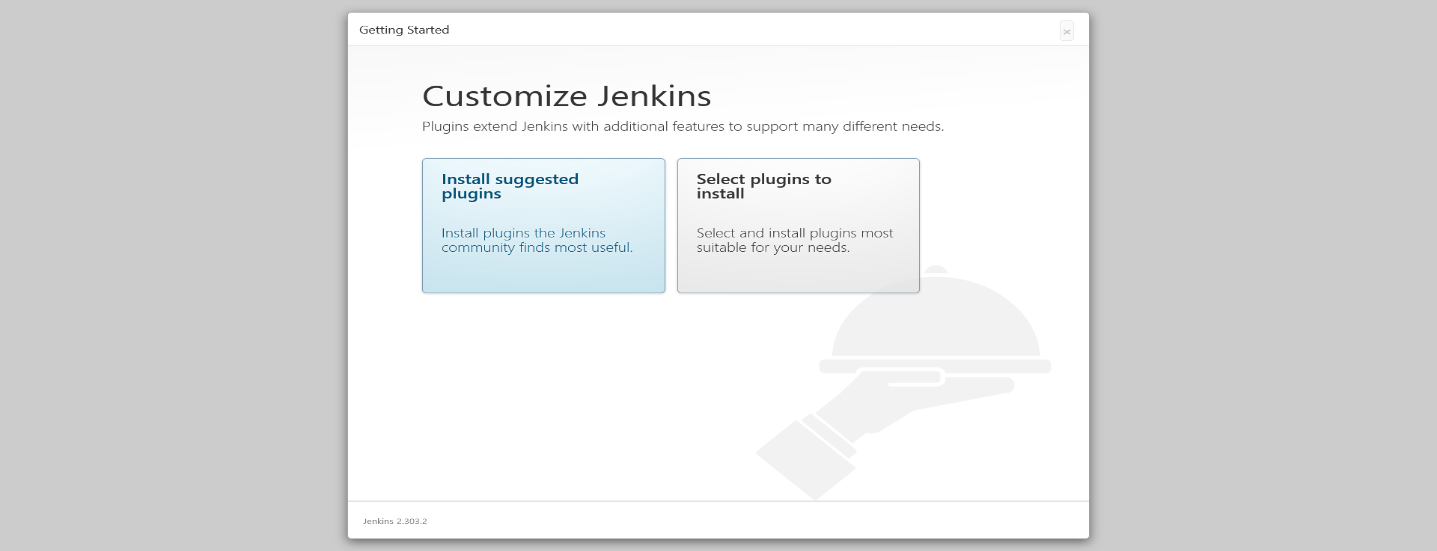
Username: admin

Password: Use the password obtained from the initial\_admin\_password file

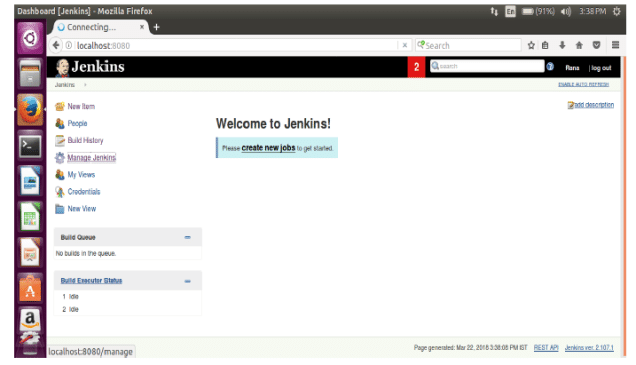
After logging in, update the root password for security.



The next screen presents the option of installing suggested plugins or selecting specific plugins:

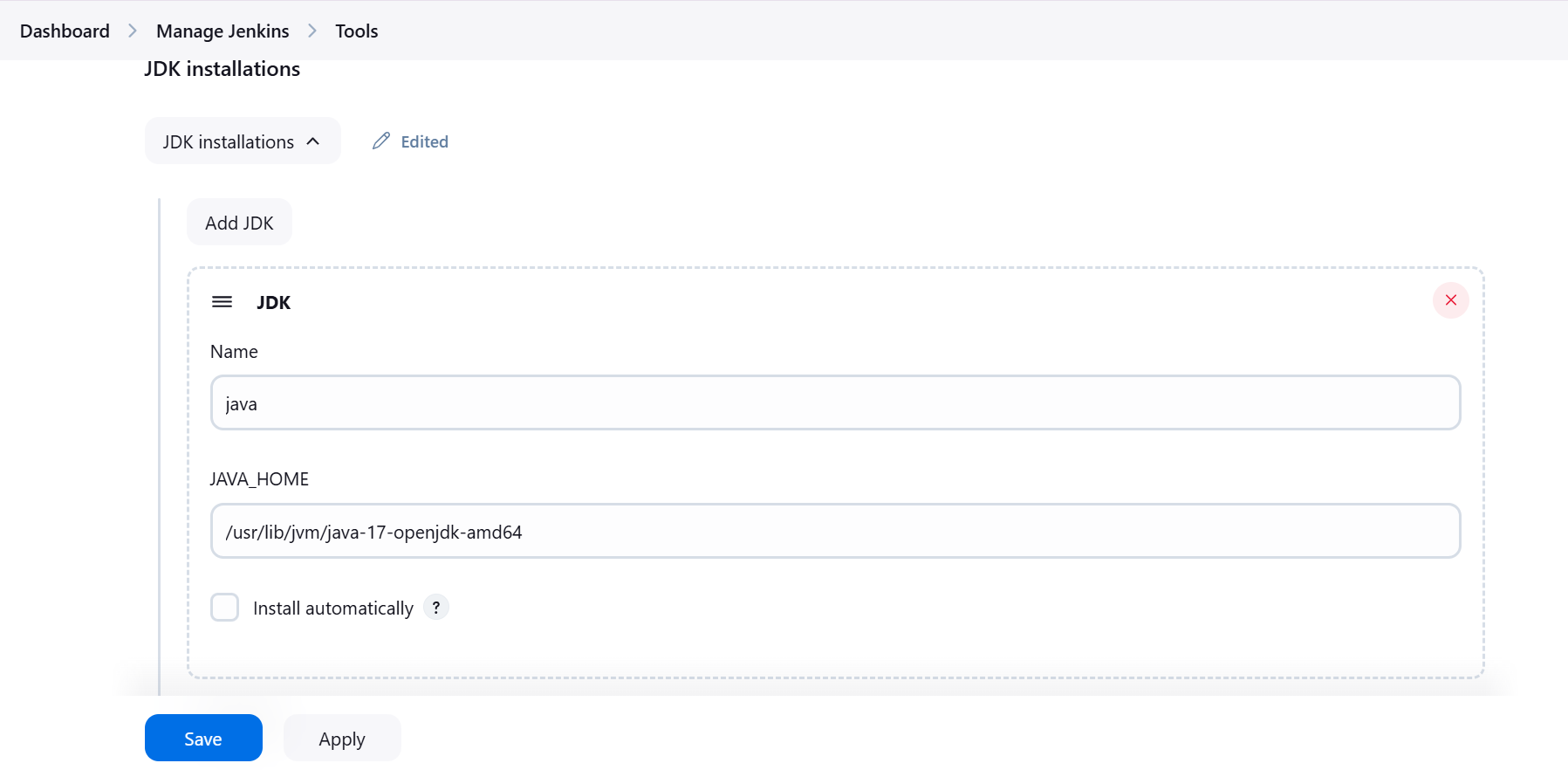


After confirming the appropriate information, click **Save and Finish**. You’ll receive a confirmation page confirming that **“Jenkins is Ready!”:**



Configure Jenkins

* The default Username is admin
* Grab the default password
* Password Location: /var/lib/jenkins/secrets/initialAdminPassword
* Skip Plugin Installation; We can do it later
* Change admin password
  + Admin  > Configure > Password
* Configure java path
  + Manage Jenkins > Tool > JDK



Test Jenkins Jobs

1. Create “new item”
2. Enter an item name – My-First-Project
   * Chose Freestyle project
3. Under the Build section Execute shell: echo "Welcome to Jenkins Demo"
4. Save your job
5. Build job
6. Check "console output"

**2. GitLab**

**Ensure your system is up-to-date:**

#sudo apt update && sudo apt upgrade -y

**Install the required dependencies:**

#sudo apt install ca-certificates curl openssh-server postfix tzdata -y

During the installation of postfix, you may be prompted to select a mail server configuration type. Choose "Internet Site" and set your domain if required.

**Step-by-Step Installation**

**Navigate to the temporary directory:**

#cd /tmp

**List the contents of the directory to verify access:**

#ls

**Download the GitLab installation script:**

#curl -LO https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh

Optionally, review the contents of the script:

#less /tmp/script.deb.sh

**Run the installation script:**

#bash /tmp/script.deb.sh

**Install the GitLab Community Edition:**

#sudo apt install gitlab-ce -y

**Firewall Configuration**

#ufw enable

**If ufw is active, allow necessary ports:**

#sudo ufw allow http

#sudo ufw allow https

#sudo ufw allow OpenSSH 3

**Verify the updated firewall status:**

#sudo ufw status

**Configure GitLab**

#sudo nano /etc/gitlab/gitlab.rb

Configure the external URL and other settings as required. Save the file (Ctrl+O followed by Enter) and exit (Ctrl+X).

**Reconfigure GitLab to apply changes:**

#sudo gitlab-ctl reconfigure

**Access the Initial Root Password**

Open the file containing the initial root password:

#sudo nano /etc/gitlab/initial\_root\_password

Note the password provided in the file. This password is generated during installation and is needed for the first login.

**Logging In**

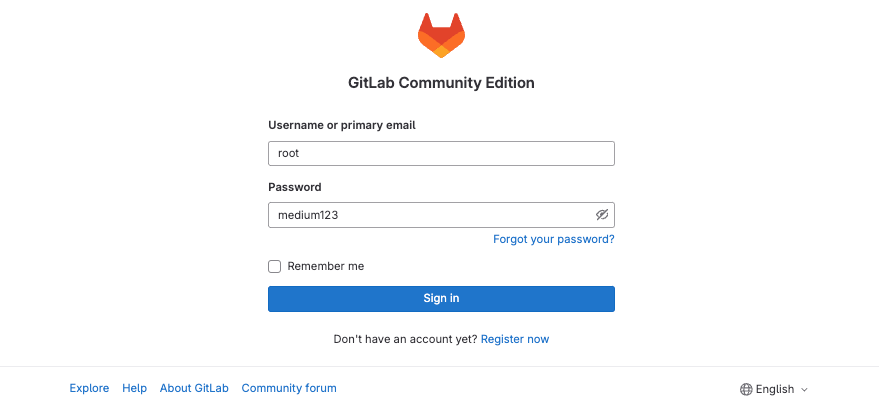
Access GitLab through your browser by navigating to the external URL (e.g., http://).

