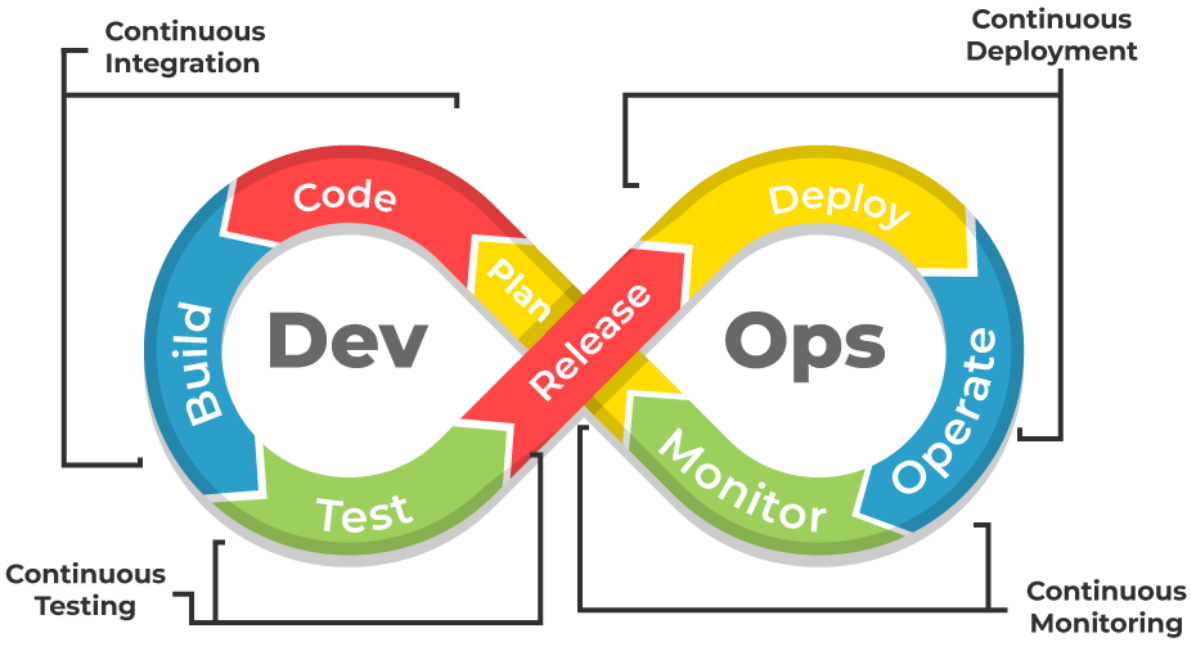
ROS2-CI/CD Automation Project

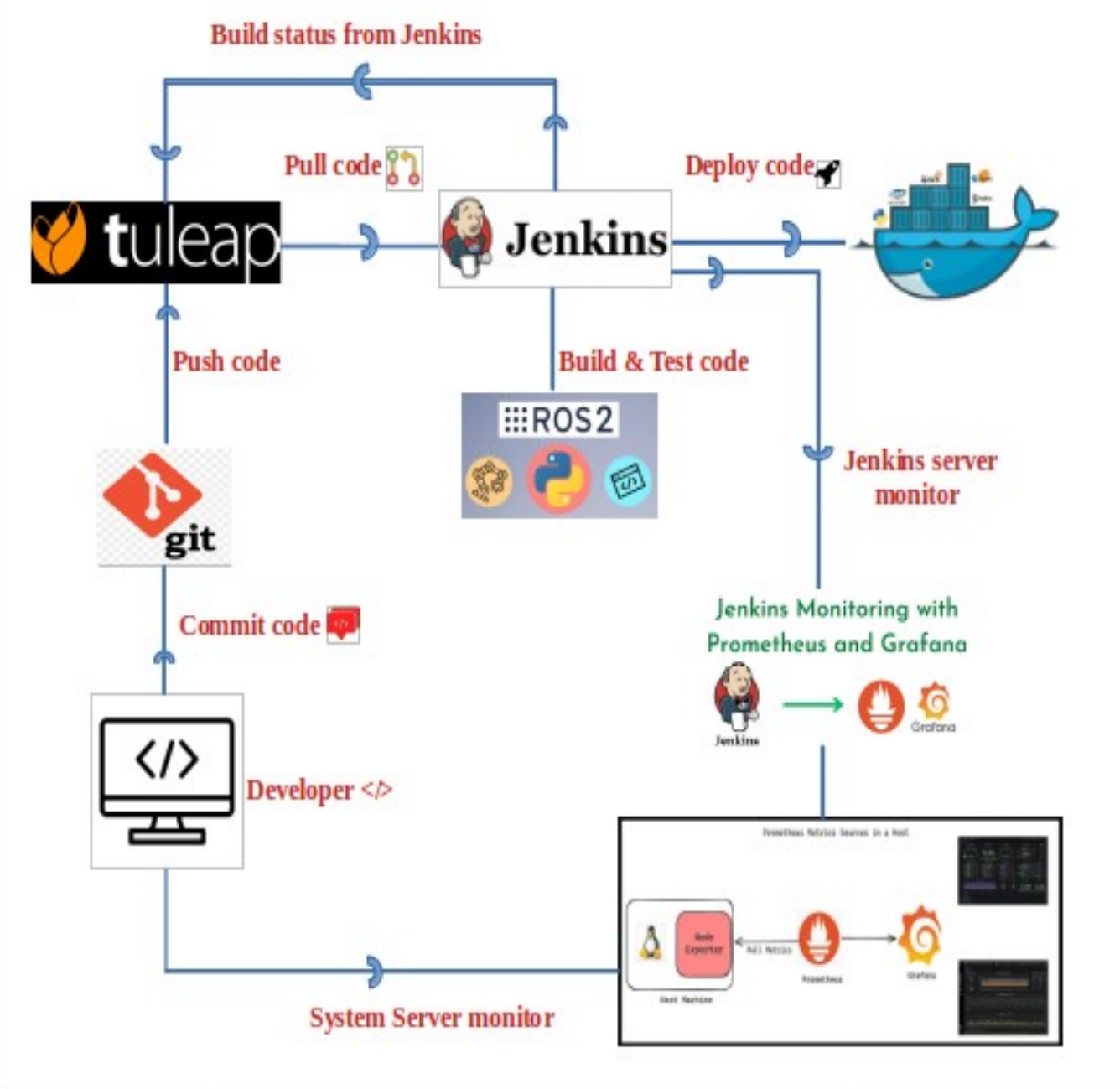
PROJECT AIM:

The aim of the project is to automate the continuous integration and continuous deployment (CI/CD) of ROS2 C++ or Python code. This involves using Tuleap for project management and version control, Jenkins for building and testing the code, and Docker for containerized deployment. Additionally, the project includes monitoring servers with Grafana and Prometheus, ensuring a streamlined and efficient development workflow.

ROS2 CI/CD Architecture:



Architecture:



Project Requirement Tools:

Foundation

- ROS2: Robot Operating System
- Red Hat OS: Base Operating System
- Nginx: Web Server and Reverse Proxy

1. **Plan**
    - Tuleap: Project Management and Version Control
2. **Code**
    - Visual Studio Code: Integrated Development Environment
    - Git: Version Control System
3. **Build**
    - Colcon: Build Tool
4. **Test**
    - Google Test (gtest) and Google Mock (gmock): Testing Frameworks
    - pytest: Python Testing Framework
    - Jenkins: Continuous Integration and Deployment
5. **Release**
    - CI/CD Workflow: Automation and Processes (Jenkins)
6. **Deploy**
    - Docker: Containerized Deployment
    - Docker Compose: Multi-container Docker Applications
7. **Operation**
    - Security Measures: SSL/TLS and Access Controls
8. **Monitor**
    - Grafana: Monitoring and Visualization
    - Prometheus: Metrics Collection and Monitoring

System Specifications:

- **OS:** Red Hat Enterprise Linux-9.4
- **RAM:** 12 GB
- **CPU:** Quad-core processor (e.g., Intel i5)
- **Storage:** 512 GB SSD

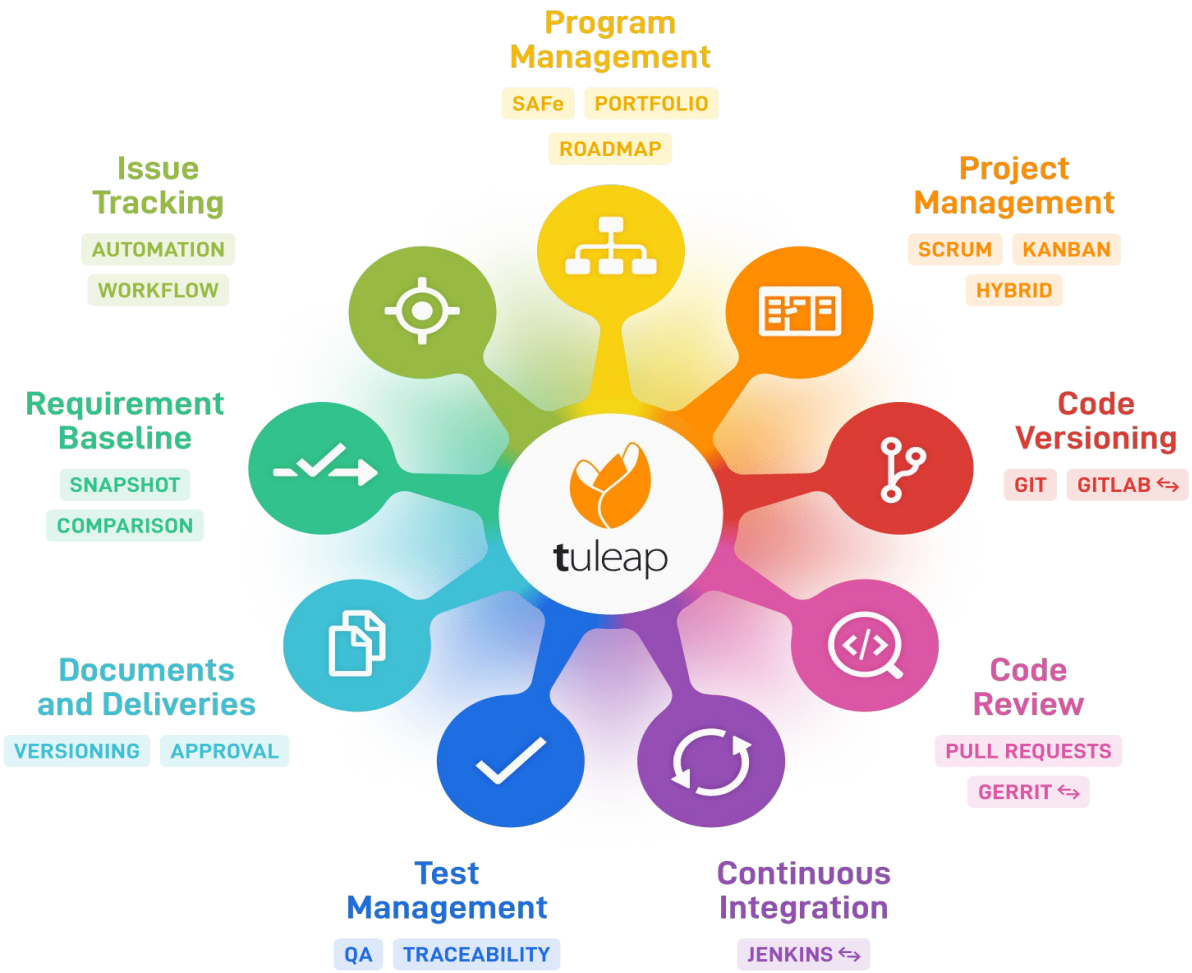
Tuleap Installation on Red Hat 9.4

TULEAP:

Tuleap is an open-source suite for application lifecycle management (ALM). It helps teams to manage projects, track issues, perform continuous integration, and manage version control, among other tasks.

Key Points:

- **Project Management**: Tuleap offers tools for agile project management (Scrum, Kanban), traditional project management (Waterfall), and hybrid methodologies.
- **Issue Tracking**: Tuleap provides comprehensive issue tracking capabilities, allowing teams to report, track, and resolve bugs and other issues.
- **Version Control**: It supports Git, SVN, and CVS repositories, enabling teams to manage their source code and collaborate effectively.
- **Continuous Integration**: Tuleap integrates with Jenkins to facilitate continuous integration and continuous deployment (CI/CD) processes.
- **Document Management**: Users can create, store, and manage project-related documentation within Tuleap.
- **Collaboration Tools**: Tuleap includes features for forums, wikis, and chat to enhance team communication and collaboration.
- **Customization**: Tuleap is highly customization, allowing users to tailor the platform to meet their specific project needs and workflows.
- **REST API**: Tuleap offers a robust REST API for integrating with other tools and services, facilitating automation and data exchange.


Program Management
SAFe
PORTFOLIO
ROADMAP
Issue Tracking
AUTOMATION
WORKFLOW
Project Management
SCRUM
KANBAN
HYBRID
Requirement Baseline
SNAPSHOT
COMPARISON
tuleap
Code Versioning
GIT
GITLAB
Documents and Deliveries
VERSIONING
APPROVAL
Code Review
PULL REQUESTS
GERRIT
Test Management
QA
TRACEABILITY
Continuous Integration
JENKINS

Tuleap Installation on red hat 9.4:

Process:

Tuleap:-

- Enterprise Linux 9 (RHEL 9.4).
- You must disable SE Linux prior to the install.
- The server will need an Internet connection as it will download external packages.