Log in using the following credentials:

Username: root

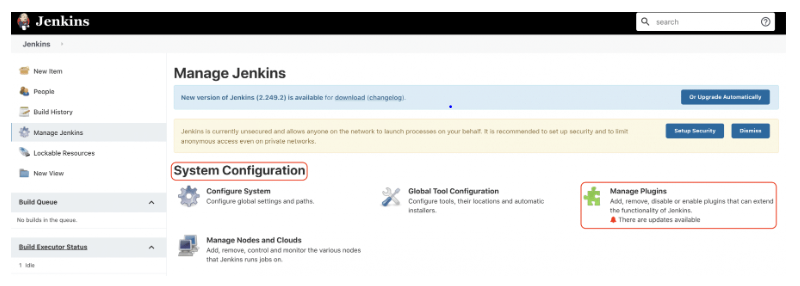
Password: Use the password obtained from the initial\_root\_password file

After logging in, update the root password for security.



**Integrate Gitlab with jenkins**

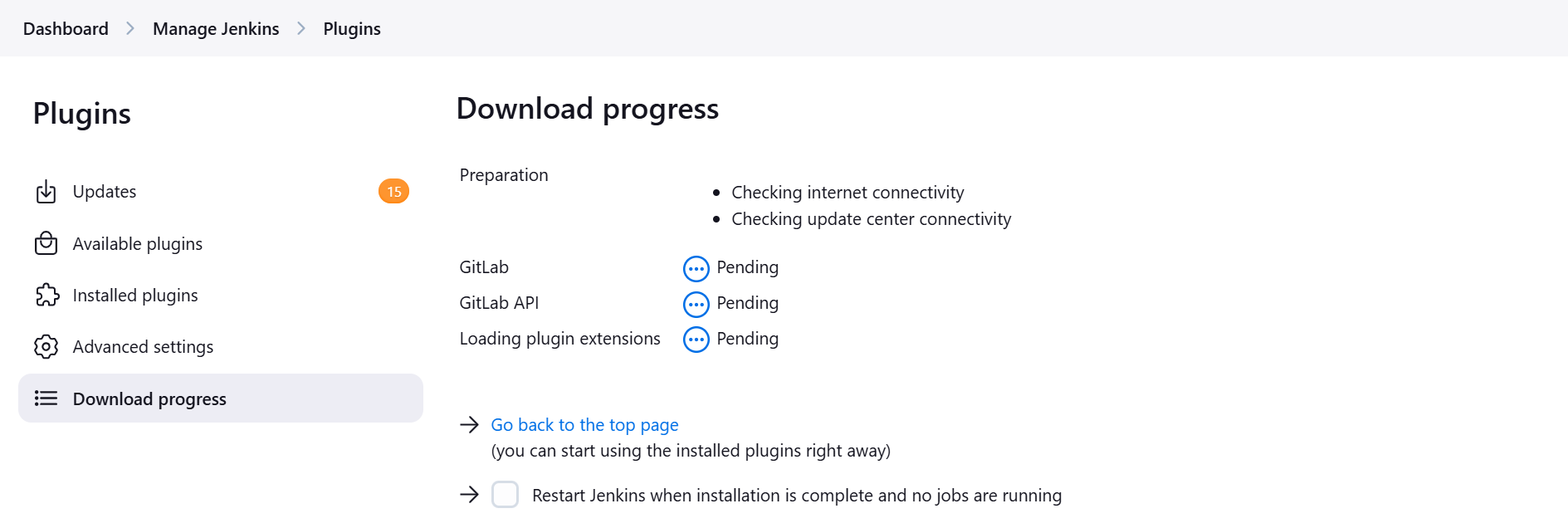
* Install Gitlab plugin without restart
  + Manage Jenkins > Plugins > Available > GitLab Plugins
  + Manage Jenkins > Plugins > Available > GitLab API



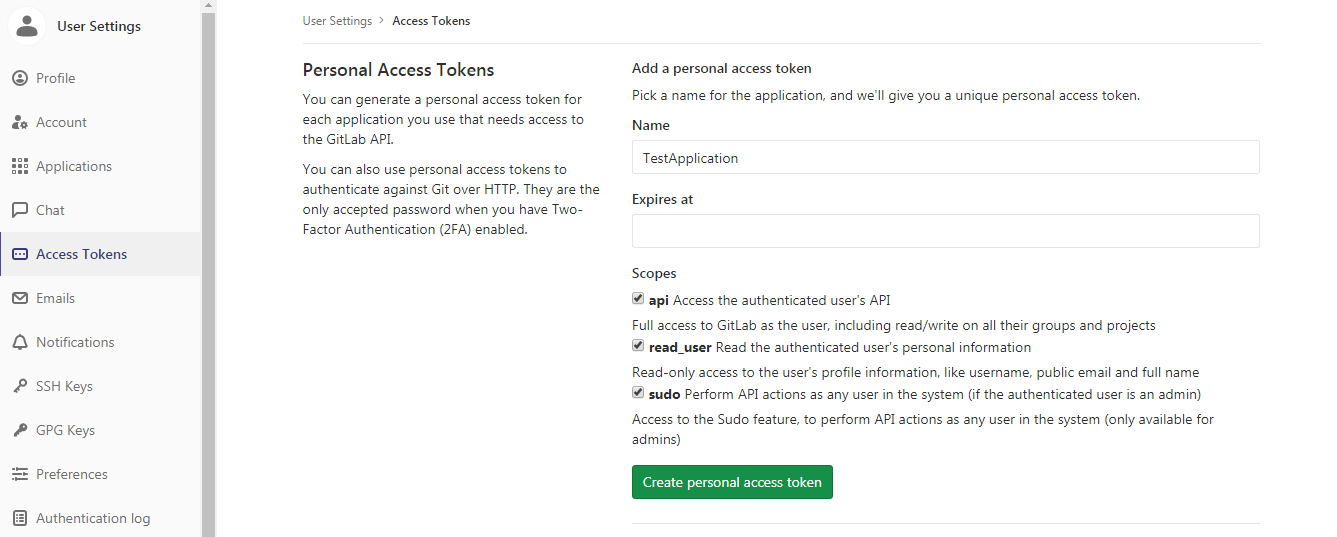
Under the Plugin Manager, click on the **Available** tab and search for the **Gitlab** plugin. It will show **Gitlab plugin** and **GitLab API** as a result.

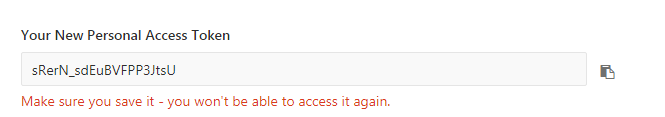


Once the plugin installs successfully, click the checkbox to restart Jenkins



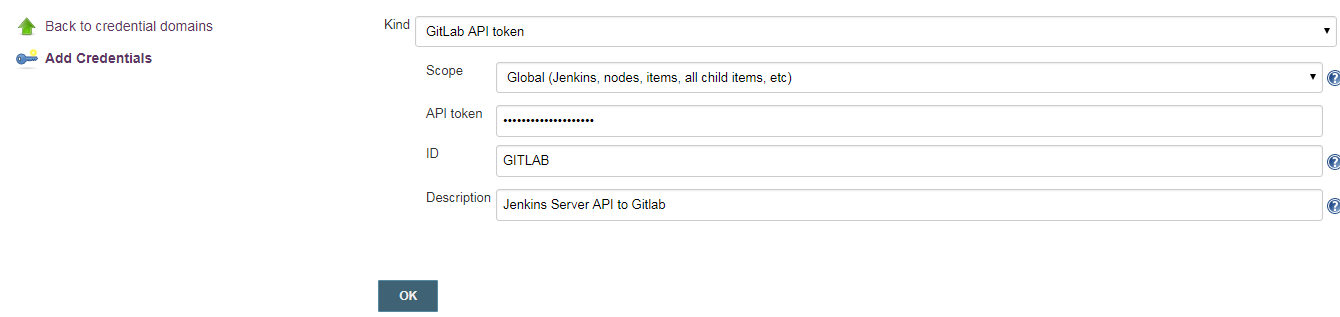
Generate Access Token in Gitlab: Gitlab → User Settings → Access Token





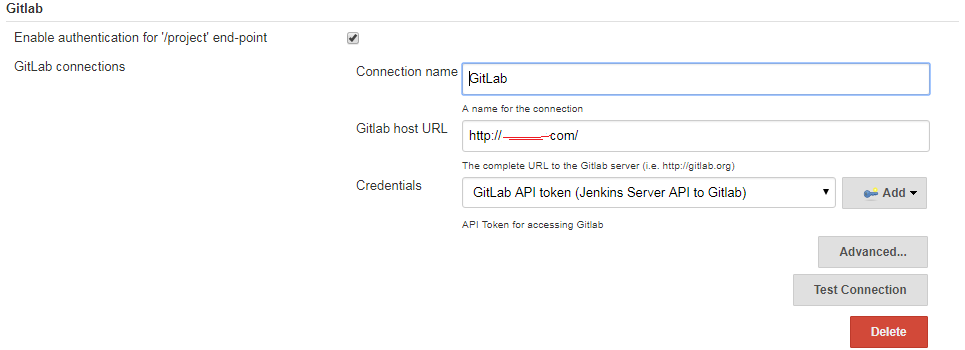
Configure Global Key in Jenkins: Jenkins → Credentials → System → Global Credentials → Add

Credentials



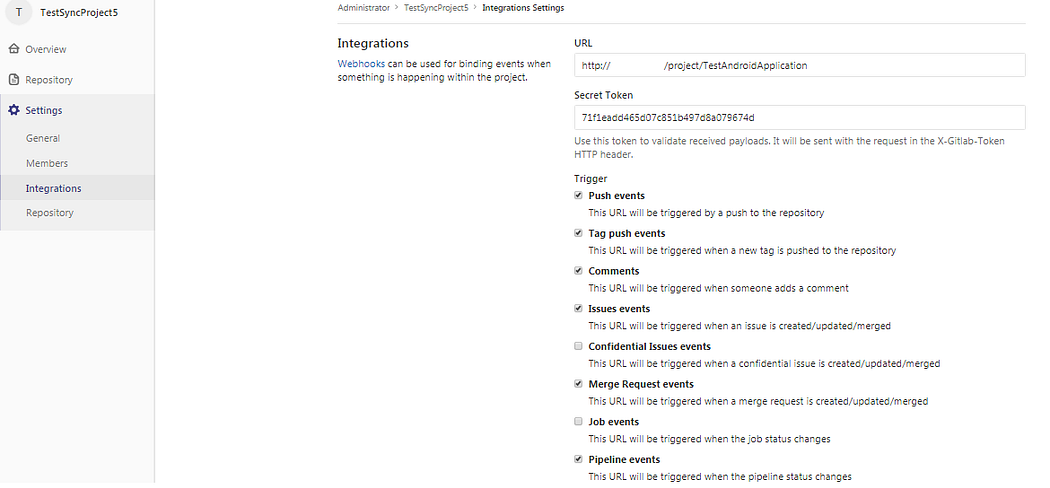
Add the API Token generated in previous steps. Once added it will appear in Global Credentials listing.