Commands:

- sestatus
- sudo nano /etc/selinux/config
- SELINUX=disabled
- sudo reboot
- sestatus

Install dependencies:

$ dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm

$ dnf install https://rpms.remirepo.net/enterprise/remi-release-9.rpm

Install packages - Tuleap Community Edition:

$ dnf install https://ci.tuleap.net/yum/tuleap/rhel/9/dev/x86_64/tuleap-community-release.rpm

Install Tuleap by running the following command:

```
$dnf install -y \
  mysql-server \
  tuleap \
  tuleap-theme-burningparrot \
  tuleap-theme-flamingparrot \
  tuleap-plugin-agiledashboard \
  tuleap-plugin-graphontrackers \
  tuleap-plugin-git \
  tuleap-plugin-hudson-git \
  tuleap-plugin-pullrequest \
  tuleap-plugin-gitlfs \
  tuleap-plugin-document \
  tuleap-plugin-onlyoffice \
  tuleap-plugin-embed \
  tuleap-plugin-gitlab \
  tuleap-plugin-openidconnectclient \
  tuleap-plugin-ldap
```

Additional plugins for tuleap:

https://docs.tuleap.org/installation-guide/step-by-step/install-plugins.html

 # Deploy RPM package
$ dnf install tuleap-plugin-<>

Project management:

tracker
graphontrackers
agiledashboard
cardwall

File deliveries and documentation:
document
frs
mediawiki
webdav

Source control and continuous integration:
svn
git
gitlfs
pullrequest
hudson

Authentication and permissions:
ldap
captcha
archivedeleteditems

Integrations:

embed
gitlab
botmattermost
botmattermost-agiledashboard
botmattermost-git

Tuleap API plugin:

dnf install tuleap-plugin-api-explorer

Prepare the tuleap database:

 # Create /etc/my.cnf.d/tuleap.cnf file
$ echo -e '[mysqld]\nsql-mode="NO_ENGINE_SUBSTITUTION"' >
/etc/my.cnf.d/tuleap.cnf

 # Activate MySQL
$ systemctl enable mysqld

 # Start it
$ systemctl start mysqld

 # Status it
$ systemctl status mysqld

 # Set a password
$ mysqladmin -u root password (cair123)

Verify the mysql:
$ mysql -u root -p
password: cair123

Setup:

As root, run:

```
/usr/share/tuleap/tools/setup.sh \
  --configure \
  --server-name=tuleap.example.com \
  --mysql-server=localhost \
  --mysql-password=cair123
```

Note:

FQDN being the name of the server as you access it on your network (localhost for a local test, tuleap.example.com with a DNS entry 192.168.1.123 if you only have an IP address)

XXXXX being the password of root password of the db configured earlier.

Ensure the firewall is properly configured. Open needed ports:

Web (TCP/80 & TCP/443)

SSH (git, admin): TCP/22

Allow the ports:

$ sudo systemctl start firewalld
$ sudo systemctl enable firewalld
$ sudo systemctl status firewalld

Open HTTP and HTTPS ports on tuleap:

$ sudo firewall-cmd --permanent --add-service=http
$ sudo firewall-cmd --permanent --add-service=https
$ sudo firewall-cmd --permanent --add-service=ssh
$ sudo firewall-cmd --reload
$ sudo systemctl restart tuleap
$ sudo systemctl start tuleap

Configure the host name and IP address:

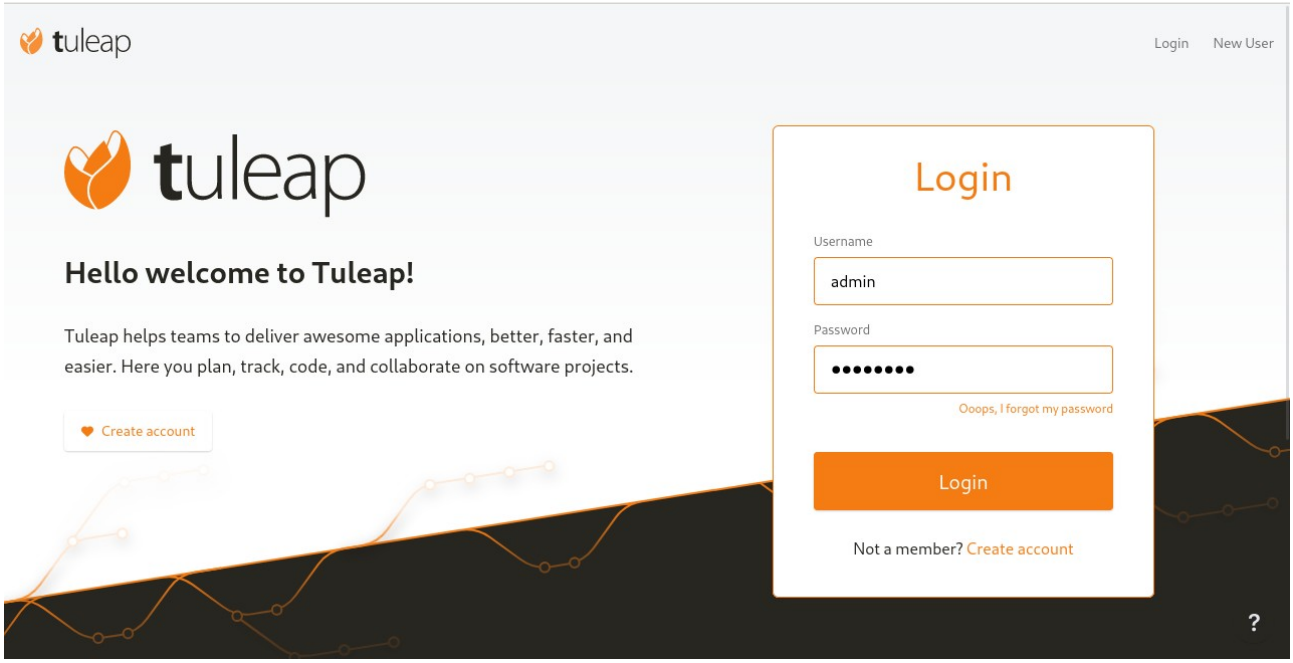
$ nano etc/hosts
127.0.0.1 tuleap.example.com

Browser Access:

(localhost)----https://tuleap.example.com or 127.0.0.1

Credentials:

username: admin
$ cat /root/.tuleap_passwd (password location for tuleap)



Update Tuleap:

$ dnf check-update tuleap*

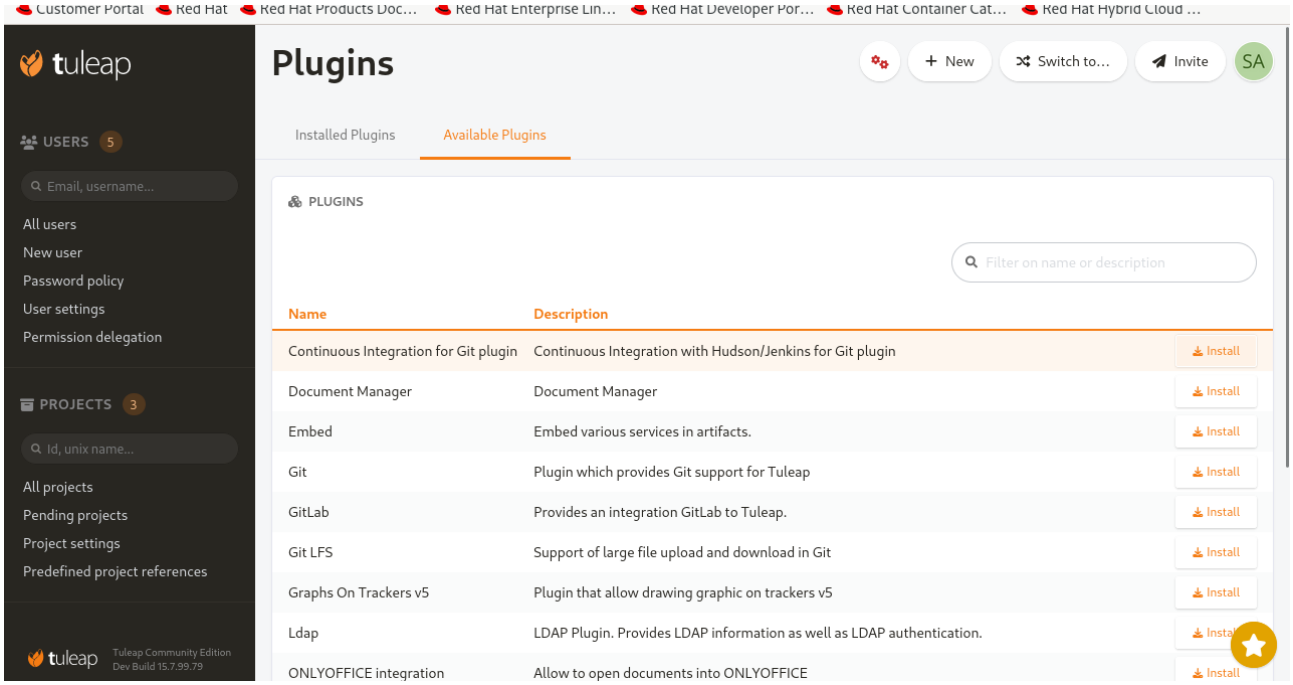
# Stop service
$ systemctl stop tuleap nginx

# Upgrade packages
$ dnf update

# Restart service
$ systemctl start tuleap nginx

Plugins enabled manually inside the tuleap:

Go to settings----manage all plugins----available plugins---enable all plugins



$ sudo systemctl restart tuleap
$ sudo systemctl start tuleap

Create new user account in tuleap:

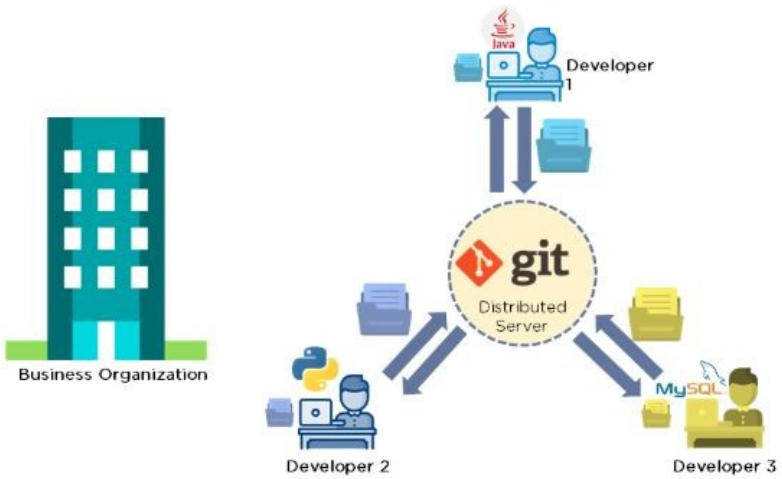
Go to settings--new user—create user
admin:
username:admin
passwd: cair@123

username:mah320
passwd: cair@123

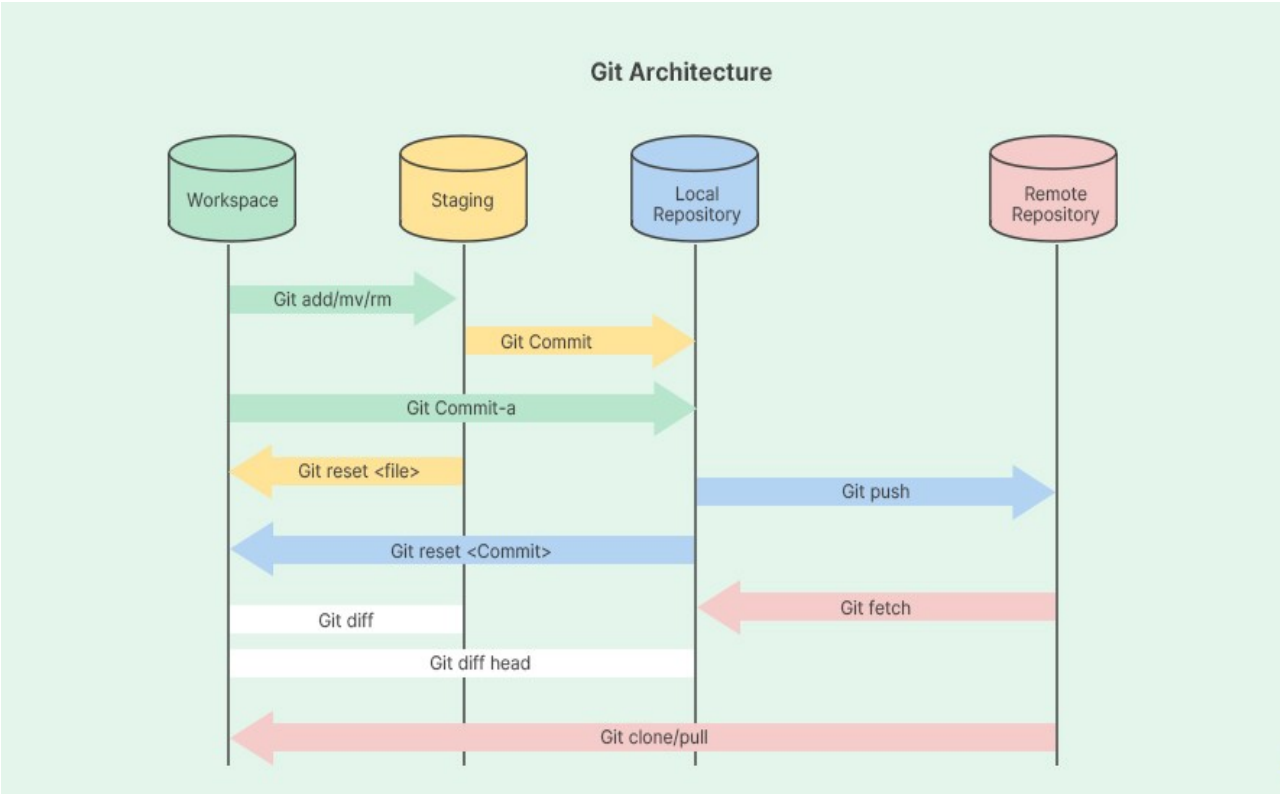
GIT Installation On Red Hat 9.4

GIT:

Git is a free and **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.