Configure Connection Between Jenkins and Gitlab: Jenkins → Manage Jenkins → Configure System → Gitlab

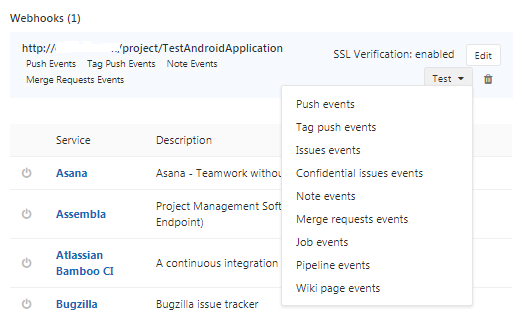


**Step 3: Configure Gitlab Webhooks**

Go to Gitlab → Projects → <Project\_Name> → Settings → Webhook



Add Gitlab CI Service URL and Secret Token Generated and Click on Add Webhook.



**3. Apache Maven**

Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

**Java Instalation**

#sudo apt update -y

#sudo apt install openjdk-17-jdk

#java --version

**Install Maven**

Download maven packages <https://maven.apache.org/download.cgi> onto Jenkins server. In this case using /opt/maven as installation directory.

Creating maven directory under /opt:

#mkdir /opt/maven

#cd /opt/maven

Downloading maven version 3.9.9:

#wget <https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.zip>

#unzip /opt/maven/apache-maven-3.9.9-bin.zip

Start Maven

#apt install maven

Check maven version:

#mvn -v



**Set the Maven Path**

Setup M2\_HOME and M2 paths in .profile of user and add these to path variable.

Open vim .profile and add below lines

JAVA\_HOME=/usr/lib/jvm/java-17-openjdk-amd64

M2\_HOME=/opt/maven/apache-maven-3.9.9

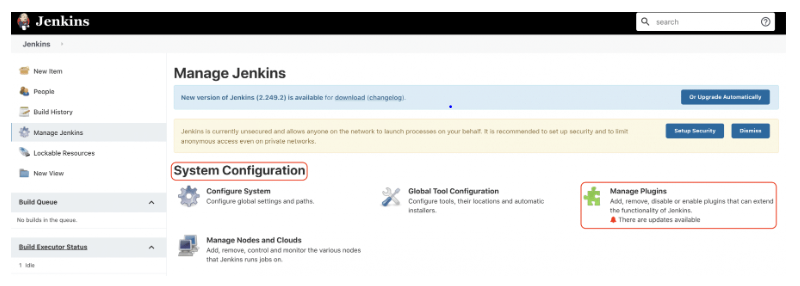
M2=$M2\_HOME/bin

PATH=$PATH:$JAVA\_HOME:$M2\_HOME:$M2:$HOME/bin

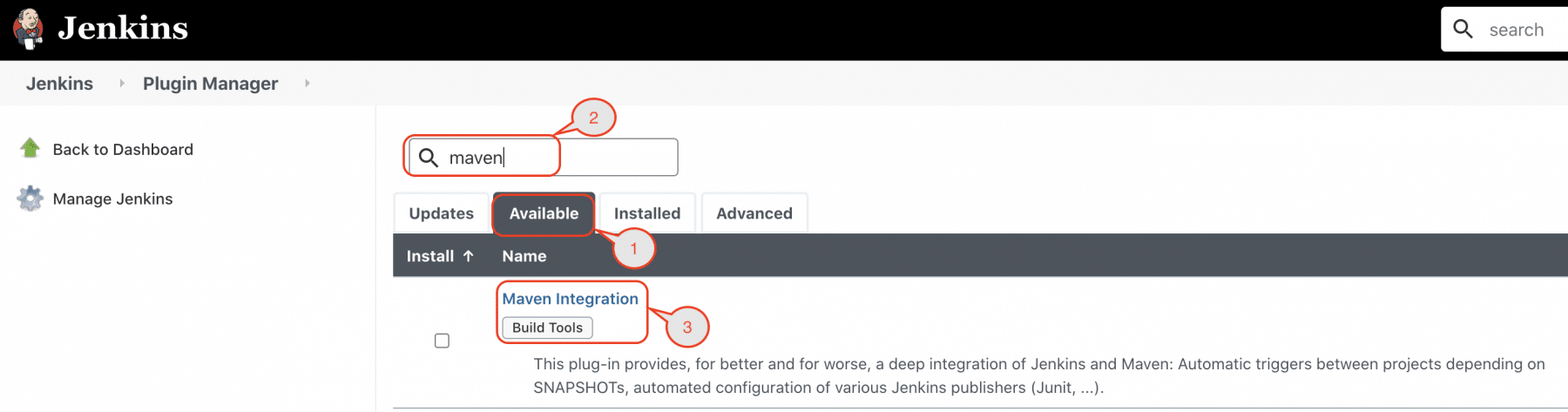
So far we have completed installation of maven software to support maven plugin on jenkins console. Let's jump on to jenkins to complete remining steps.

**Integrate maven with jenkins**

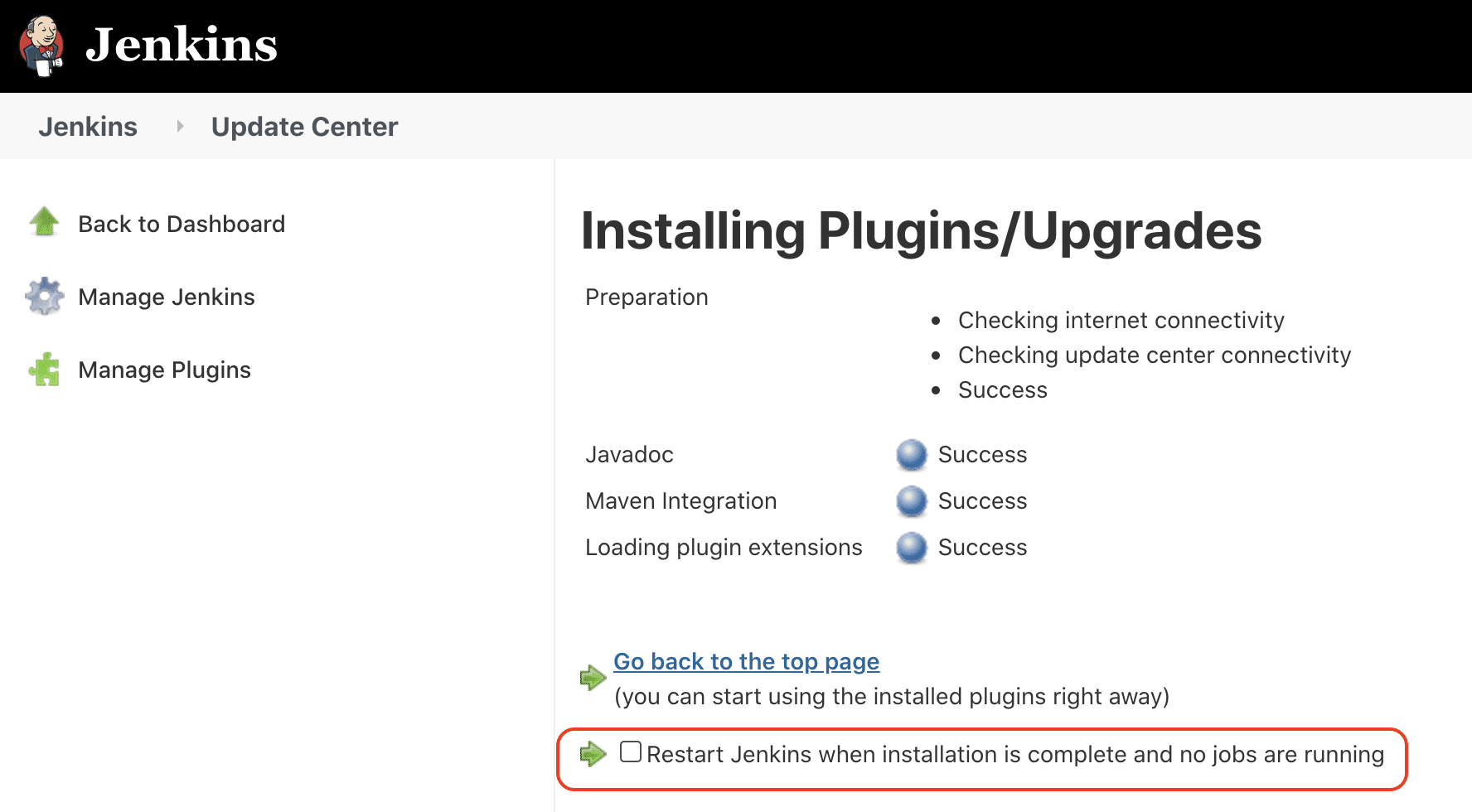
* Install maven plugin without restart
  + Manage Jenkins > Plugins > Available > Maven Invoker
* Install maven Integration Plugin without restart
  + Manage Jenkins > Plugins > Available > Maven Integration



Under the Plugin Manager, click on the **Available** tab and search for the **maven** plugin. It will show the **Maven Integration** and **Maven Invoker** plugin as a result.