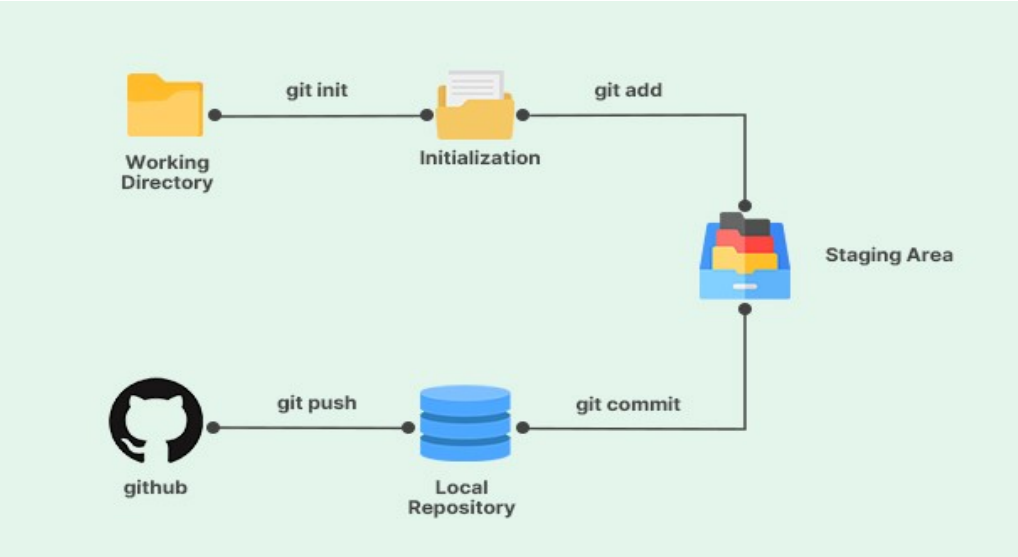


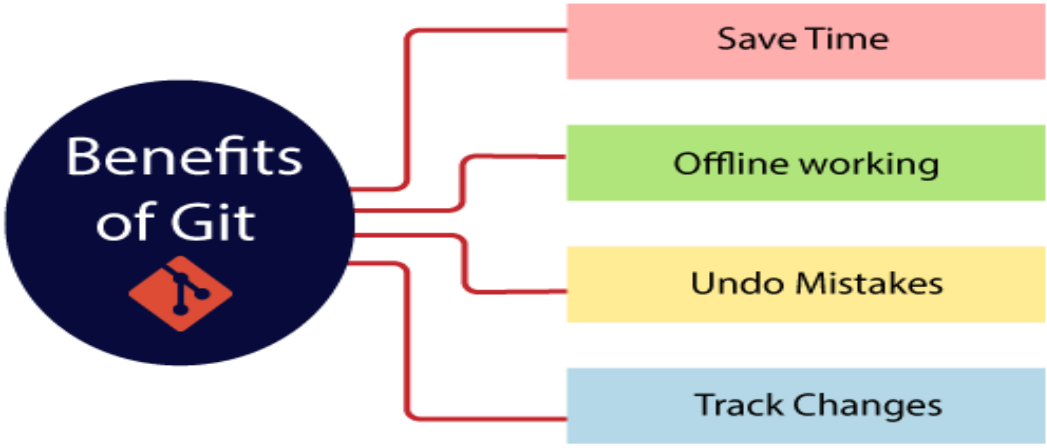
GIT Architecture:



Features of git

- Tracks history
- Free and open source
- Supports non-linear development
- Creates backups
- Scalable
- Supports collaboration
- Branching is easier
- Distributed development





Git Installation:

$ sudo yum install git -y

$ git --version

Configure the git in terminal:-

admin:

$ sudo git config --list
$ sudo git config --global user.name "admin"
$ sudo git config --global user.email "codendi-admin@tuleap.mytuleapserver.info"
$ sudo git config --global http.sslVerify false

user:

$ sudo git config --global user.name "mah320"
$ sudo git config --global user.email "maheshtemmanaboina@gmail.com"
$ sudo git config --global http.sslVerify false
$ sudo git config --list
$ git config --global --list

In-case any errors:

$ git remote remove origin
$ git remote set-url origin ssh://gitolite@tuleap.example.com/ros2-gtest/ros2_gtest.git
$ git remote add new-origin ssh://gitolite@tuleap.example.com/ros2-gtest/ros2_gtest.git
$ git remote -v
$ git config --global --unset user.name
git config --unset user.email
sudo git config --system --unset http.sslverify
git config –list

SSH setup for tuleap both admin and user:

User:

$ mkdir -p /home/cair/.ssh
$ cd /home/cair/.ssh
$ ls -al
$ ssh-keygen -t rsa -b 4096 -C "maheshtemmanaboina@gmail.com"
#id_rsa id_rsa.pub
$ eval $(ssh-agent)
#Agent pid 19609
$ ssh-add /home/cair/.ssh/id_rsa
#Identity added: /home/cair/.ssh/id_rsa (maheshtemmanaboina@gmail.com)

$ chmod 600 /home/cair/.ssh/id_rsa

$ cat id_rsa.pub ------(Add public key into a tuleap preferences location)
$ cd /var/lib/tuleap
$ ls -al
$ .gitconfig

#Add the user credentials E-mail and password

Admin optional:

admin:
$ /root/.ssh
$ ssh-keygen -t rsa -b 4096 -C "codendi-admin@tuleap.example.com"
$ eval $(ssh-agent)
$ ssh-add /root/.ssh/id_rsa
$ chmod 600 /root/.ssh/id_rsa
$ cat id_rsa.pub ------to tuleap section add
$ cd /var/lib/tuleap
$ ls -al
$ .gitconfig

Create a sample project in local terminal and try to push in tuleap ssh process:

$ mkdir my_project
$ cd my_project
$ git init
$ echo "Hello, Tuleap!" > README.md
$ git add README.md
$ git commit -m "Initial commit"
$ git remote add origin
https://tuleap.mytuleapserver.info/plugins/git/testproject/testproject.git
$ git push -u origin master (ssh or https)

Basic Git Commands:

Git Configuration & Setup

- git config –global user.name “Your Name” (`Set the global username`)
- git config –global user.email youremail@example.com
- git config --global –list (`Verify your global configuration`)
- git help
- man <command>

Initializing a Repository

- git init (`Initialize a new Git repository`)
- git clone <repository_url>

- git remote add <remote_name> <remote_url> (`Add a remote repository`
- git status
- git push -u <remote_name> <branch_name>

Git Commands

- git add <file>
- git add. or git add –all
- git status
- git status –ignored
- git diff
- git diff <commit1> <commit2>
- git diff –staged or git diff –cached
- git diff HEAD
- git commit
- git commit -m “<message>” or git commit –message “<message>”
- git commit -a or git commit –all
- git log
- git notes add
- git restore <file>
- git reset <commit>
- git reset –soft <commit>
- git reset –hard <commit>
- git mv

Branching and Merging

- git branch
- git branch <branch-name>
- git branch -d <branch-name>
- git branch -a
- git branch -r
- git checkout <branch-name>
- git checkout -b <new-branch-name>
- git checkout — <file>
- git merge <branch>
- git log
- git log <branch-d
- git log –all
- git stash
- git stash list
- git stash pop
- git tag
- git tag <tag-name>
- git tag <tag-name> <commit>
- git tag -a <tag-name> -m “<message>”

Remote Repositories

- git pull <remote_name> <branch_name>
- git pull –all **(Pull changes for all branches)**
- git pull origin main (eg)
- git pull --rebase <remote_name> <branch_name> **(Pull changes and rebase)**
- git pull --rebase origin main **(eg)**

- git remote show <remote_name> (**Show detailed information about a remote**)
- git pull –rebase
- git fetch <remote_name> <branch_name>
- git merge <remote_name>/<branch_name>
- git fetch origin main (eg)
- git merge origin/main (eg)
- git remote set-url --delete <remote_name> <url> **(Delete a remote URL for a remote name)**
- git push <remote> <branch>
- git push –all
- git push --all <remote_name> **(Push all branches to a remote repository)**
- git remote add <name> <url>
- git remote rename <old_name> <new_name> (Rename a remote repository)
- git remote remove <remote_name> (**Remove a remote repository)**

Git Comparison

- git show
- git show <commit>
- git diff
- git diff –staged
- git diff branch1..branch2
- git diff commit1 commit2
- git diff commit1 commit2 -- <file>
- git show <commit>
- git diff <branch>
- git diff main
- git diff origin/main

Jenkins Installation On Red Hat 9.4 local host machine
(By Default-HTTP)

What is Jenkins:

Jenkins is a tool that is used for automation. It is mainly an open-source server that allows all the developers to build, test and deploy software. It is written in Java and runs on java only. By using Jenkins we can make a continuous integration of projects(jobs) or end-to-endpoint automation.

or

Jenkins is an open source automation tool written in Java programming language that allows continuous integration.

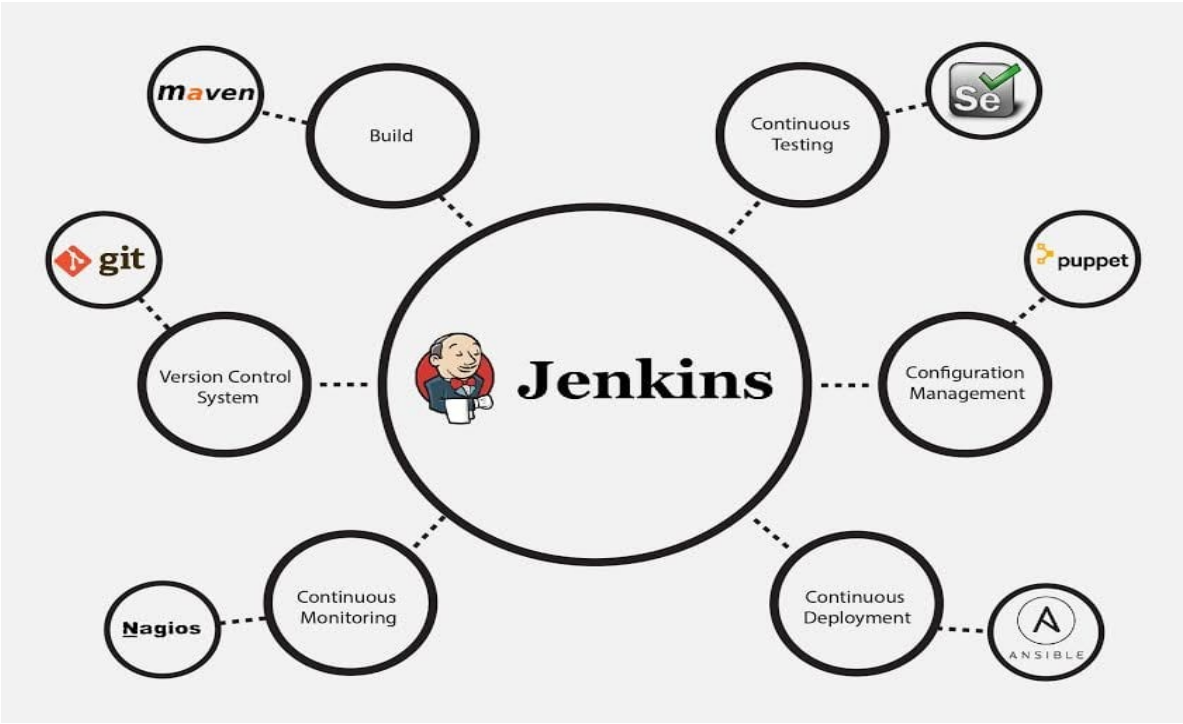
Jenkins History:

Jenkins is an open-source automation server initially developed as "**Hudson**" by Kohsuke Kawaguchi in 2004 at Sun Microsystems. It gained popularity for its ability to automate software build, test, and deployment processes.
In 2011, due to governance issues, Jenkins was forked from Hudson by Kawaguchi and the community. This move led to rapid community adoption and development under the Jenkins name.