Once the plugin installs successfully, click the checkbox to restart Jenkins*:*



* Configure maven path
  + Manage Jenkins > Tool > Maven

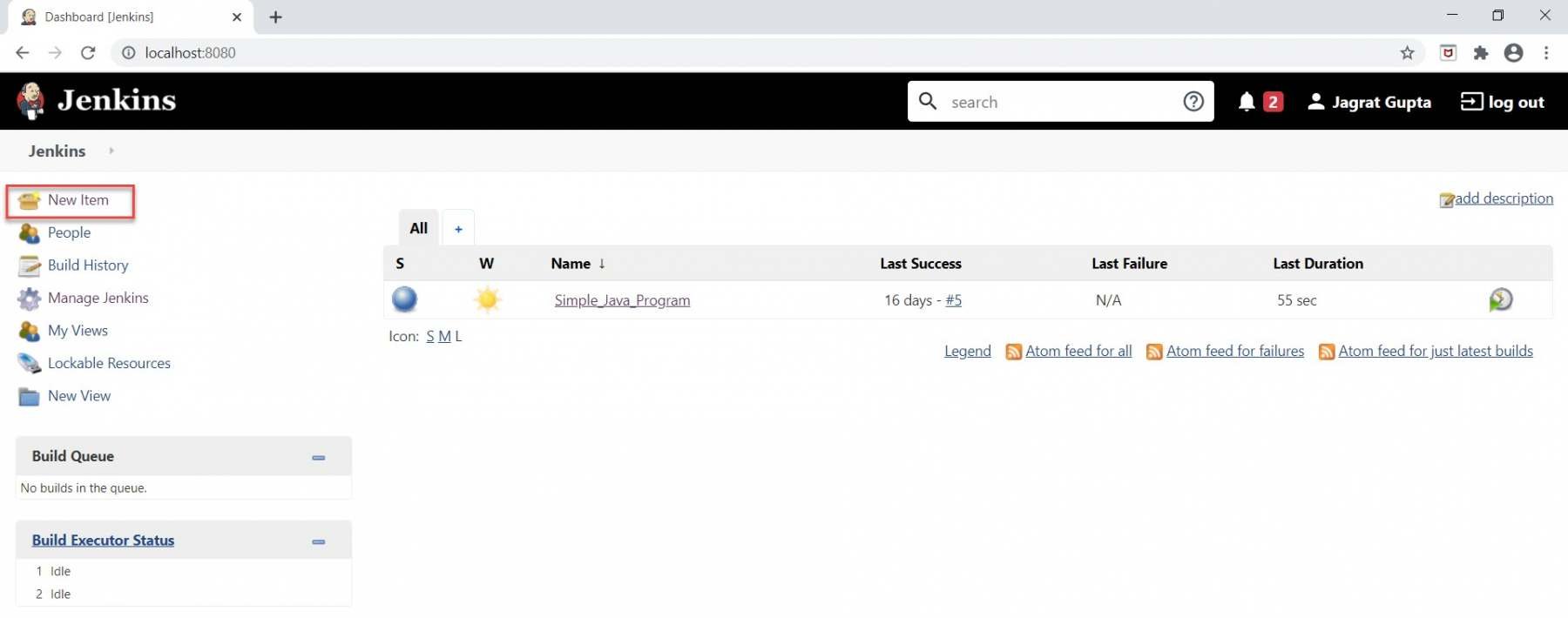


**Maven project in Jenkins**

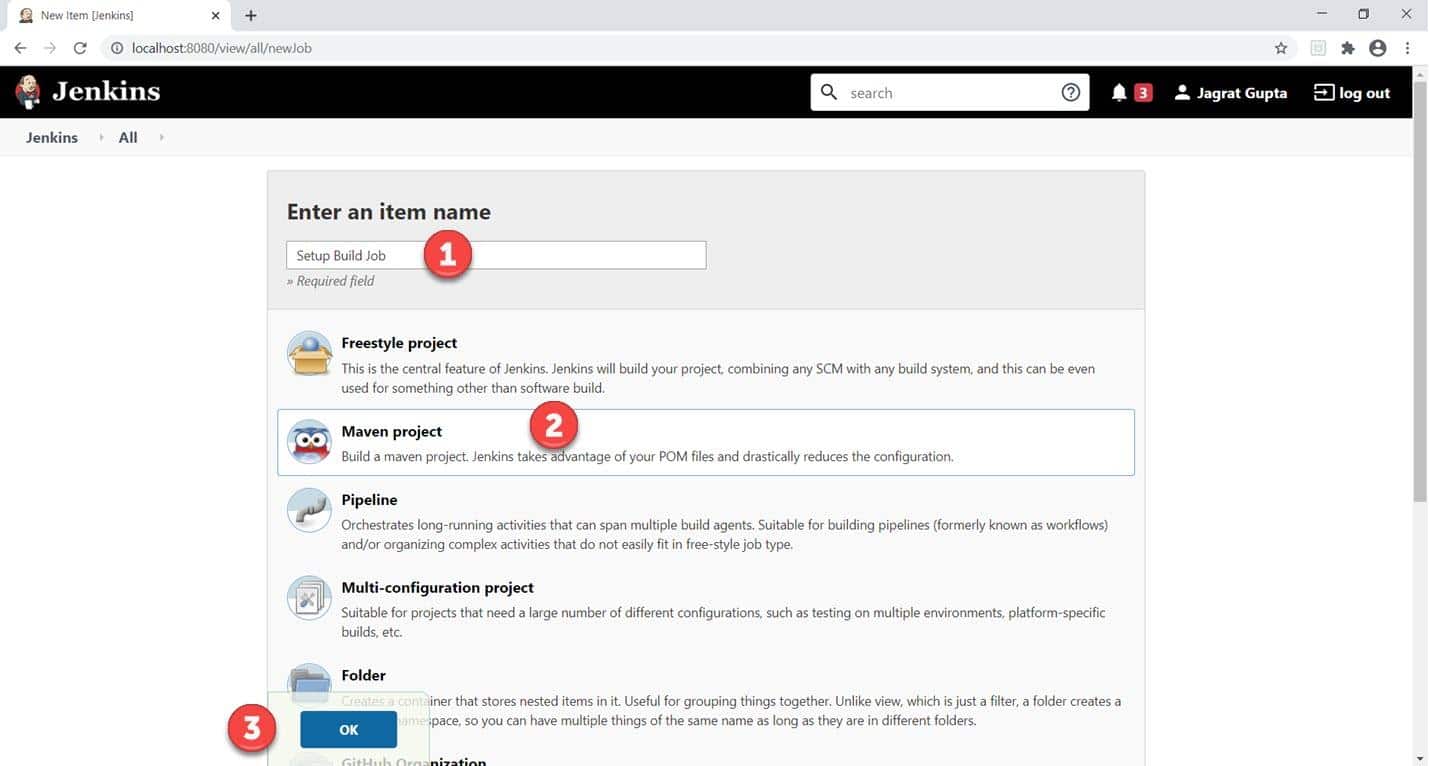
Jenkins provides a particular job type, which explicitly provides options for configuring and executing a Maven project. This job type is called the "Maven project." Let's see how we can create a Maven project in Jenkins and run the same.

**Create a Maven project in Jenkins**

Firstly, we need to create a job. To create this, click on the "New Item" option as highlighted below:

****

Now, do the following steps to create a new maven project:

****

1. Give the Name of the project.
2. Click on the Maven project. Kindly note that If this Maven Project option is not visible, we need to check whether the "Maven Integration" plugin is installed in Jenkins. If not installed, then install it and restart Jenkins.
3. Click on the OK button.

**4. Docker**

Docker is a software platform that allows you to build, test, and deploy applications quickly using containers. Its open-source project that automates the deployment of applications in containers. Docker images are read-only artifacts that contain the files needed to run an application. Docker containers are running instances of Docker images.

**Docker Installation**

Here we install docker on Jenkins server which we already install and configure.

Update your existing list of packages:

#sudo apt update

 Install a few prerequisite packages:

#sudo apt install apt-transport-https ca-certificates curl software-properties-common

Add the GPG key for the official Docker repository to your system:

#curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

Add the Docker repository to APT sources:

#sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"

Install Docker:

#sudo apt install docker-ce

Start docker:

#sudo systemctl start docker

User Jenkins need to add docker group

#usermod -aG docker Jenkins && sudo chmod 777 /var/run/docker.sock

Restart docker:

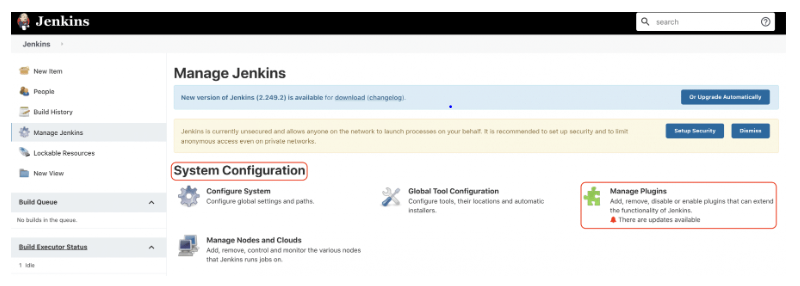
#sudo systemctl restart jenkins

Check docker version:

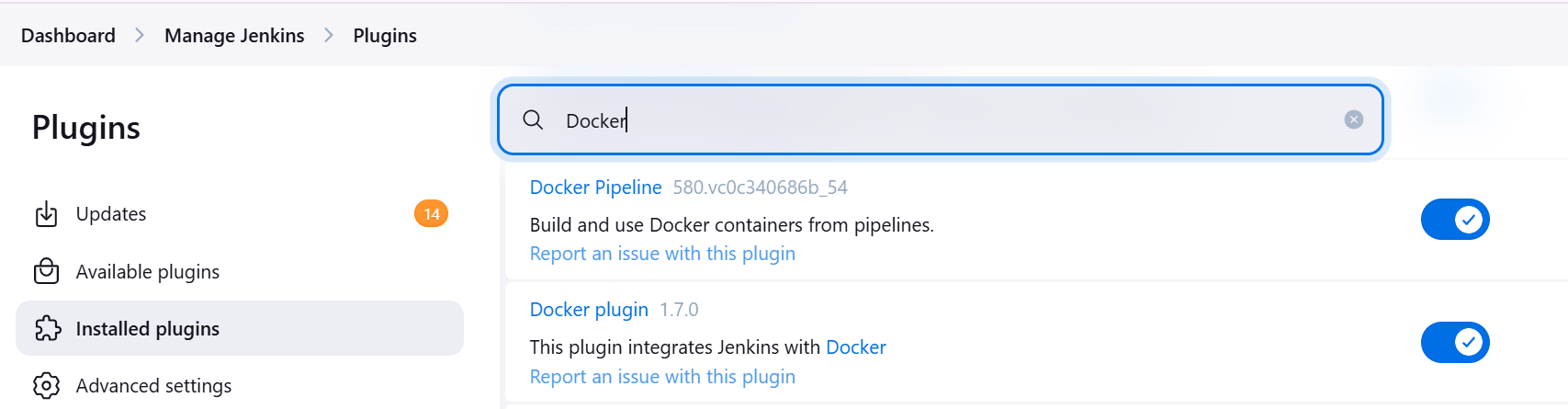


**Integrate Docker with Jenkins**

* Install docker plugin without restart
  + Manage Jenkins > Plugins > Available > Docker Plugins



Under the Plugin Manager, click on the **Available** tab and search for the **Docker** plugin. It will show the **Docker Plugins** plugin as a result. Install the plugins, now docker integrate with Jenkins.



**5. Trivy**

Trivy is a vulnerability scanner for containers, designed to detect vulnerabilities within your

container images, file systems, and Git repositories. It stands out for its simplicity and comprehensive database of vulnerabilities. Trivy scans for vulnerabilities in OS packages (Alpine, Red Hat, etc.) and application dependencies (NPM, Ruby Gems, etc.).

**Trivy Installation On Jenkins**

#sudo apt-get install wget apt-transport-https gnupg lsb-release

#wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -

#echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb\_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

#sudo apt-get update

#sudo apt-get install trivy

**Check trivy version**

#trivy --version

**Scan your remote git repository**

#trivy repo https://github.com/your-org/your-repo.git --format table -o repo\_details.html

**Scan Container Images**

#trivy image fir3eye/springboot1:latest --format table -o scan\_details.html

**Jenkinsfile**

Stage ('Trivy FS Scan') {

steps {

sh "trivy fs --format table -o fs.html . "

}

}

**6. SonarQube Scanner**

SonarQube is a code quality assurance tool that performs in-depth code analysis and generates an analysis report to ensure code reliability. It supports 30 major programming languages with various plugins.