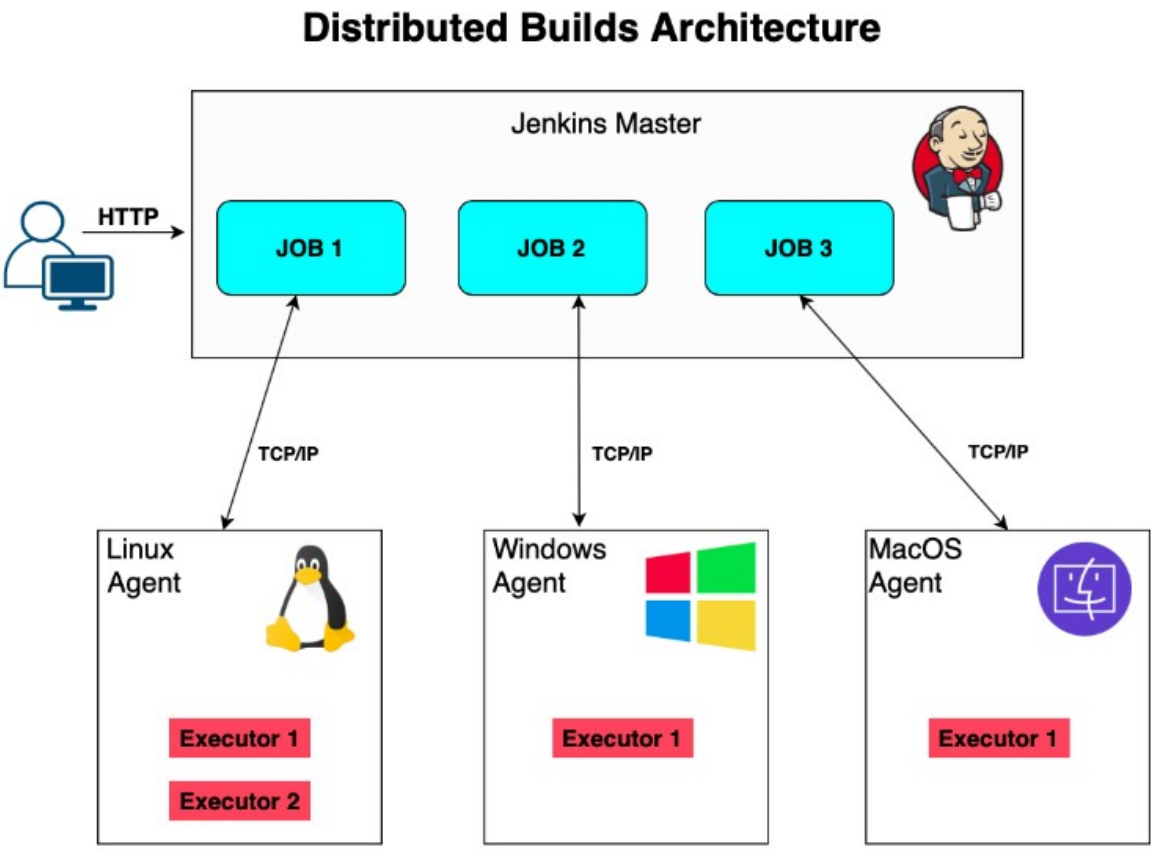
Since then, Jenkins has become a leading tool for Continuous Integration (CI) and Continuous Deployment (CD), known for its flexibility, extensive plugin support, and user-friendly interface.

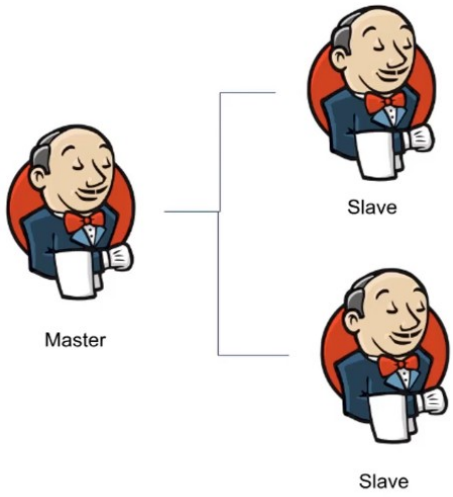
Today, Jenkins continues to evolve with regular updates and a strong community, supporting organizations worldwide in automating their software development workflows.

Jenkins Plugins Architectures:



Jenkins Master and Slave Architecture:





Jenkins Master:

- Scheduling jobs
- Communicating with the slaves
- Monitor the slaves
- Present the results
- Master can also execute build jobs

Jenkins Slave:

- Communicates with Jenkins Master
- Can run in different OS
- Execute jobs
- Flexibility

Jenkins Master:

1. **Responsibilities:**
    - Manages the Jenkins environment, schedules jobs, and coordinates builds.
    - Stores configurations, job definitions, build logs, and artifacts.
2. **Key Functions:**
    - **Job Management:** Configures and manages jobs.
    - **Build Coordination:** Distributes build tasks to agents.
    - **User Interface:** Web-based UI for configuration and monitoring.
    - **Plugins:** Extends functionality and integrates with external tools.
3. **Security:**
    - Controls access and permissions, supports authentication mechanisms.

Jenkins Agent (Slave):

1. **Responsibilities:**
    - Executes build jobs dispatched by the master.
    - Runs on different platforms (Windows, Linux, macOS).
2. **Types:**
    - **Dedicated Agents:** Assigned to specific jobs.
    - **Shared Agents:** Used for general build tasks.
3. **Communication:**
    - Communicates with the master over the network, receives build instructions, executes tasks, and reports back.
4. **Configuration:**
    - Labeled with tags to describe capabilities and environments.
    - Jobs can target agents with specific labels.

Simple Example:

- **Master:** Like a manager who assigns tasks.
- **Agents (Slaves):** Like workers who perform the tasks.

Advantages of Using Jenkins Master with Slaves

1. **Scalability:**
    - **Handle More Jobs:** Add more agents (slaves) to run more jobs at the same time.
    - **Spread the Load:** Distribute work across multiple machines to avoid overloading one.

2. **Efficiency:**
    - **Use Resources Wisely:** Spread tasks across multiple machines to use your hardware better.
    - **Faster Builds:** Run multiple builds and tests in parallel to save time.
3. **Flexibility:**
    - **Different Environments:** Run jobs on different operating systems (Windows, Linux, macOS).
    - **Specialized Tasks:** Use agents with specific tools or settings needed for certain jobs.
4. **Reliability:**
    - **Less Downtime:** If one agent fails, others can take over, ensuring continuous operation.
    - **Consistent Performance:** Distribute tasks to prevent any single machine from becoming a bottleneck.
5. **Cost-Effective:**
    - **Optimized Use:** Use a network of less powerful machines instead of relying on one high-performance machine.
    - **Scale as Needed:** Easily add or remove agents based on workload demands.

Work Flow of master and slave:

Setting Up Jenkins Master-Slave (Red Hat Linux to Windows)

- Install Java Development Kit (JDK) on Windows
- Download Jenkins Agent (agent.jar)
- From the Jenkins master's web interface, download the agent JAR (agent.jar) specific to your Jenkins version. The URL is typically http://<JENKINS_MASTER_IP>:8080/jnlpJars/agent.jar
- Create a directory on the Windows machine where you'll run the Jenkins agent (e.g., `C:\Jenkins`)
- Add Windows Node (Slave) in Jenkins
- In Jenkins web interface (Manage Jenkins > Manage Nodes and Clouds), click New Node
- Enter a name for the node (e.g., Windows-Slave) and select Permanent Agent. Click OK
- Configure the node:

- **Remote Root Directory:** Specify the directory path on the Windows machine where Jenkins will execute jobs (e.g., C:\Jenkins).
- **Launch Method:** Choose Launch agent via Java Web Start.
- **Labels:** (Optional) Add labels if needed for job assignment.
- **Usage:** Select Use this node as much as possible.

- Save the node configuration

- Launch Jenkins Agent on Windows Slave
- On the Windows machine where you placed agent.jar (e.g., C:\Jenkins), open Command Prompt as administrator
- Navigate to the directory where agent.jar is located
- java -jar agent.jar -jnlpUrl http://<JENKINS_MASTER_IP>:8080/computer/<SLAVE_NAME>/slave-agent.jnlp -secret <SECRET_KEY>
- Verify Connection in jenkins master.

Why Organizations use Jenkins:

Jenkins facilitates the automation of several stages of the software development life cycle, including application development, testing, and deployment. Operating within servlet containers like Apache Tomcat, the technology is server-based. Continuous delivery (CD) and integration (CI) pipelines can be created and managed with Jenkins. The development, testing, and deployment of software applications are automated using CI/CD pipelines. You will be able to release software more regularly and with fewer problems if you do this.

1. Jenkins is flexible.
2. You can add the n no.of plugins you want to add to the Jenkins.
3. You can automate the process of CI/CD pipelines of all the projects.

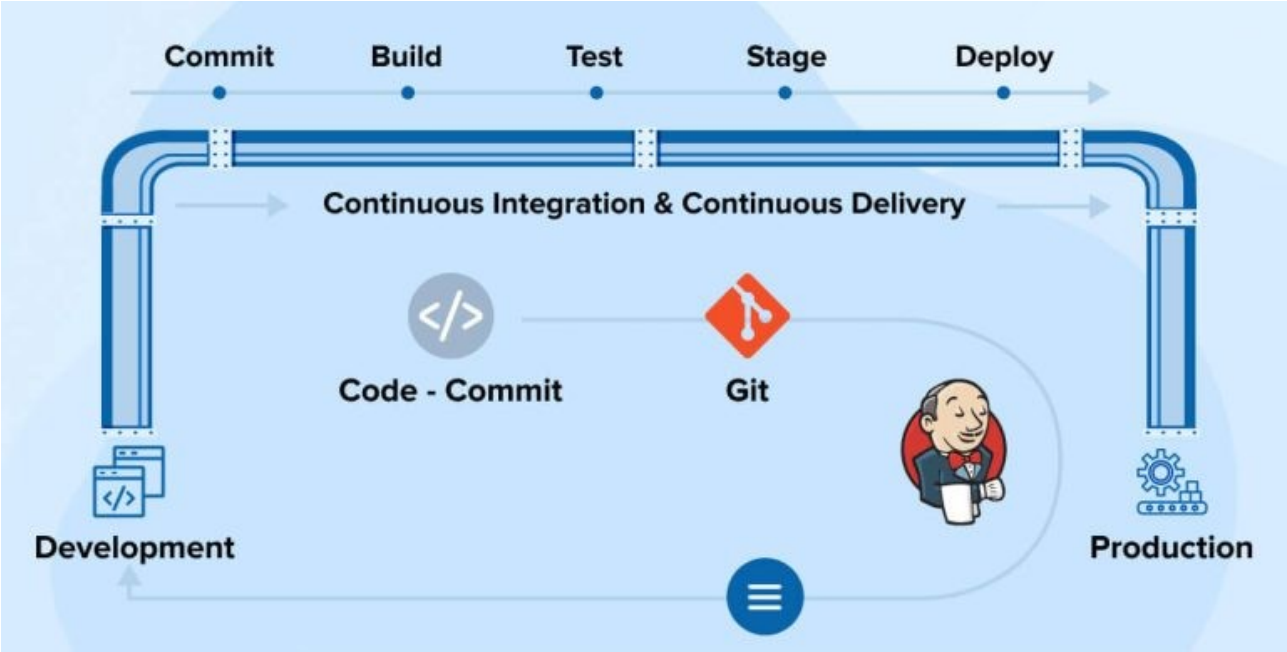
What is Jenkins CI/CD Pipeline:

Jenkins CI/CD stands for Continuous Integration / Continuous Deployment first let us try to understand what is a pipeline. In computing, a pipeline is a set of stages or processes linked together to form a processing system. Each stage in the pipeline takes an input, processes it in accordance with a set of rules, and then sends the outputs to the stage that follows. Frequently, the pipeline's overall output is its final step's output.

or

A **Jenkins CI/CD pipeline** is a set of automated workflows that allow teams to efficiently and consistently build, test, and deploy software applications. It encompasses the entire software delivery process, from code commit to production deployment, ensuring that changes are integrated smoothly and reliably.

Architecture of CI/CD Pipeline:



What is Jenkins Continuous Integration (CI):

Jenkins Continuous integration means whenever new code is committed to remote repositories like Tuleap, GitHub, GitLab, etc. Continuous Integration (CI) will continuously build, tested, and merged into a shared repository.

Or

Continuous Integration (CI) is a software development practice where developers integrate code into a shared repository frequently, usually several times a day.

- **Build:** Automatically compile source code into executable artifacts.
- **Test:** Run automated tests (unit tests, integration tests, etc.) to verify code quality and functionality.
- Bugs can be found quickly

What is Jenkins Continuous Deployment (CD):

- **Continuous Deployment (CD)** with Jenkins automates the entire software release process, from code commit to production deployment, without requiring manual intervention once changes pass all automated tests. This approach allows for rapid and reliable delivery of software updates to end-users.

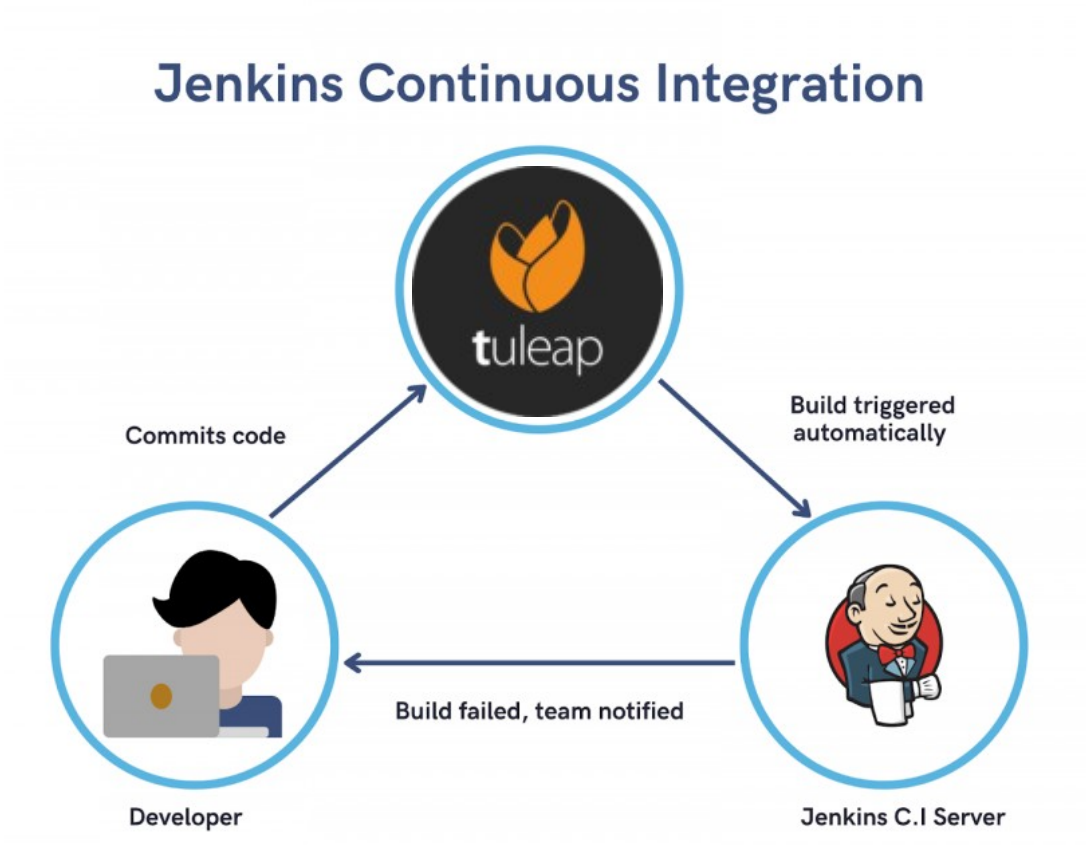
Or

Continuous Deployment (CD): Automates the entire pipeline, including deployment to production, once changes pass automated tests, enabling rapid and frequent releases.