It acts as a code inspector, analyzing code to identify errors, bugs, issues, mistakes, duplication, and security vulnerabilities. SonarQube provides integration with different build tools like Maven, Ant, Gradle, and continuous integration using Jenkins.

**1. Prepare Ubuntu server**

#sudo apt update

#sudo apt upgrade -y

**2. Install OpenJDK 17**

#sudo apt install -y openjdk-11-jdk

**3. Install and Configure PostgreSQL**

* Add PostgreSQL repository:

#sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt/ `lsb\_release -cs`-pgdg main" >> /etc/apt/sources.list.d/pgdg.list'

* Add PostgreSQL signing key:

#wget -q https://www.postgresql.org/media/keys/ACCC4CF8.asc -O - | sudo apt-key add –

* Install PostgreSQL:

#sudo apt install -y postgresql postgresql-contrib

* Enable DB server to start automatically on reboot:

#sudo systemctl enable postgresql

* Start DB server:

#sudo systemctl start postgresql

* Change the default PostgreSQL password:

#sudo passwd postgres

* Switch to the postgres user:

#su – postgres

* Create a user named sonar:

#createuser sonar

* Log into PostgreSQL:

#psql

* Set a password for the sonar user. Use a strong password in place of admin:

ALTER USER sonar WITH ENCRYPTED password 'admin';

* Create SonarQube database and set its owner to sonar:

CREATE DATABASE sonarqube OWNER sonar;

* Grant all privileges on SonarQube database to the user sonar:

GRANT ALL PRIVILEGES ON DATABASE sonarqube to sonar;

* Exit PostgreSQL:

\q

* Return to your non-root sudo user account:

#exit

**4. Download and Install SonarQube**

* Install the zip utility, which is needed to unzip the SonarQube files:

#sudo apt install -y zip

* Locate the latest download URL from [SonarQube official download page](https://www.sonarqube.org/downloads/). At the time of writing this document, the download URL was as follows:

[https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.8.100196.zip](https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-9.9.8.100196.zip?_gl=1*2b06an*_gcl_au*MTYxNTU0MjM4MC4xNzM0MzIyNzE2*_ga*NjE0NTg3MTYuMTczNDMyMjcxNg..*_ga_9JZ0GZ5TC6*MTczNTcyOTA1Ny4xMi4wLjE3MzU3MjkwNTcuNjAuMC4w)

* Unzip the downloaded file:

#sudo unzip sonarqube-9.9.8.100196.zip

* Move the unzipped files to /opt/sonarqube directory:

#sudo mv sonarqube-9.9.8.100196 /opt/sonarqube

**5. Add SonarQube Group and User**

* Create a sonar group:

#sudo groupadd sonar

#sudo passwd admin

* Create a sonar user and set /opt/sonarqube as the home directory:

#sudo useradd -d /opt/sonarqube -g sonar sonar

* Grant the sonar user access to the /opt/sonarqube directory:

#sudo chown -R sonar:sonar /opt/sonarqube

**6. Configure SonarQube**

* Edit the SonarQube configuration file:

#sudo vim /opt/sonarqube/conf/sonar.properties

**Step 1:** Find the following lines.

#sonar.jdbc.username=

#sonar.jdbc.password=

**Step 2:** Uncomment the lines, and add the database user sonar and password my\_strong\_password you created in Section 3.

sonar.jdbc.username=sonar

sonar.jdbc.password=admin

**Step 3:** Below those two lines, add sonar.jdbc.url.

sonar.jdbc.url=jdbc:postgresql://localhost:5432/sonarqube

**Step 4:** Save and exit the file.

* **Step 5**: Edit the sonar script file:

#sudo vim /opt/sonarqube/bin/linux-x86-64/sonar.sh

Uncomment the line and change it to.

RUN\_AS\_USER=sonar

Save and exit the file.

**7. Sonarqube Configure**

Edit this file sudo vim /etc/security/limits.conf

Add the following lines:

sonarqube   -   nofile   65536  
sonarqube   -   nproc    4096

**8**. **Modify Kernel System Limits**

SonarQube uses Elasticsearch to store its indices in an MMap FS directory. It requires some changes to the system defaults.

* **Step 2**: Edit the sysctl configuration file:

#sudo vim /etc/sysctl.conf

**Step 1:** Add the following lines:

vm.max\_map\_count=262144

fs.file-max=65536

ulimit -n 65536

ulimit -u 4096

**Step 3:** Save and exit the file.

* Reboot the system to apply the changes:

#sudo reboot

**9. Configure Firewall**

We need to add ports in firewall.

#ufw allow 9000/tcp  
#ufw allow 80/tcp  
#ufw allow 443/tcp  
#ufw reload

**10. Start Sonarqube Service**

* cd /opt/sonarqube/bin/linux-x86-64

#sudo ./sonar.sh start

* Check Status

#sudo ./sonar.sh status

**11. SonarQube Integration with Jenkins**

1. Generate an Authentication token on SonarQube Account > my account > Security > Generate Tokens
2. On Jenkins create credentials    Manage Jenkins > manage credentials > system > Global credentials > add credentials  - Credentials type: Secret text  - ID: sonarqube-key
3. Install SonarQube plugin  Manage Jenkins > Available plugins Search for **sonarqube scanner**

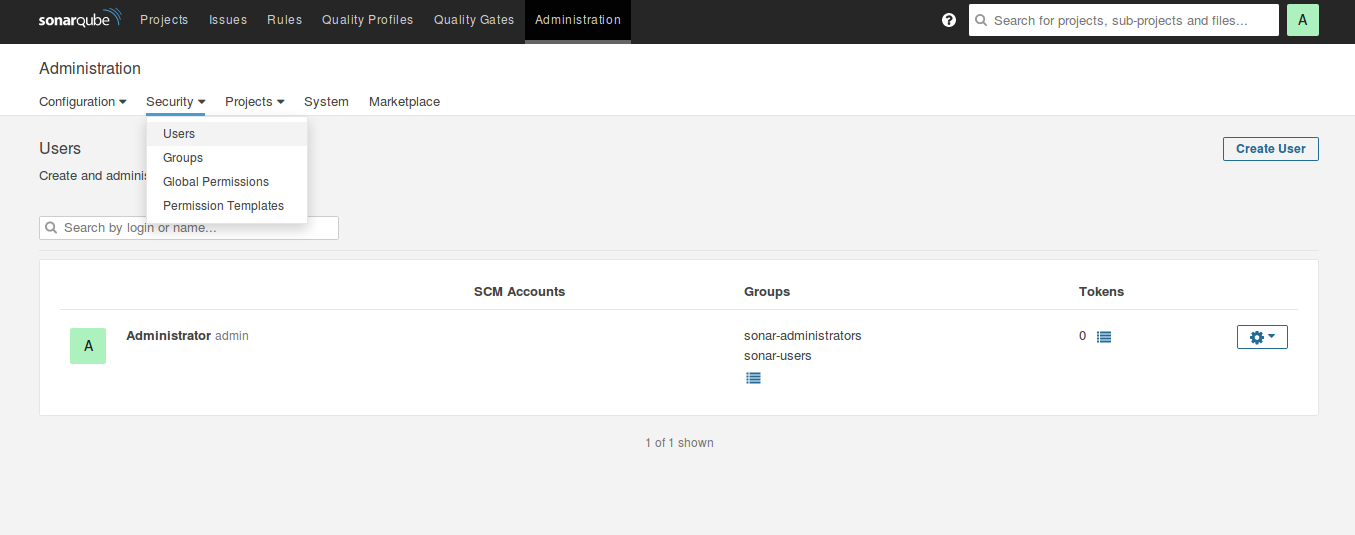
**12. Access SonarQube Web Interface**

* Access SonarQube in a web browser at your server's IP address on port 9000. For example:

http://localhost:9000

* Log in with username admin and password admin. SonarQube will prompt you to change your password.

**WARNING** SonarQube ships with a default administrator username and password of admin. This default password is not secure, so you’ll want to update it to something more secure as a good security practice.

[](https://github.com/jae1choi/sonaqueue-installation-guide/blob/master/eUpM2OE.png)

**13. If SonarQube fails a pipeline build**

When SonarQube fails a pipeline build due to a critical bug, it indicates that the analyzed code contains a high-severity issue that violates the defined quality standards. The configured Quality Gate in SonarQube has flagged this issue as significant enough to halt the build, effectively blocking the deployment process until the critical bug is resolved.

**Immediate Action Required the Development Team**

* **Prioritize the fix:** Developers must address the critical bug identified by SonarQube without delay to proceed with the build and deployment.

**Steps to Resolve the Issue**

1. **Investigate the issue:**

Access the SonarQube dashboard to locate the critical bug in the code and analyze its potential.

2**. Code review and fix:**

Review the affected code section and apply the necessary changes to eliminate the critical bug.

3. **Re-run analysis:**

After implementing the fix, re-run the SonarQube analysis to ensure the issue is resolved and the Quality Gate is passed. Push clean code in the source code repository.

**Key Points to Remember:**

* **Quality Gate Configuration:**
  + SonarQube Quality Gates can be customized to define which severity levels (e.g. critical, sever) will cause a build failure.
* **Communication and Collaboration:**
  + Maintain clear communication within the development team to ensure everyone is informed and aligned on resolving the issue promptly.
* **Continuous Improvement:**
  + Use SonarQube insights to enhance overall code quality by addressing critical issues, as well as other code smells and potential vulnerabilities.

**7. Nexus Repository**

Nexus Repository Manager is a powerful tool for managing artifacts and dependencies in your development environment. It supports various formats, including Maven, npm, docker etc.

**Prerequisites**

Before you begin, ensure you have the following:

* Ubuntu Latest
* A user with sudo privileges
* Java 17 or higher

**Step 1: Install Java**

#sudo apt update -y

#sudo apt install openjdk-17-jdk -y

**Verify Java version**

**#** java -version

**Step 2: Download Nexus**

Navigate to the [Sonatype download page](https://help.sonatype.com/repomanager3/download) and copy the download link for the latest version of Nexus Repository Manager OSS.

#cd /opt

#sudo wget https://download.sonatype.com/nexus/3/latest-unix.tar.gz

**Extract the downloaded archive:**

#sudo tar -xvf latest-unix.tar.gz

#sudo mv nexus-3\* nexus

#sudo rm latest-unix.tar.gz

**Step 3: Configure Nexus**

Create a dedicated user for running Nexus:

#sudo useradd nexus

**Change the ownership of the Nexus files to the nexus user:**

#sudo chown -R nexus:nexus /opt/nexus

#sudo chown -R nexus:nexus /opt/sonatype-work

**Edit the nexus.rc file to specify the Nexus user:**

#sudo vim /opt/nexus/bin/nexus.rc

Add the following line:

run\_as\_user="nexus"

Save and exit the file.