Example Workflow:CD

- **Commit:** A developer pushes code to the Git repository.
- **Trigger:** Jenkins detects the commit and starts the pipeline.
- **Build:** Jenkins compiles the code and creates a Docker image.
- **Test:** Jenkins runs unit tests, integration tests, and other automated tests.
- **Deploy:** Jenkins deploys the Docker image to the staging environment.
- **Verify:** Automated tests run in staging to verify the deployment.
- **Promote:** Jenkins automatically promotes the Docker image to production if all tests pass.

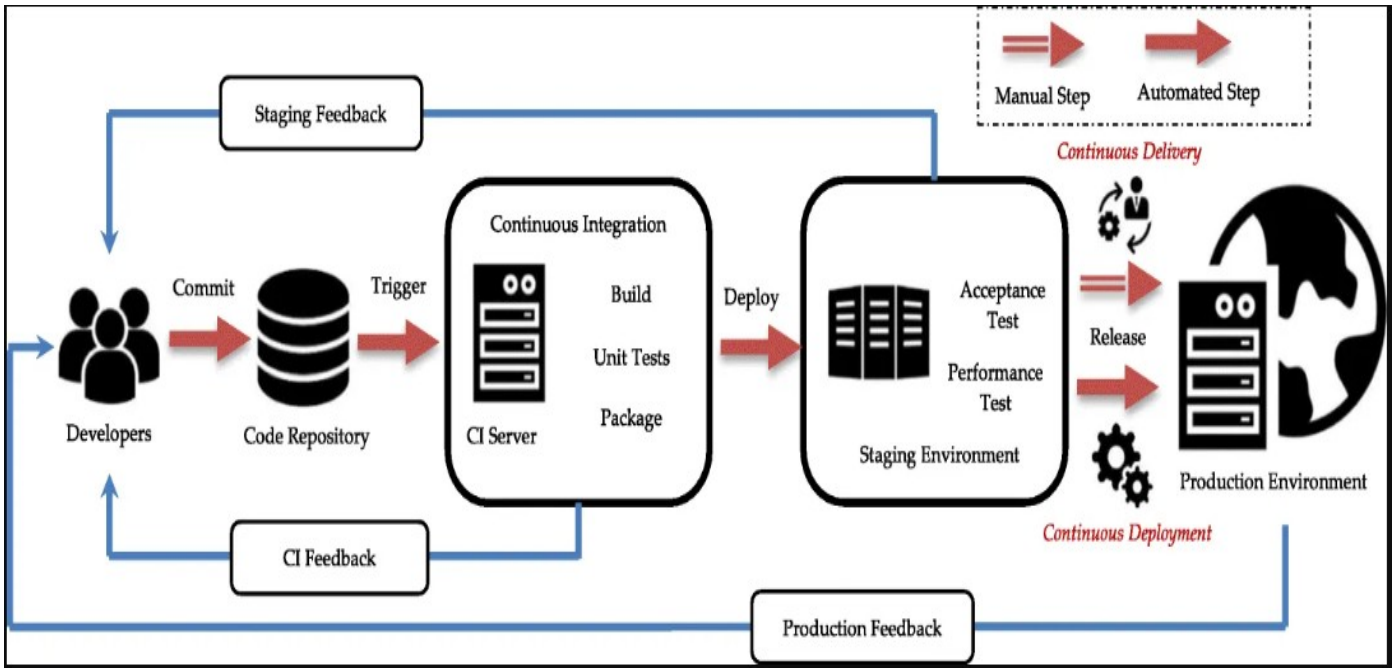
Architecture of Continuous integration (CI):



Advantages:

- **Speed:** Rapid delivery of new features and bug fixes to production.
- **Automation:** Reduces manual intervention, minimizing human errors.
- **Consistency:** Ensures a consistent deployment process across environments.
- **Feedback:** Provides quick feedback to developers, enabling faster iterations.

Architecture of Continuous Deployment (CD):



What is Jenkins Pipeline

DevOps professionals mostly work with pipelines because pipelines can automate the processes like building, testing, and deploying the application. Doing manually by UI takes lots of time and effort which will affect productivity. With the help of Continuous Integration / Continuous Deployment (CI/CD) Pipeline scripts we can automate the whole process which will increase productivity and save lots of time for the organization and can deliver quality applications to the end users.

Or

Jenkins Pipeline is a suite of plugins that supports implementing and integrating continuous delivery pipelines into Jenkins. It allows you to define your build, test, and deployment processes as code using a `Jenkins file`. Pipelines can be simple or complex, enabling automation of tasks from code integration to production deployment.

Key Features of Jenkins Pipeline:

Pipeline as Code:

- Define your entire CI/CD process in a `Jenkins file`, which can be version-controlled alongside your source code.

Declarative and Scripted Syntax:

- Pipelines can be written in two syntax's: Declarative (simpler, more structured) and Scripted (more flexible, powerful).

Stages and Steps:

- Pipelines consist of stages that group build steps, allowing for better organization and visualization of the pipeline flow.

Parallel Execution:

- Supports parallel execution of tasks, speeding up the overall pipeline by running independent stages concurrently.

Integration with SCM:

- Easily integrates with source control management tools like Git, SVN, etc., to trigger builds based on code changes.

Example of pipeline:

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any ❶
    stages {
        stage('Build') { ❷
            steps {
                // ❸
            }
        }
        stage('Test') { ❹
            steps {
                // ❺
            }
        }
        stage('Deploy') { ❻
            steps {
                // ❼
            }
        }
    }
}
```

Jenkinsfile (Scripted Pipeline)

```
node { ❶
    stage('Build') { ❷
        // ❸
    }
    stage('Test') { ❹
        // ❺
    }
    stage('Deploy') { ❻
        // ❼
    }
}
```

Jenkins Installation on Red Hat 9.4 local host machine:

Long Term Support Release(LTS)

Official Website:https://www.jenkins.io/doc/book/installing/linux/#red-hat-centos

Commands:

Jenkins Red Hat Packages:

To use this repository, run the following command:

$ 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

Install Java 17:

$ sudo yum install java-17-openjdk-devel
$ sudo update-alternatives --config java
$ ls /usr/lib/jvm/java-17-openjdk-17.0.11.0.9-2.el9.x86_64/bin/javac
$ source ~/.bashrc
$ javac --version
$ java --version

Install Jenkins:

$ sudo yum install jenkins
$ sudo systemctl daemon-reload

Start Jenkins service:

# Check Jenkins enable
$ sudo systemctl enable jenkins

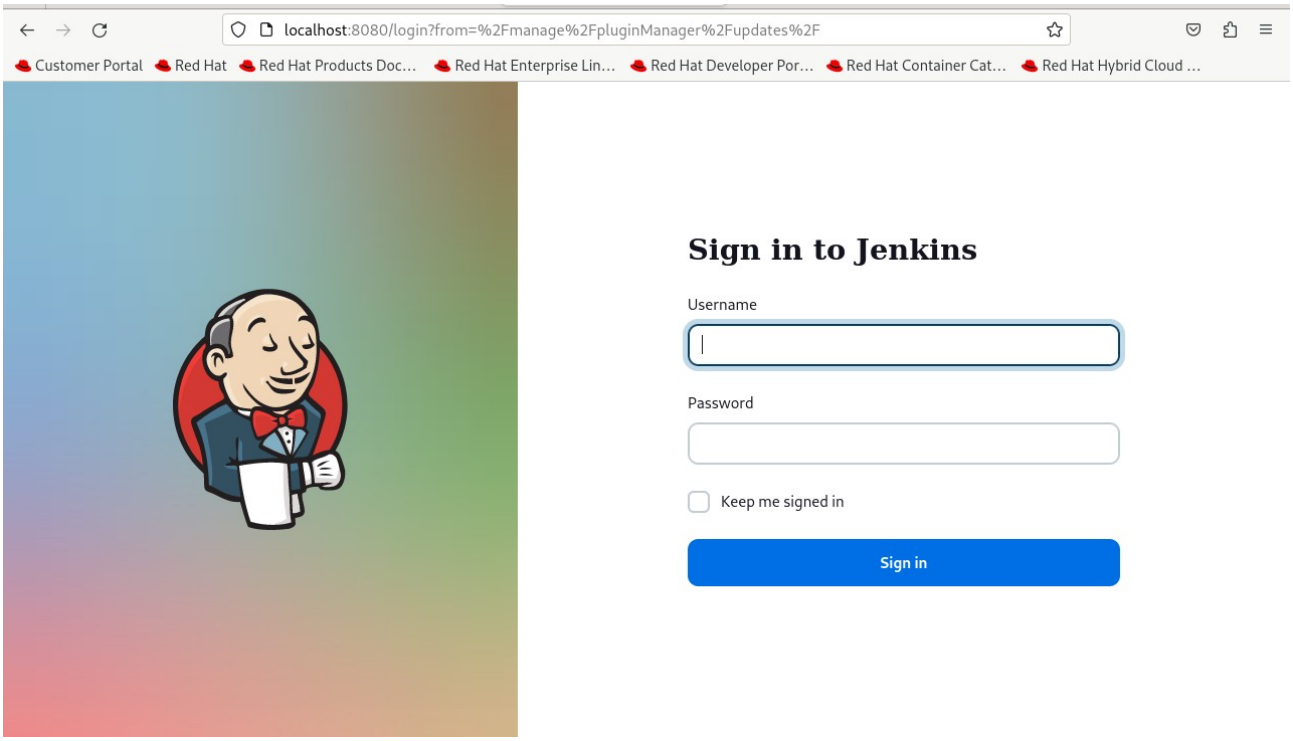
# Check Jenkins start
$ sudo systemctl start jenkins.service

or
# Check Jenkins start
$ sudo systemctl start jenkins

# Restart Jenkins
$ sudo systemctl restart jenkins

# Check Jenkins status
$ sudo systemctl status jenkins
$ systemctl status jenkins.service
$ journalctl -xeu jenkins.service

JENKINS DEFAULT HTTP:



Configuring Custom SSL/TLS Certificates For (Red Hat) Tuleap/Jenkins (HTTPS)

Red hat-9.4-Configuring custom SSL/TLS certificates on red hat machine:

Official Website:https://docs.redhat.com/en/documentation/red_hat_openstack_platform/15/html/director_installation_and_usage/configuring-custom-ssl-tls-certificates

Definition: The Red Hat Open Stack Platform director is a tool set for installing and managing a complete
Open Stack environment.

UNDERCLOUD: The under cloud is the main management node that contains the Open Stack Platform director tool set.

Official Website: https://docs.redhat.com/en/documentation/red_hat_openstack_platform/15/html/director_installation_and_usage/configuring-custom-ssl-tls-certificates

Initializing the signing host:

$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt

echo '1000' | sudo tee /etc/pki/CA/serial

Creating a certificate authority (CA):

The process described above is for setting up a Certificate Authority (CA) on a Red Hat Enterprise Linux system. A Certificate Authority is an entity responsible for issuing digital certificates. These certificates are used to verify the identity of websites, individuals, and organizations, ensuring secure, encrypted communication over networks like the internet.

Create a directory in user :

$ mkdir server
$ cd server

$ openssl genrsa -out ca.key.pem 4096

#ca.key.pem

$ openssl req -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem

#Modify the below content
#Country Name (2 letter code) [XX]:IN
State or Province Name (full name) []:Karnataka
Locality Name (eg, city) [Default City]:Bangalore
Organization Name (eg, company) [Default Company Ltd]: Red Hat
Organizational Unit Name (eg, section) []:Red Hat
Common Name (eg, your name or your server's hostname) []:10.0.2.15
Email Address []:maheshtemmanaboina@gmail.com

#ca.crt.pem

Adding the certificate authority to clients:

$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
$ sudo update-ca-trust

Creating an SSL/TLS key:

$ openssl genrsa -out server.key.pem 2048

#server.key.pem

Creating an SSL/TLS certificate signing request:

$ cp /etc/pki/tls/openssl.cnf .
$ cd /etc/pki/tls
$ sudo nano openssl.cnf

Edit the new openssl.cnf file and configure the SSL parameters to use for the director

#stages:

[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req

[req_distinguished_name]

countryName = Country Name (2 letter code)
countryName_default = IN
stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Karnataka
localityName = Locality Name (eg, city)
localityName_default = Bangalore
organizationalUnitName = Organizational Unit Name (eg, section)

```
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 10.0.2.15   #ip addres of the linux server
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names


[alt_names]
IP.1 = 10.0.2.15
DNS.1 = tuleap.example.com
DNS.2 = jenkins.example.com
IP.2  = 10.0.2.15
```

save and exit!

Run the following below command to generate a certificate signing request (server.csr.pem):

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem

#verify the details
Country Name (2 letter code) [XX]:IN
State or Province Name (full name) []:Karnataka
Locality Name (eg, city) [Default City]:Bangalore
Organization Name (eg, company) [Default Company Ltd]:Red Hat
Organizational Unit Name (eg, section) []:Red Hat
Common Name (eg, your name or your server's hostname) []:10.0.2.15
Email Address []:

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:cair123
An optional company name []:

#server.csr.pem
```

Creating the SSL/TLS certificate:

```
$ cd server
# ca.crt.pem  ca.key.pem  openssl.cnf  server.csr.pem  server.key.pem
```

$ sudo mkdir /etc/pki/CA/newcerts
$ sudo chmod 700 /etc/pki/CA/newcerts
$ sudo chown root:root /etc/pki/CA/newcerts

Run the following below command to create a certificate for your undercloud or overcloud:

$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem

Output:

Check that the request matches the signature
Signature ok
Certificate Details:
    Serial Number: 4096 (0x1000)
    Validity
      Not Before: Jun 24 17:26:37 2024 GMT
      Not After : Jun 22 17:26:37 2034 GMT
    Subject:
      countryName   = IN
      stateOrProvinceName   = Karnataka
      organizationName   = Red Hat
      organizationalUnitName   = Red Hat
      commonName   = 10.0.2.15
    X509v3 extensions:
      X509v3 Basic Constraints:
        CA:FALSE
      X509v3 Key Usage:
        Digital Signature, Non Repudiation, Key Encipherment
      X509v3 Subject Alternative Name:
        DNS:tuleap.example.com, DNS:jenkins.example.com, DNS:10.0.2.15
Certificate is to be certified until Jun 22 17:26:37 2034 GMT (3650 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated

server.crt.pem

Adding the certificate to the undercloud:

Run the following command to combine the certificate and key:

$ cat server.crt.pem server.key.pem > undercloud.pem
$ ls

ca.crt.pem openssl.cnf server.csr.pem undercloud.pem
ca.key.pem server.crt.pem server.key.pem

Create a Directory and Copy the Files into a target location:

$ sudo mkdir /etc/pki/undercloud-certs
$ sudo cp undercloud.pem /etc/pki/undercloud-certs/.

Set SELinux Context:

$ sudo semanage fcontext -a -t etc_t "/etc/pki/undercloud-certs(/.*)?"
$ sudo restorecon -R /etc/pki/undercloud-certs

Add the undercloud.pem file location to the undercloud_service_certificate option in the undercloud.conf file:

$ cd /etc/pki/undercloud-certs

$ sudo nano undercloud.conf

undercloud_service_certificate = /etc/pki/undercloud-certs/undercloud.pem

Update Trusted Certificate Authorities:

Ensure you add the certificate authority that signed the certificate to the undercloud's list of trusted Certificate Authorities so that different services within the undercloud have access to the certificate authority

$ sudo cp -r server.crt.pem /etc/pki/ca-trust/source/anchors/
#ca.crt.pem server.crt.pem

$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
$ sudo update-ca-trust

$ reboot the system! Or init 6

Configuring Jenkins and Tuleap with Nginx Web Server (HTTP TO HTTPS)

What is a server:

A **server** typically refers to a computer or a software system that provides resources, services, or functionality to other computers, devices, or users within a network.

Server Components:

Hardware Components:

- **CPU**: Processes instructions and calculations.
- **RAM**: Provides temporary storage for quick access by the CPU.
- **Storage**: Stores data permanently (e.g., HDD, SSD).
- **Network Interface**: Connects the server to networks.

Software Components:

- **Operating System**: Manages hardware resources and runs server software.
- **Server Software**: Provides specific services (e.g., web hosting, databases).
- **Security Software**: Protects against threats like unauthorized access.

Networking Components:

- **Network Interface Cards**: Connect servers to networks.
- **Switches and Routers**: Manage network traffic between servers and devices.

Management and Monitoring Tools:

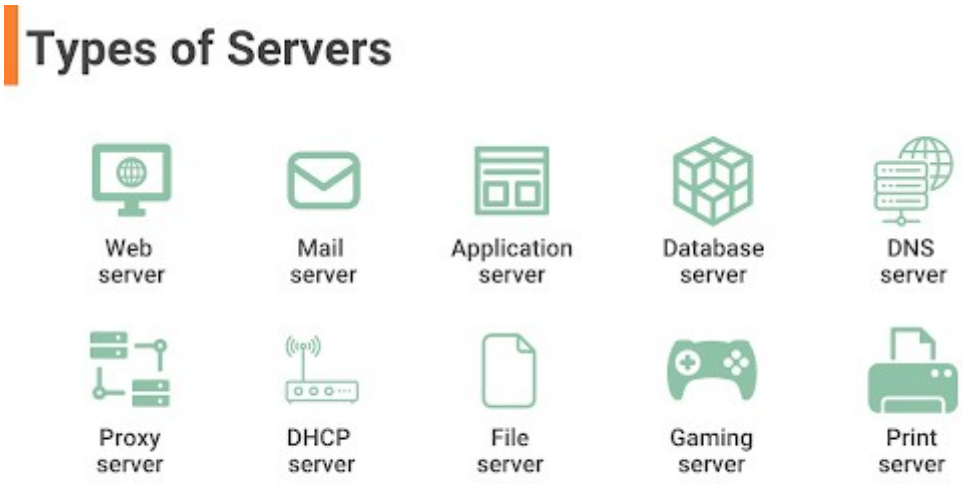
- **Monitoring**: Tracks server performance (e.g., CPU usage, network traffic).
- **Management**: Configures and maintains server settings and software.

Backup and Security Measures:

- **Backup Systems**: Safeguards data by creating copies.
- **Security**: Uses firewalls and encryption to protect data and systems.

Types of Servers:

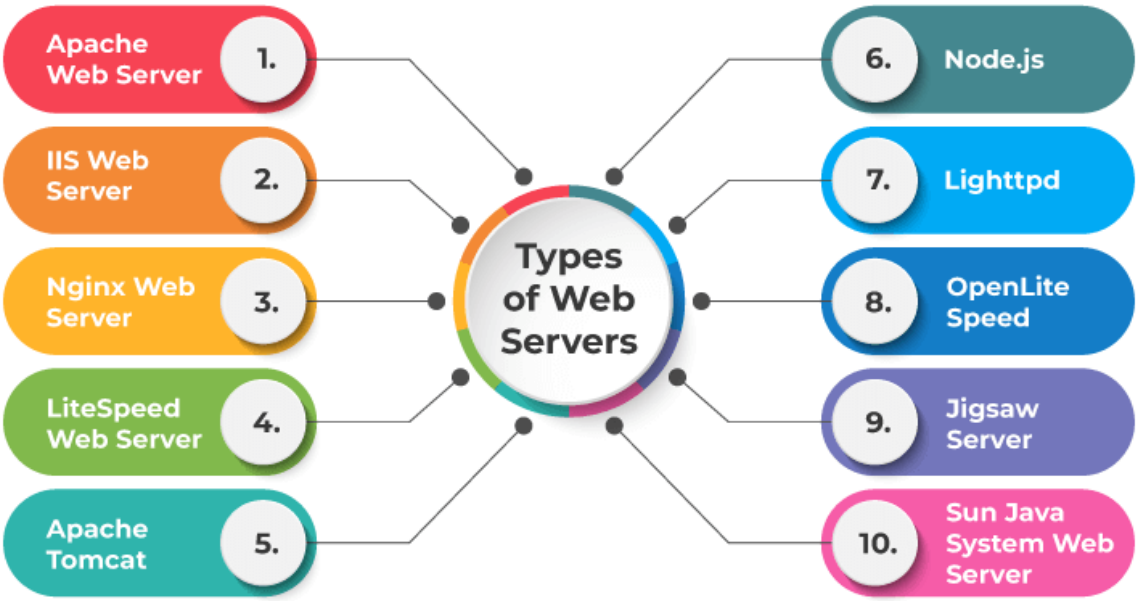
servers can perform multiple jobs, such as hosting websites, sending and receiving emails, safeguarding internal networks, etc.



What is a Web server:

Web servers are designed to run websites and apps through client programs (web browsers) such as Internet Explorer, Chrome, Firefox, Opera, or Safari. They are responsible for storing, processing, and delivering web content to users. They support protocols such as HTTP, FTP, and SMTP that are key to information exchange between network devices.

Types of Web Servers:



Apache Web Server:

Apache web server is one of the most popular web servers developed by the Apache Software Foundation. Open source software, Apache supports almost all operating systems such as Linux, Windows, Unix FreeBSD, Mac OS X and more. Approximately, 60% of the machines run on Apache Web Server.

Nginx Web Server:

NGINX is open-source web server software used for reverse proxy, load balancing, and caching. It provides HTTPS server capabilities and is mainly designed for maximum performance and stability. It also functions as a proxy server for email communications protocols, such as IMAP, POP3, and SMTP.

Difference:

Apache	NGINX
Apache runs on all Unix like systems such as Linux, BSD, etc. as well as completely supports Windows.	Nginx runs on modern Unix like systems; however it has limited support for Windows.
Apache uses a multi-threaded approach to process client requests.	Nginx follows an event-driven approach to serve client requests.
Apache cannot handle multiple requests concurrently with heavy web traffic.	Nginx can handle multiple client requests concurrently and efficiently with limited hardware resources.
Apache processes dynamic content within the web server itself.	Nginx can't process dynamic content natively.
Apache is designed to be a web server.	Nginx is both a web server and a proxy server.
Modules are dynamically loaded or unloaded, making it more flexible.	Since modules cannot be loaded dynamically, they must be compiled within the core software itself.
A single thread can only process one connection.	A single thread can handle multiple connections.
The performance of Apache for static content is lower than Nginx.	Nginx can simultaneously run thousands of connections of static content two times faster than Apache and uses little less memory.

Nginx is often preferred. For flexibility, dynamic content, and extensive module support

Reverse Proxy: Often used as a reverse proxy and load balancer due to its efficiency in handling concurrent connections.

Setup tuleap.conf in nginx--1

```
$ cd /etc/nginx/conf.d

$ nano tuleap.conf   # Edit the file

upstream tuleap-apache {
    server 127.0.0.1:8080;
}

upstream tuleap-php-fpm {
    server 127.0.0.1:9000;
}

upstream tuleap-php-fpm-long-running-request {
    server 127.0.0.1:9002;
}

server {
    listen       443 ssl http2;
    listen       [::]:443 ssl http2;
    server_name  tuleap.example.com;

    ssl_certificate /home/cair/server/server.crt.pem;
    ssl_certificate_key /home/cair/server/server.key.pem;
    ssl_session_timeout 1d;
    ssl_session_cache shared:MozSSL:10m;
    ssl_session_tickets off;

    # Tweak to your needs
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-
SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;

    # Tweak for file upload and SVN
    client_max_body_size 256M;

    include conf.d/tuleap.d/*.conf;
}

server {
    listen       80;
    listen       [::]:80;
    server_name  tuleap.example.com;
```

```
    location / {
      return 301 https://$server_name$request_uri;
    }
}
```

save and exit!