**Step 4: Create a Systemd Service File**

Create a service file to manage the Nexus service:

#sudo vim /etc/systemd/system/nexus.service

Add the following content to the file:

[Unit]

Description=Nexus Repository Manager

After=network.target

[Service]

Type=forking

LimitNOFILE=65536

ExecStart=/opt/nexus/bin/nexus start

ExecStop=/opt/nexus/bin/nexus stop

User=nexus

Restart=on-abort

[Install]

WantedBy=multi-user.target

Save and exit the file.

**Reload the systemd daemon to apply the changes:**

#sudo systemctl daemon-reload

**Enable and start the Nexus service:**

#sudo systemctl enable nexus

#sudo systemctl start nexus

**Check the status of the Nexus service:**

#sudo systemctl status nexus

**Step 5: Configure Firewall**

#sudo systemctl start ufw

#sudo systemctl enable ufw

#sudo ufw enable

**Open the necessary ports**

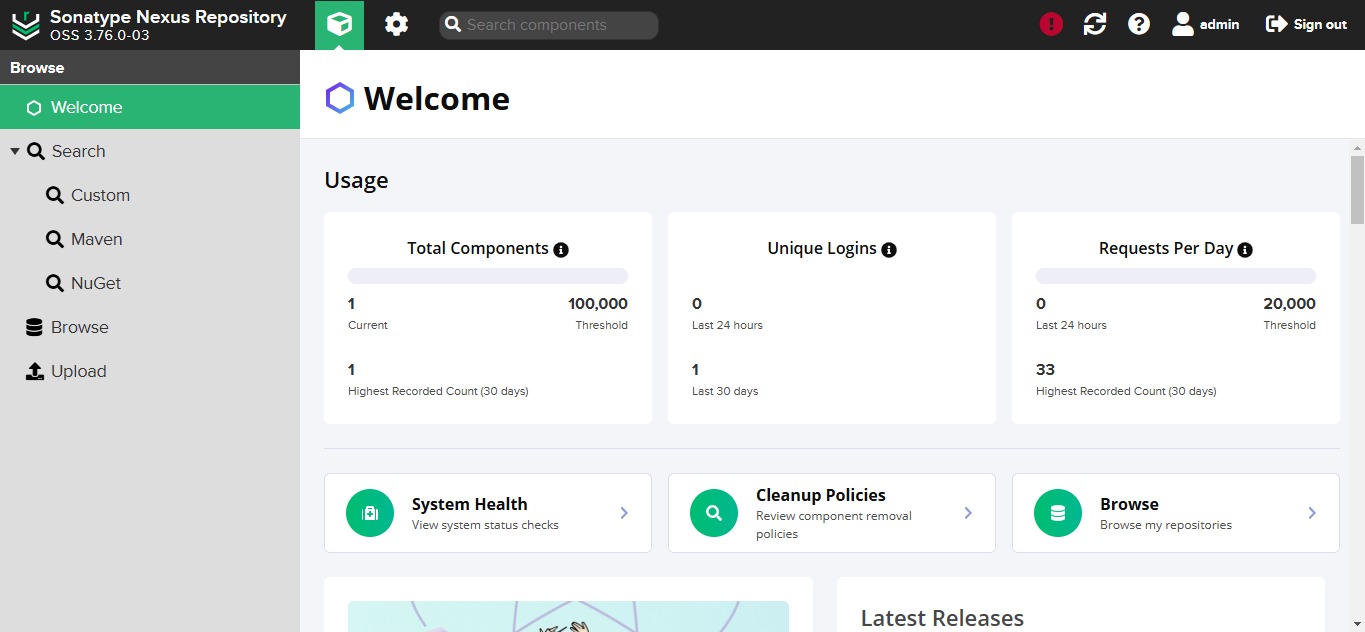
#sudo ufw allow 8081/tcp

#sudo ufw allow http

#sudo ufw reload

**Step 6: Access Nexus Repository Manager**

Open a web browser and navigate to http://public-ip>:8081. You should see the Nexus Repository Manager interface.



**Step 7: Initial Configuration**

The default admin credentials are:

* Username: admin
* Password: admin123

Use these credentials to log in for the first time. You will be prompted to change the password.

**Nexus Integration with Jenkins**

1. Configure Nexus server Manage Jenkins > Configure System > Name: nexus-server > Server URL https://localserver:8081/ - Server authentication token

2. In Jenkins create credentials Manage Jenkins > manage credentials > system > Global credentials > add credentials  - Credentials type: Username & Password  > Nexus login credential

3. Install SonarQube plugin Manage Jenkins > Available plugins Search for **Nexus Artifact Uploader**

**8. Harbor Registry**

Harbor is an open source registry that secures artifacts with policies and role-based access control, ensures images are scanned and free from vulnerabilities, and signs images as trusted. Harbor, a CNCF Graduated project, delivers compliance, performance, and interoperability to help you consistently and securely manage artifacts across cloud native compute platforms like Kubernetes and Docker.

**1. Update the system**:

#sudo apt update

**2. Install Docker and Docker Compose:**

#sudo apt install docker.io docker-compose -y

**3. Start and enable Docker:**

#sudo systemctl start docker

#sudo systemctl enable docker

**4. Install unzip and curl:**

#sudo apt install unzip curl -y

**5. Download the Harbor installer:**

#cd /opt

#wget <https://github.com/goharbor/harbor/releases/download/v2.12.1/harbor-online-installer->[v2.12.1.tgz](l)

**6. Extract the installer:**

#tar xzvf harbor-online-installer-v2.12.1.tgz

#cd harbor

Setup Docker to use an insecure registry by creating an entry into /etc//docker/daemon.json

Create a /etc//docker/daemon.json and add below lines

{ "insecure-registries" : [ "harber\_public\_ip" ] }

#sudo systemctl daemon-reload

#sudo systemctl restart docker

**7. Download the asc file**

# wget [https://github.com/goharbor/harbor/releases/download/v2.12.1/harbor-online installer- v2.12.1.tgz.asc](https://github.com/goharbor/harbor/releases/download/v2.12.1/harbor-online%20installer-%20v2.12.1.tgz.asc)

**8. Obtain the public key for the asc file**

#gpg --keyserver hkps://keyserver.ubuntu.com --receive-keys 644FF454C0B4115C

**9. Verify the genuity of the package**

#gpg -v --keyserver hkps://keyserver.ubuntu.com --verify harbor-online-installer v2.12.1.tgz.asc

**10. Generate or Obtain SSL Certificates:**

#mkdir -p /data/cert

#openssl req -newkey rsa:2048 -nodes -keyout /data/cert/server.key -x509 -days 365 -out /data/cert/server.crt

It will create two certificate inside /data/cert

server.crt

server.key

**11. Edit the Harbor yaml file**

vi ~/harbor/harbor.yml

hostname: 10.10.41.8

https:

# https port for harbor, default is 443

port: 443

# The path of cert and key files for nginx

certificate: /data/cert/server.cert

private\_key: /data/cert/server.key

**12. Setup Docker to use an insecure registry**

#sudo vi /etc/docker/daemon.json

insert mode and enter:

{

"insecure-registries" : [ "harbor\_public\_ip" ]

}

#sudo systemctl daemon-reload

#sudo systemctl restart docker

**11. Install Harbor with trivy using script:**

#cd /opt/harbor

#sudo ./install.sh --with-trivy

#sudo docker ps

**12. Start harbor, need to use**

#cd /opt/harbor

#docker-compose up (for start harbor )

#docker-compose down (for stop harbor )

**13. Log into Harbor from the Docker client**

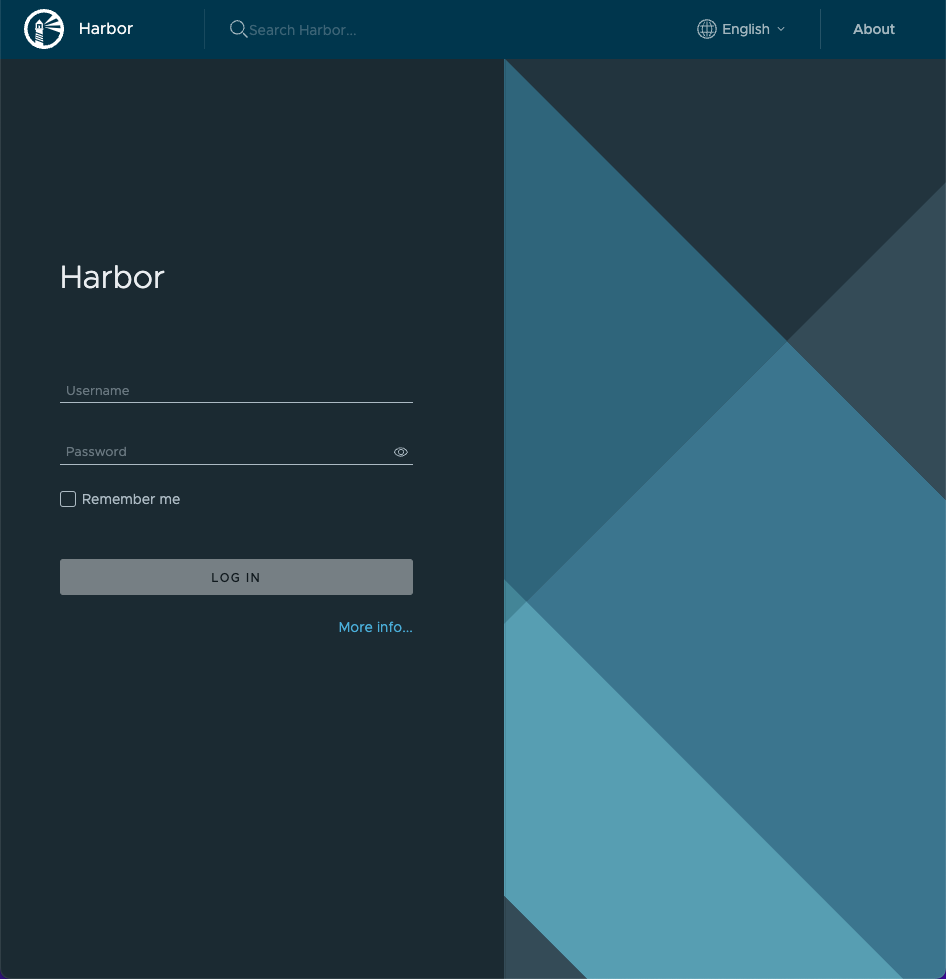
#docker login harbor\_public\_ip

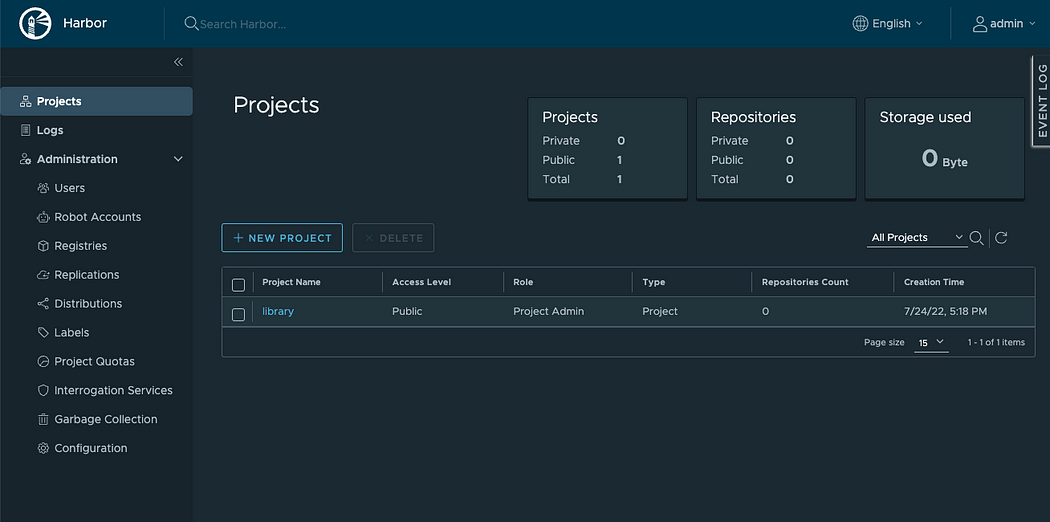
Use the default admin credentials to login: username: admin

password: Harbor12345

**14. Accessing Harbor from browser**

By default Harbor runs at port 80, access on browser <http://YOUR-SERVER-PUBLIC-IP:80>





**15. Push the Container Image to Harbor**

**15.1.** Use docker tag to create an image to include your Harbor instance and project

#docker tag test:v1 Harbor\_public\_ip/project/test:v1

**15.2 Use docker to push your container image to Harbor**

#docker push Harbor\_public\_ip/project/test:v1

**15.3**. Go to the Harbor instance and select the project to verify the container image is present

**16. Pull and run your container image from Harbor**

**16.1 Let's remove the existing simple-app container images from our local VM**

**#**docker rmiHarbor\_public\_ip/project/test:v1

**16.2 Now pull container image from your Harbor instance**

**#**docker pull Harbor\_public\_ip/project/test:v1

**Harbor Integration with Jenkins**

1. On Jenkins create credentials  Manage Jenkins > manage credentials > system > Global credentials > add credentials  - Credentials type: Username & Password  - Harbor login credentials
2. Manage Jenkins > Plugins > Available plugins Search for **Harbor Plugin**

# 9. Setup of Deployment Server

**Kubernetes**

Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerized applications across distributed to the environments. Developed initially by Google and now governed by the Cloud Native Computing Foundation (CNCF).

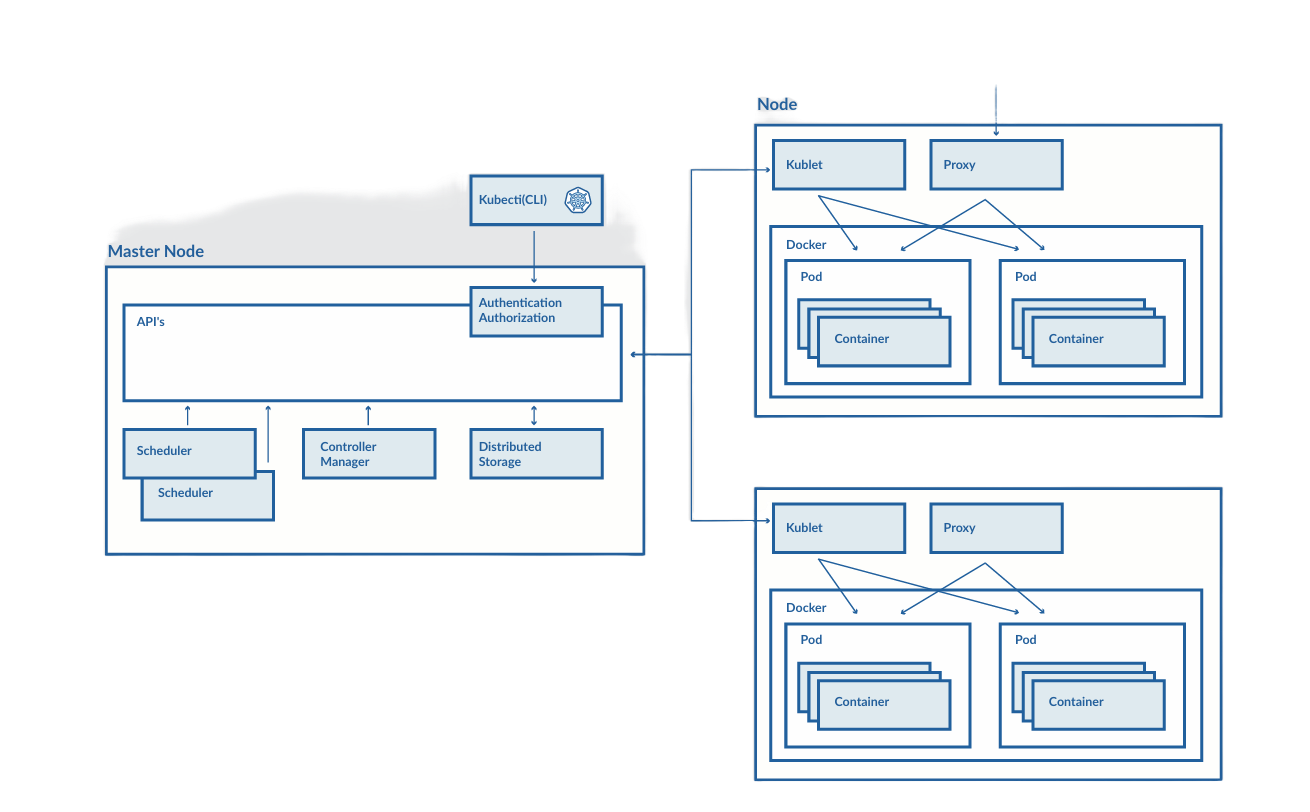


Fig: Kubernetes Architecture

Set up a 3-node Kubernetes cluster with kubeadm. The cluster includes one master node and two worker nodes. Pods can now communicate across the cluster using the Calico CNI.

**Requirements**

* Latest Ubuntu Server
* 3 Ubuntu instances with 20GB root volume
* 2 GB or more of RAM per machine (any less will leave little room for your apps).
* 2 CPUs or more for control plane machines
* SSH access to all instances

## Installation Steps (Run all command on master and Worker, except init and token create command on Master)

Launch 3 ubuntu server with 20GB root volumes. Run below command on each 3 servers (One Master & Two Nodes)

**Launch Instances**

#sudo apt update && sudo apt upgrade -y

#sudo apt install -y net-tools unzip

#sudo apt install apt-transport-https curl -y

#sudo swapoff -a

#sudo apt install sed -y

#sudo sed -i '/ swap / s/^\(.\*\)$/#\1/g' /etc/fstab

**This sets the hostname of the system to k8s-master**

#sudo hostnamectl set-hostname "k8s-master"

#exec bash

**IPs of your master and worker nodes in the /etc/hosts file**

sudo vim /etc/hosts

<master-ip> k8s-master

<worker-ip> k8s-worker

**configuration file to ensure the overlay and br netfilter kernel modules**

cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf

overlay

br\_netfilter

EOF

**Load the overlay and br\_netfilter modules into the kernel**

#sudo modprobe overlay

#sudo modprobe br\_netfilter

**Network settings for Kubernetes**

#cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf

net.bridge.bridge-nf-call-iptables = 1

net.bridge.bridge-nf-call-ip6tables = 1

net.ipv4.ip\_forward =1

EOF

#sudo sysctl –system

**Download and verify the Kubernetes apt repositories and keys.**

#sudo apt update

#sudo apt update && sudo apt install -y curl gpg

#sudo mkdir -p /etc/apt/keyrings

#curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.32/deb/Release.key | sudo tee /etc/apt/keyrings/kubernetes-#apt-keyring.asc > /dev/null

#echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.asc] #https://pkgs.k8s.io/core:/stable:/v1.32/deb/ /" | sudo tee /etc/apt/sources.list.d/kubernetes.list

**Install Kubernetes packages**

#sudo apt update

#sudo apt install -y kubeadm=1.32.0-1.1 kubelet=1.32.0-1.1 kubectl=1.32.0-1.1

**Installs Docker, which is the container runtime that Kubernetes uses to run containers**

#sudo apt install docker.io

**Install Container Runtime on Each Server**

SSH into each server using their public IP and install containerd:

#sudo apt update && apt install containerd -y

#ps -ef | grep -i containerd | grep -v grep

#netstat -nltp | grep -i containerd | grep -v grep

## Configure container

#sudo mkdir -p /etc/containerd

## #sudo containerd config default | sudo tee /etc/containerd/config.toml

## Modifies the containerd configuration to use systemd as the cgroup driver

#sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml

**Restart containerd Service**

#sudo systemctl restart containerd.service

**Pulls the necessary container images for Kubernetes to run the cluster**

#sudo kubeadm config images pull

***( Run Above all commands on each server i.e master & nodes/workers)***

## Initialize the Cluster (Run only on master)

#sudo kubeadm init --pod-network-cidr=10.10.0.0/16

**Create a .kube directory in your home directory**:

#mkdir -p $HOME/.kube

**Copy the Kubernetes configuration file to your home directory**:

#sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

**Change ownership of the file:**

#sudo chown $(id -u):$(id -g) $HOME/.kube/config

**Alternatively, if you are the root user, you can run:**

export KUBECONFIG=/etc/kubernetes/admin.conf

## Install Calio (Run only on master)

#kubectl create –f <https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml>

#curl [https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-resources.yaml](https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-resources.yaml%20) -O

**Check if MasterNode is ready**

#kubectl get nodes

## Join Worker Nodes

**Generate the join command on the master node:**

#sudo kubeadm token create --print-join-command

Example output:

sudo kubeadm join 172.31.33.66:6443 --token qg5kgy.o1ov92iu7d50dkye --discovery-token-ca-cert-hash sha256:e3f0feef4ad831253c3535f72e17c3bddc0c631e789c621f7a130e7e798aa313

**( Run the join command on each worker node to connect them to the cluster )**

**Integrate Kubernetes with Jenkins**

* Install Kubernetes plugin without restart
  + Manage Jenkins > Plugins > Available > **Kubernetes Plugins** and **Kubernetes CLI**

**10. Rancher (K8s Cluster Management)**

***Docker Installation***

Here we install docker on Jenkins server which we already install and configure.