Setup Jenkins.conf in nginx—2

$ nano jenkins.conf

```
server {
    listen 8443 ssl;
    server_name jenkins.example.com;

    ssl_certificate /home/cair/server/server.crt.pem;
    ssl_certificate_key /home/cair/server/server.key.pem;

    location / {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port $server_port;
        proxy_redirect http://localhost:8080 https://jenkins.example.com;
    }

    location /websocket {
        proxy_pass http://localhost:8080;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

save and exit!

Restart the services:

$ nginx -t (ok)

 # nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
$sudo systemctl restart nginx

Convert PEM files to PKCS12 Format:

 # openssl command to convert your certificate and key files to a PKCS12 format

$ sudo openssl pkcs12 -export -in /home/cair/server/server.crt.pem -inkey /home/cair/server/server.key.pem -out /home/cair/server/jenkins.p12 -name jenkins -CAfile /home/cair/server/ca.crt.pem -caname root -password pass:cair123

Convert PKCS12 to JKS Format:

 # keytool command to convert the PKCS12 file to a JKS file.

$ sudo keytool -importkeystore -deststorepass cair123 -destkeypass cair123 -destkeystore /var/lib/jenkins/jenkins.jks -srckeystore /home/cair/server/jenkins.p12 -srcstoretype PKCS12 -srcstorepass cair123 -alias jenkins

Set the Permissions:

$ sudo chown jenkins:jenkins /var/lib/jenkins/jenkins.jks
$ sudo chmod 640 /var/lib/jenkins/jenkins.jks

Configure Jenkins to Use the Keystore:

$ sudo nano /etc/sysconfig/jenkins

JENKINS_ARGS="--webroot=/var/cache/$NAME/war --httpPort=-1 --httpsPort=8443 --httpsKeyStore=/var/lib/jenkins/jenkins.jks --httpsKeyStorePassword=cair123"

save and exit!

Add Certificates path both Tuleap and Jenkins into java trust store:

$ sudo keytool -import -alias tuleap -keystore /usr/lib/jvm/java-17-openjdk-17.0.11.0.9-2.el9.x86_64/lib/security/cacerts -file /home/cair/server/server.crt.pem -storepass changeit -noprompt

$ sudo keytool -import -alias jenkins -keystore /usr/lib/jvm/java-17-openjdk-17.0.11.0.9-2.el9.x86_64/lib/security/cacerts -file /home/cair/server/server.crt.pem -storepass changeit -noprompt

Verify the certificates:

$ sudo keytool -list -keystore /usr/lib/jvm/java-17-openjdk-17.0.11.0.9-2.el9.x86_64/lib/security/cacerts -storepass changeit | grep tuleap

$ sudo keytool -list -keystore /usr/lib/jvm/java-17-openjdk-17.0.11.0.9-2.el9.x86_64/lib/security/cacerts -storepass changeit | grep jenkins

Firewall allow for Jenkins:

$ sudo firewall-cmd --permanent --add-port=8443/tcp
$ sudo firewall-cmd --reload
$ sudo firewall-cmd --list-all

 # Incase the server will not run increase the time

$ sudo nano /lib/systemd/system/jenkins.service
[Unit]
Description=Jenkins Continuous Integration Server
Requires=network.target
After=network.target

[Service]
Type=notify
NotifyAccess=main
ExecStart=/usr/bin/jenkins
TimeoutStartSec=300

save and exit!

Restart the all Servers:

$ sudo systemctl restart jenkins
$ sudo systemctl restart tuleap
$ sudo systemctl start jenkins
$ sudo systemctl start tuleap
$ sudo systemctl start nginx

SSL/TLS Certificates verification both tuleap and jenkins:

$ openssl s_client --connect tuleap.example.com:443
$ openssl s_client --connect jenkins.example.com:8443

#restart system once(main)

$ nano /etc/hosts:

ifconfig (system ip address local)

127.0.0.1 tuleap.example.com
127.0.0.1 jenkins.example.com

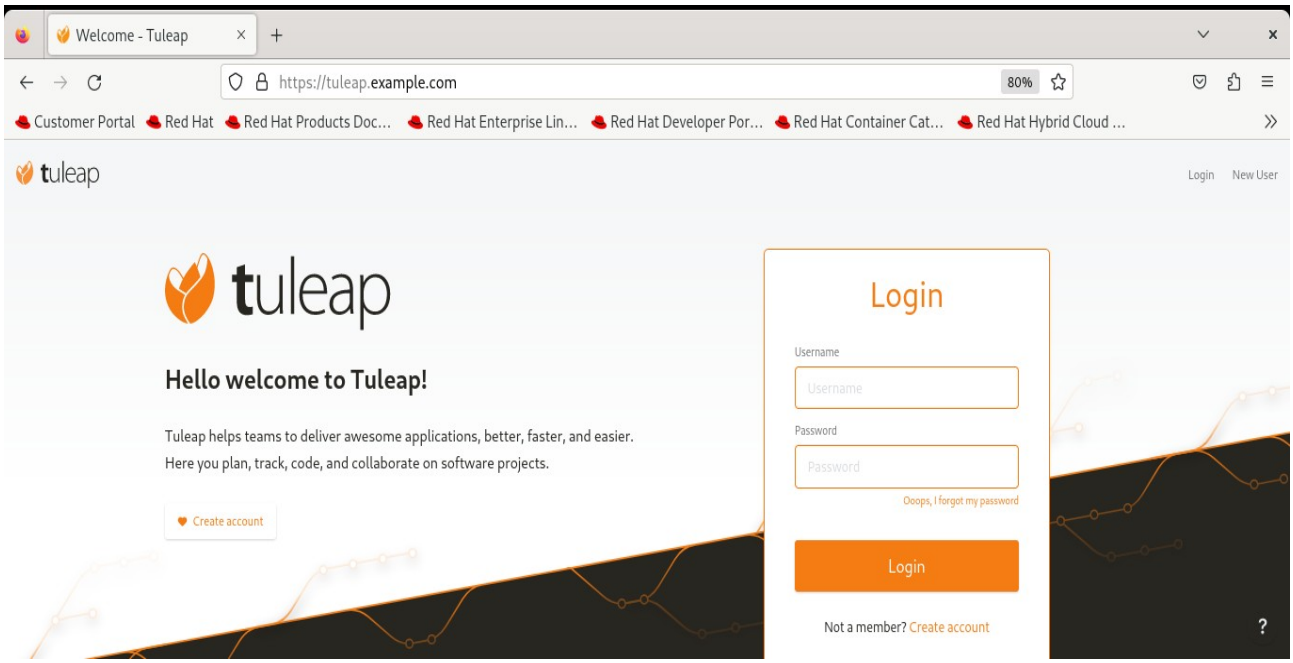
Browser access both servers:

https://tuleap.example.com or localhost
https://jenkins.example.com:8443

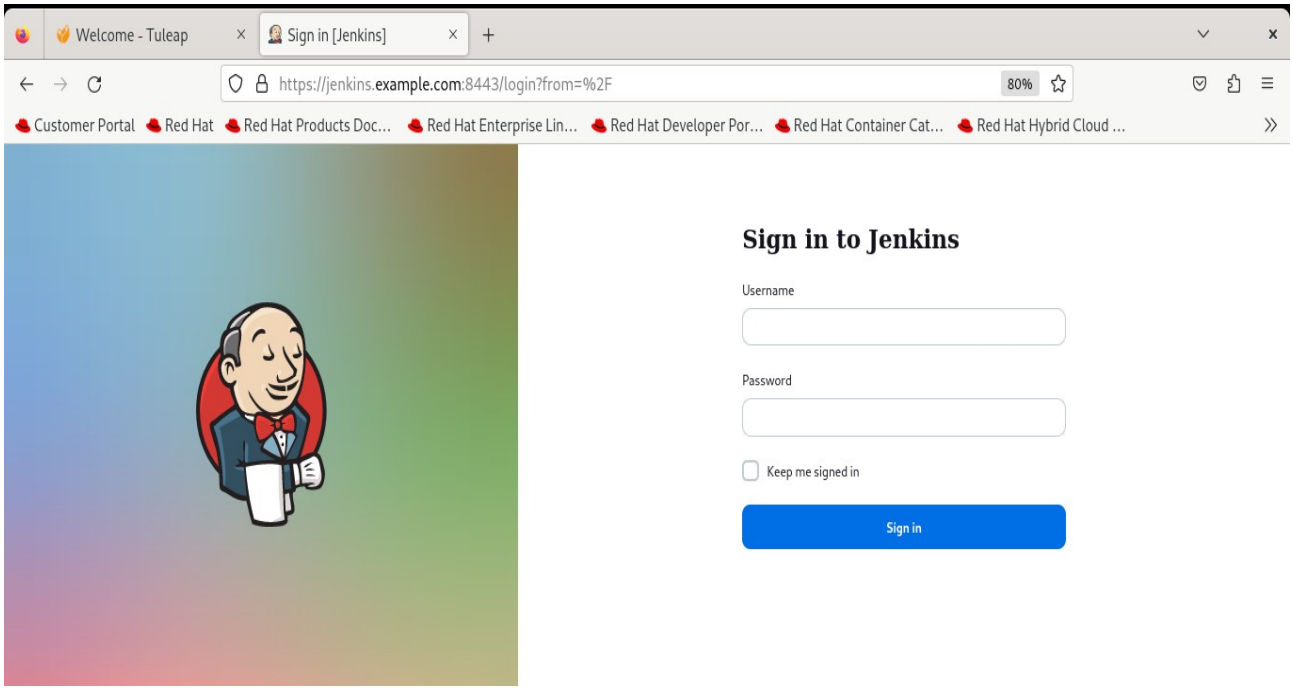
Both servers ip is 127.0.0.1 (loopback ip address)-localhost