Update your existing list of packages:

#sudo apt update

 Install a few prerequisite packages:

#sudo apt install apt-transport-https ca-certificates curl software-properties-common

Add the GPG key for the official Docker repository to your system:

#curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –

Add the Docker repository to APT sources:

#sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"

Install Docker:

#sudo apt install docker-ce

Start docker:

#sudo systemctl start docker

Download Docker Rancher image

# sudo docker pull rancher/rancher:latest

Run Rancher Container

# docker image ls

#sudo docker run --privileged -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher:latest

**Access Rancher UI**.

<http://Server_IP:80>

**Log in to Rancher**

Access Rancher UI , Rancher will ask you to enter Bootstrap Password , and to get the password, use the docker logs command with your container ID as suggested on the welcome page.

# docker logs container-id 2>&1 | grep "Bootstrap Password:"

The above command to get the password

Next Rancher will ask you to set a password for the admin user with two options for you to choose and set the password according to your needs

1) Use a randomly generated password:

2) Use the randomly generated password below. Set a specific password to use: Set a specific password to use.

**Now Register An Existing Kubernetes Cluster Into Rancher**

Step 1: Login To Rancher Open a web browser and type your Rancher’s instance IP address.

Step 2: Choose Import Existing Cluster Once logged in, click on “Import Existing” option as show below

Step 3: In the next screen, choose “Generic” under ‘Cluster Management

Step 4: Fill in the necessary details for your existing Kubernetes cluster:

Cluster Name

Description

Member Roles

Connect to your Kubernetes cluster via command line and run following commands to get cluster name of your Kubernetes cluster.

# kubectl config current-context

# kubectl config view

As we can see our cluster name is “Kubernetes”, now head back to Rancher GUI and specify the cluster name there.

Click on Create.

We will get the following instructions to register Kubernetes cluster to Rancher. Run these commands into your Kubernetes cluster. In most of the cases, we are using self-signed certificates in Kubernetes, so we can run the highlighted command.

Step 3: Registering Kubernetes Cluster To Rancher

Connect to your Kubernetes cluster via command line and run following command to register, select insecure command

# curl --insecure -sfL <https://192.168.1.4/v3/import/fhpr4>

Above command will create the necessary ClusterRole, namespace, secret and deployments etc.

Wait for 2 to 3 minutes to start all the Kubernetes objects and then head back to Rancher UI.

Perfect, above screen confirms that our Kubernetes cluster is registered successfully on Rancher UI. Now, when you go to home page of Rancher, you will see one more cluster is available there

When you click on newly registered cluster, you will get all details as shown below.

**11. Prometheus & Grafana**

## Install HELM

Run this on kubernetes master

#curl -fsSL -o get\_helm.sh <https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3>

#chmod 700 get\_helm.sh

#./get\_helm.sh

Check helm version

#helm version

## Install Prometheus using HELM

#helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

#helm repo add stable https://charts.helm.sh/stable

#helm repo update

# helm install kind-prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --set prometheus.service.nodePort=30000 --set prometheus.service.type=NodePort --set grafana.service.nodePort=31000 --set grafana.service.type=NodePort --set alertmanager.service.nodePort=32000 --set alertmanager.service.type=NodePort --set prometheus-node-exporter.service.nodePort=32001 --set prometheus-node-exporter.service.type=NodePort

#kubectl create -ns monitoring

#kubectl get namespace

## Run Prometheus and Grafana

# kubectl port-forward svc/kind-prometheus-kube-prome-prometheus -n monitoring 9090:9090 --address=0.0.0.0 &

#kubectl port-forward svc/kind-prometheus-grafana -n monitoring 31000:80 --address=0.0.0.0 &

**Now Run Prometheus**

https://public\_ip:9090

user: admin

pass: prom-operator

**Now Run Grafana**

https://public\_ip:31000

user: admin

pass: admin

## Prometheus Queries

sum (rate (container\_cpu\_usage\_seconds\_total{namespace="default"}[1m])) / sum (machine\_cpu\_cores) \* 100

sum (container\_memory\_usage\_bytes{namespace="default"}) by (pod)

sum(rate(container\_network\_receive\_bytes\_total{namespace="default"}[5m])) by (pod)

sum(rate(container\_network\_transmit\_bytes\_total{namespace="default"}[5m])) by (pod)

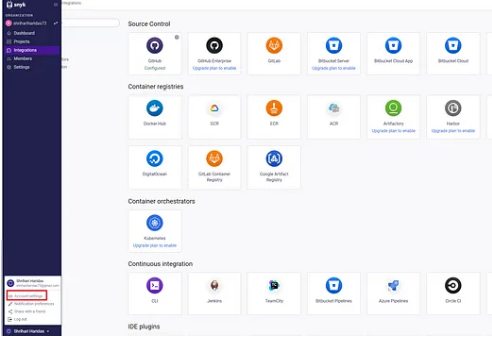
# **Node Exporter already installed in it**

**12. Snyk**

Snyk is a platform that allows you to scan, prioritize, and fix security vulnerabilities in your code, open-source dependencies, container images, and infrastructure as code configurations

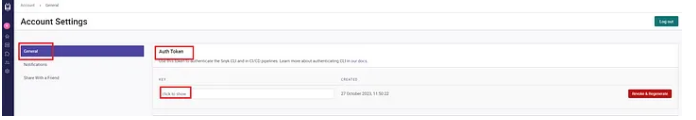
**Step1**:

Go to ‘**Snyk**,’ create your account, then navigate to the **dashboard**. On the left-side menu bar, click on **your account name**. From there, click on ‘**Account Settings**”



**Step2:**

After entering, proceed to the ‘**General**’ tab. You will find the ‘**Generate Auth Token**’ option there; click on it and**copy the token**. Save the token in Notepad. In my case, I have already created that token.



**Step3:**

Now, go back to server/AWS server, and follow the command below to **install the Snyk CLI**:

#curl https://static.snyk.io/cli/latest/snyk-linux -o snyk  
#chmod +x ./snyk  
#mv ./snyk /usr/local/bin/

**Step3:**

**Authenticate Snyk** with the token you generated.

snyk auth <Token>

It show your account authenticate successfully

**Step4:**

Next, **clone the following repository** for demonstration purposes.

#git clone https://github.com/shrihariharidass/devsecops-jenkins-k8s-tf-sast-sca-sonarcloud-snyk-repo.git

**#** cd clone repo

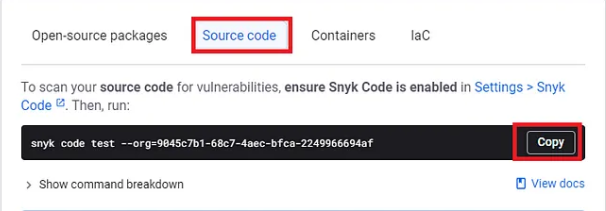
**Step5:**

Return to the ‘Snyk Dashboard.’ On the right side, you will find the ‘Add Project’ option. Click on it, and choose ‘CLI,’ as we performed the above operation.



**Step 6:**

Scroll down a bit and navigate to the ‘Source Code’ tab. You will find a command there; copy that.



**Step7:**

Proceed to the AWS console, navigate to the project folder, and **paste the copied command**. This will scan your code and provide you with the results.

# You will observe the severity of issues in your code, and based on that, you can address and resolve the errors.

**13. OWASP Dependency-Check**

A tool to scan source code and find public vulnerabilities in your dependencies.

**Initial Setup**

Required Plugins

*OWASP Dependency-Check:*

Go to Manage Jenkins > Manage Plugins > Available Plugins.

Search for **OWASP Dependency-Check** and install.

**Jenkins Configuration**

1. Global Tool Configuration:
2. Go to Manage Jenkins > Global Tool Configuration.
3. Dependency-Check:
4. Add a new dependency called **DP-check**.
5. Select automatic installation and version 7.2.0

**In cicd pipeline**

stage('Dependency Check') {

steps {

dependencyCheck additionalArguments: '--scan target/', odcInstallation: 'DP-check'

dependencyCheckPublisher pattern: '\*\*/dependency-check-report.xml'

}

}

**Final CICD pipeline code**

**pipeline {**

**agent any**

**environment {**

**DOCKERHUB\_CREDENTIALS = credentials('harbor-cred')**

**HARBOR\_REGISTRY = '10.10.41.8:80'**

**NEXUS\_URL = 'http://10.10.41.9:8081'**

**NEXUS\_REPO = 'vprofile-snapshots'**

**ARTIFACT\_GROUP = 'com/visualpathit/vprofile'**

**ARTIFACT\_NAME = 'vprofile'**

**SETTINGS\_PATH = '/var/lib/jenkins/.m2/settings.xml'**

**WAR\_FILE\_NAME = 'vprofile-2.0-SNAPSHOT.war'**

**KUBECONFIG = credentials('kube-cred')**

**}**

**tools{**

**maven 'maven'**

**}**

**stages {**

**stage('Checkout Gitlab code') {**

**steps {**

**echo 'Cloning repository...'**

**git branch: 'main', url: 'https://github.com/cdivyanshu/vprofile-project.git'**

**}**

**}**

**stage('Unit Tests') {**

**steps {**

**echo 'Running Unit Tests...'**

**sh 'mvn compile'**

**}**

**}**

**stage('Code Quality scanner') {**

**steps {**

**echo 'Running SonarQube analysis...'**

**withSonarQubeEnv('sonar-server') {**

**sh 'mvn clean package sonar:sonar'**

**}**

**}**

**}**

**stage('Maven Code Build') {**

**steps {**

**echo 'Building application...'**

**sh 'mvn clean package'**

**}**

**}**

**stage('Deploy Snapshot to Nexus') {**

**steps {**

**script {**

**sh """**

**mvn deploy \**

**-DaltDeploymentRepository=nexus::http://10.10.41.9:8081/repository/maven-snap/ \**

**--settings ${SETTINGS\_PATH}**

**"""**

**}**

**}**

**}**

**stage('Build Docker Image') {**

**steps {**

**script {**

**sh 'docker build -t tester:v2 .'**

**sh 'docker tag tester:v2 10.10.41.8:80/cicd-testing/tester:v2'**

**}**

**}**

**}**

**stage('Push Docker Image to Harbor') {**

**steps {**

**sh 'echo $DOCKERHUB\_CREDENTIALS\_PSW | docker login 10.10.41.8:80 -u $DOCKERHUB\_CREDENTIALS\_USR --password-stdin'**

**sh 'docker push 10.10.41.8:80/cicd-testing/tester:v2'**

**}**

**}**

**stage('Deploy to Kubernetes') {**

**steps {**

**script {**

**sh "kubectl get node"**

**sh "kubectl create namespace tts-group"**

**}**

**}**

**}**

**}**

**}**