In browser verify the padlock of both Jenkins and Tuleap:

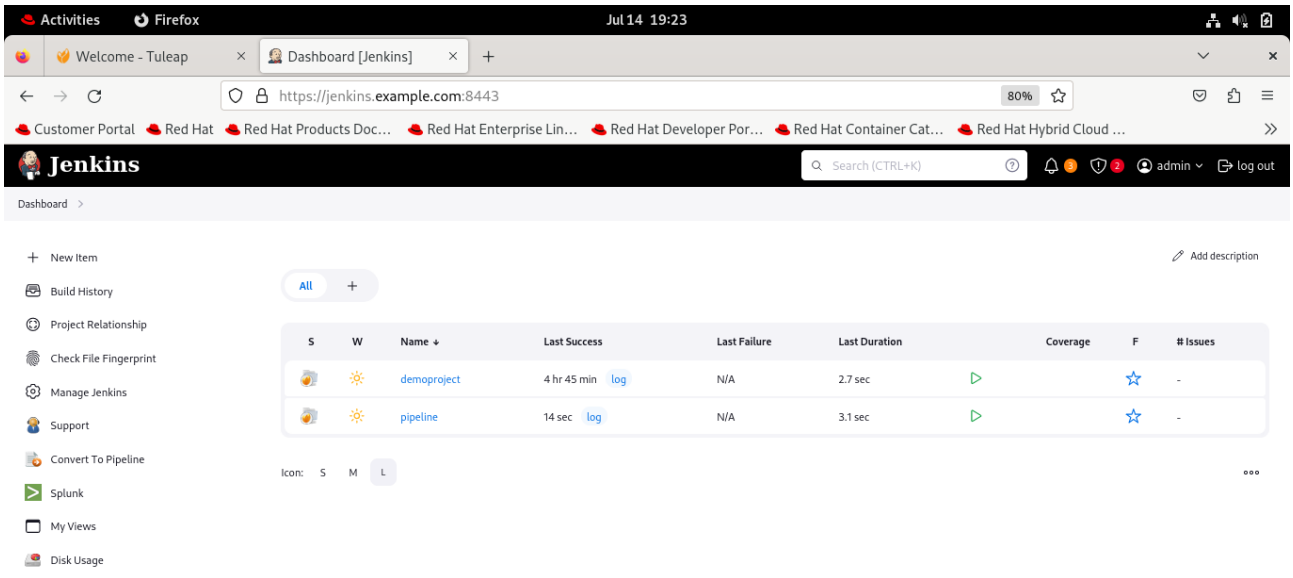
Tuleap:



Jenkins:

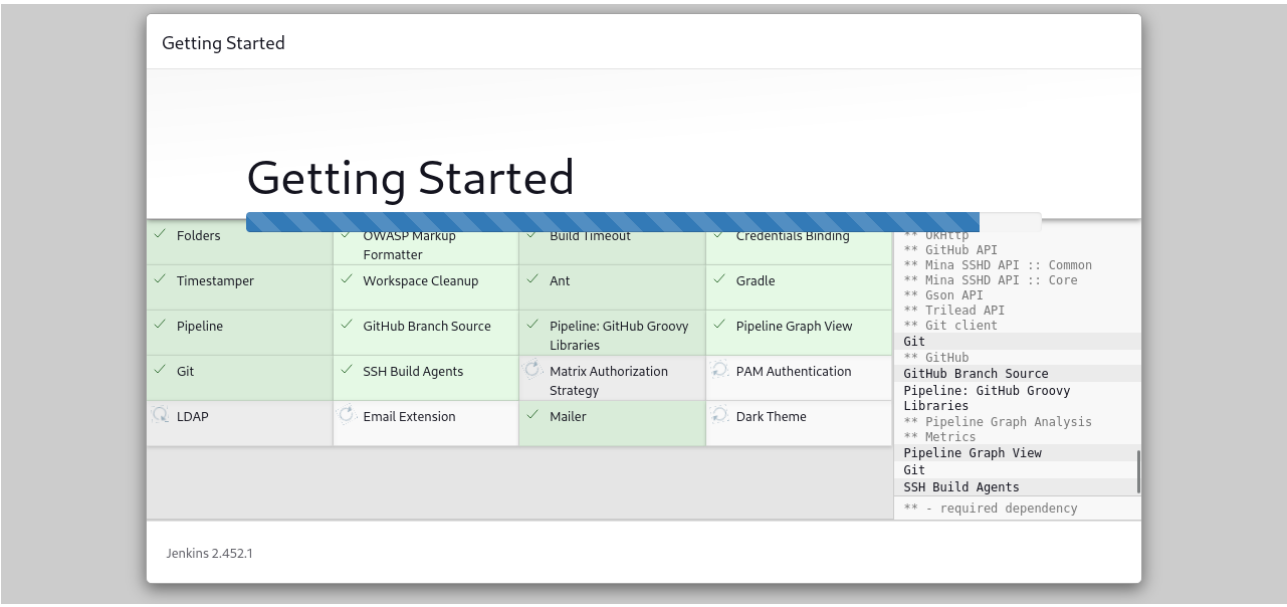


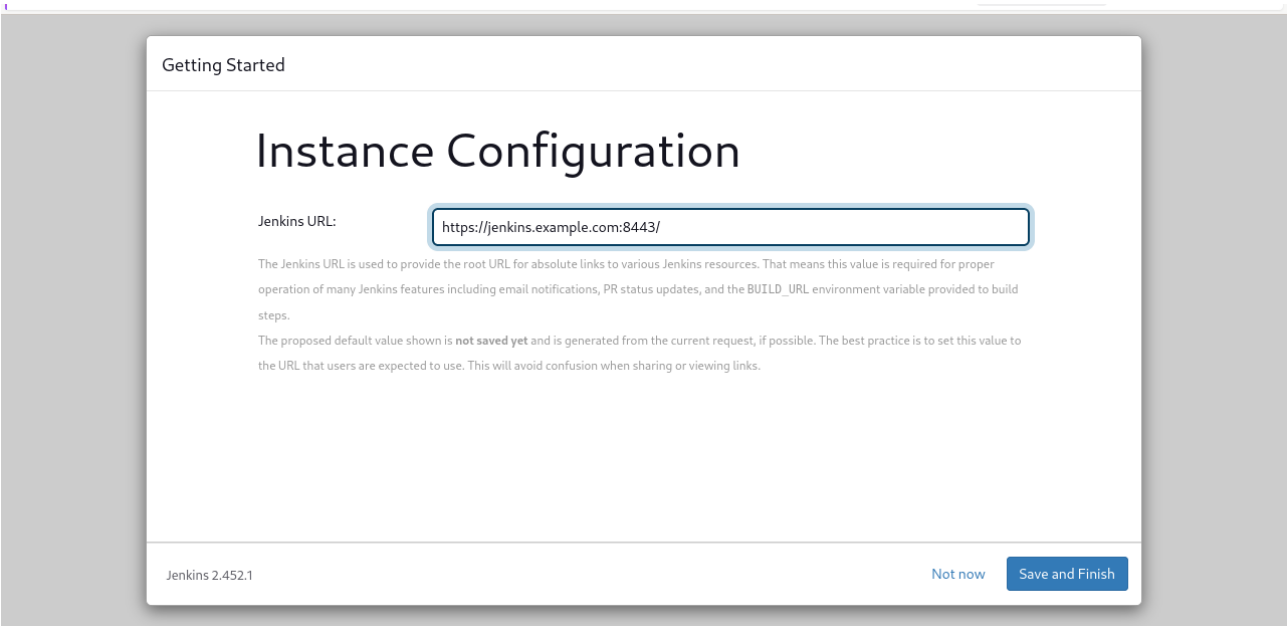
Jenkins Dashboard:



Jenkins initial setup:

$ cd /var/lib/jenkins/secrets (#jenkins home directory)
$ cat initialAdminPassword (Intial password)





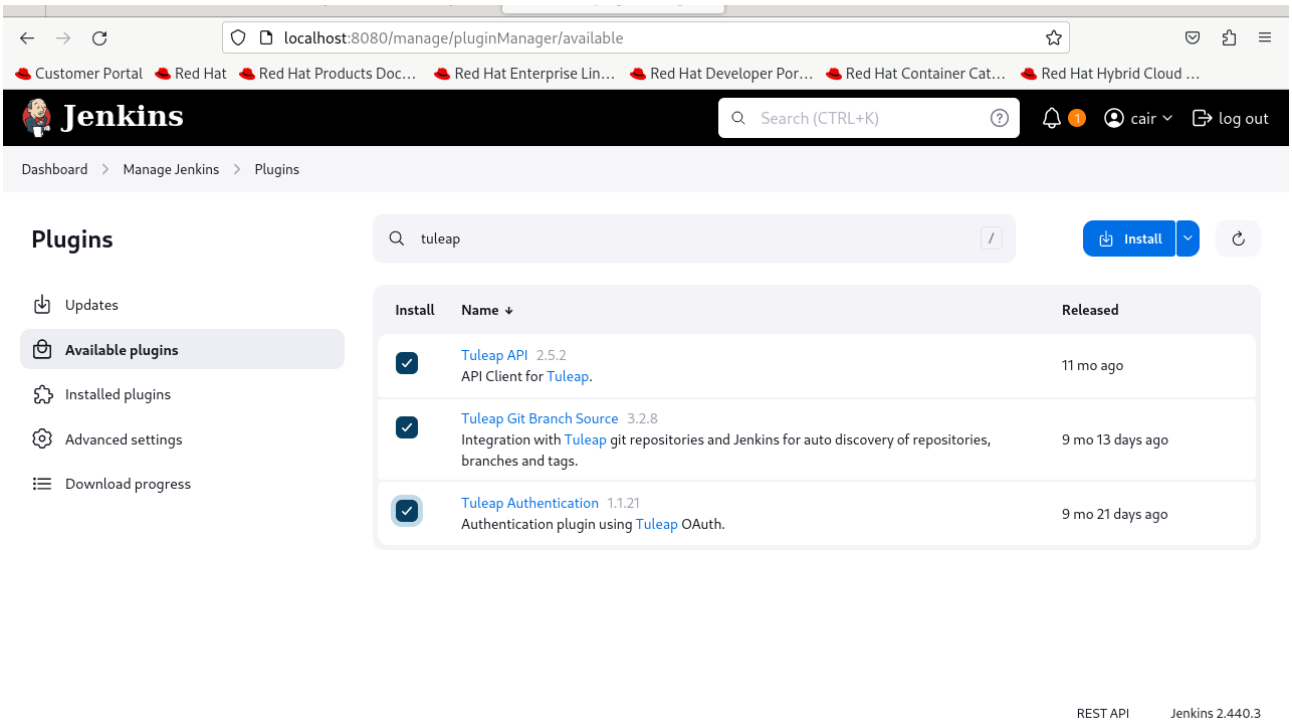
- **Create the Jenkins admin user credentials:**

Jenkins admin credentials:

username: admin
password: cair

Install Jenkins plugins:

Manage jenkins----manage---- plugins-Availble plugins (search)



Tuleap:
Tuleap API Version
Tuleap Git Branch Source Version
Tuleap Authentication Version

Git:
Git server Version
Pipeline: Deprecated Groovy Libraries Version
Jersey 2 API
Git Pipeline for Blue Ocean
GitHub Pipeline for Blue Ocean (optional)
Git Parameter Version
GitLab Version
Generic Webhook Trigger
Monitoring Version
Git Push
Last Changes
Job and Stage monitoring
Git Tag Message

Docker:

Docker #This plugin integrates Jenkins with Docker
Docker Commons
Docker Pipeline
Docker API Version
docker-build-step
CloudBees Docker Build and Publish
Docker Compose Build Step
Docker Slaves
CloudBees Docker Custom Build Environment Version
JFrog
CloudBees Docker Hub/Registry Notification

SSH:

ssh # This plugin executes shell commands remotely using SSH protocol.
SSH server #Adds SSH server functionality to Jenkins, exposing CLI commands through it.
Publish Over SSH
SSH Agent Version
SSH Pipeline Steps
SSH2 Easy
SCP publisher

CMake #Integrates the CMake tool suite to Jenkins. Supports to build cmake based
python # Adds the ability to execute python scripts as build steps.

cppcheck #his plug-in collects the Cppcheck analysis results of the project and visualizes the found errors.

Testing:

Performance
Test In Progress
JUnit Realtime Test Reporter

E-Mail:

Email Extension Template #This plugin allows administrators to create global templates for the Extended Email Publisher.
LDAP Email

Pipeline:

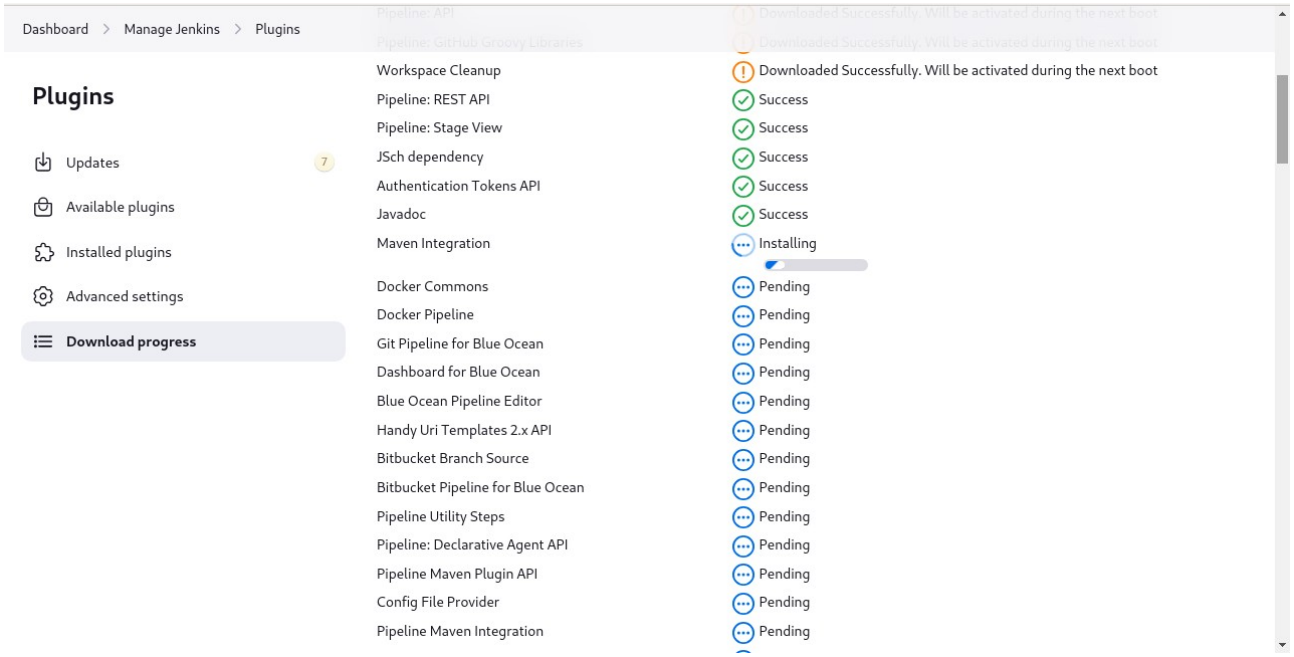
Pipeline: REST API Version #Provides a REST API to access pipeline and pipeline run data.
Pipeline: Stage View #Pipeline Stage View Plugin.
Pipeline Utility Steps
Artifactory Version
Pipeline: Multibranch with defaults
Plot
Webhook Step
Build Token Trigger
Gerrit Trigger Version
Multibranch Scan Webhook Trigger
BuildResultTrigger Version

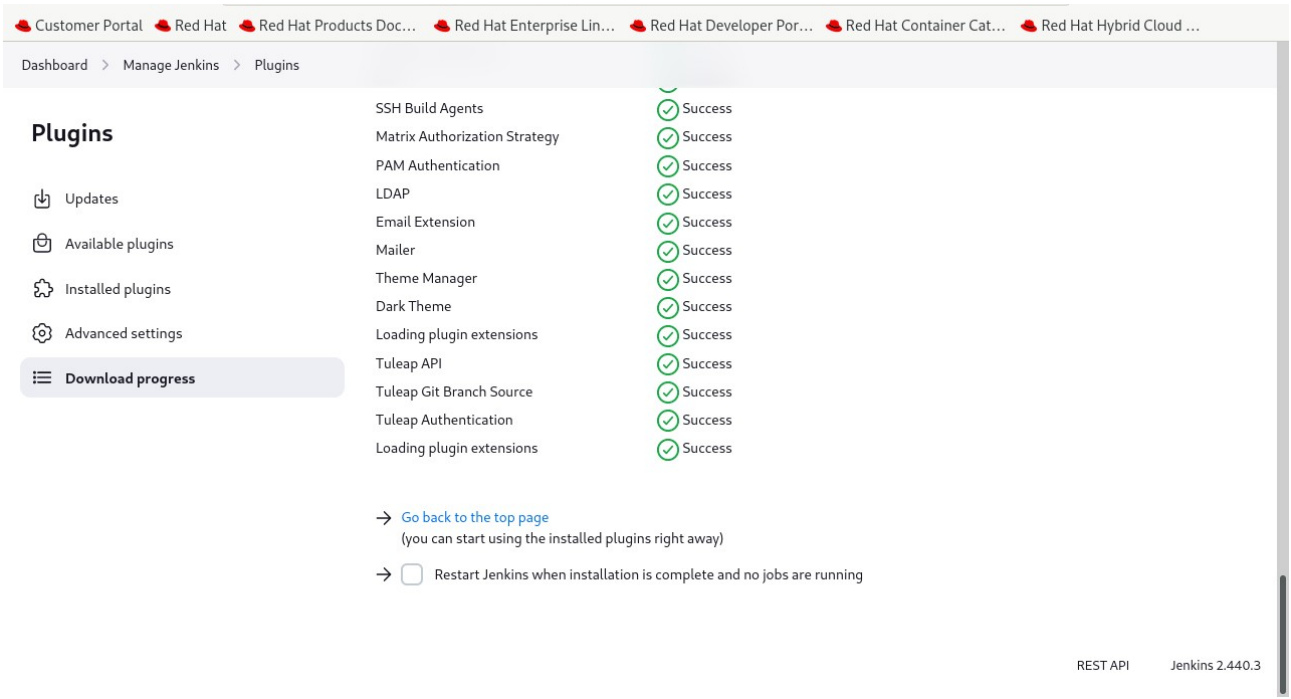
Additional plugins:

Role-based Authorization Strategy
Email Extension Template
SonarQube Scanner
prometheus
performance
junit
build metrics plugin
jacoco
cloud base disk usage

 # Restart the jenkins service
$sudo systemctl restart jenkins.service

Plugins Installing inJenkins:





Tuleap with Jenkins Integration:

This integration enables you to automate the building, testing, and deployment of your projects while keeping track of your development progress within Tuleap.

Or

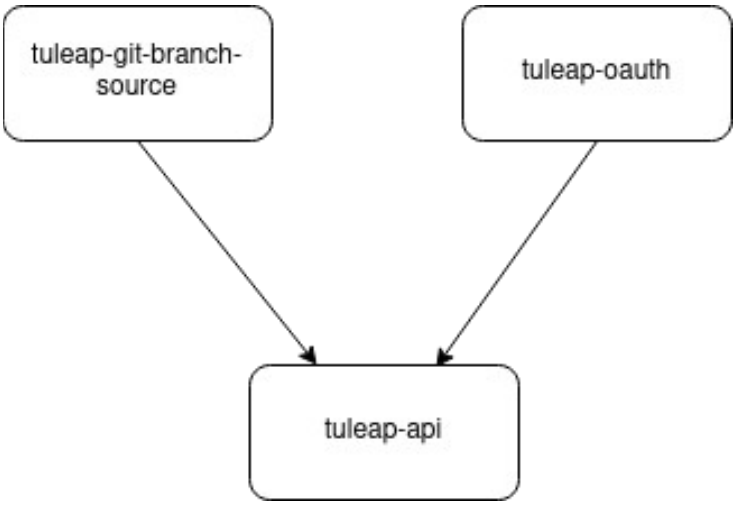
Tuleap-Jenkins Integration is a setup that connects Tuleap, an open-source project management and Agile planning tool, with Jenkins, an open-source automation server used for continuous integration and continuous delivery (CI/CD). This integration facilitates automatic build, test, and deployment processes while managing tasks, tracking progress, and collaborating on projects within Tuleap.

Benefits of Tuleap and Jenkins Integration:

- Unified Monitoring
- Automated Workflows
- Improved Collaboration
- Better Quality Control
- Cost Efficiency
- Increased Productivity

Integrating tuleap with jenkins before we need below plugins installed in Jenkins and tuleap:

Jenkins:

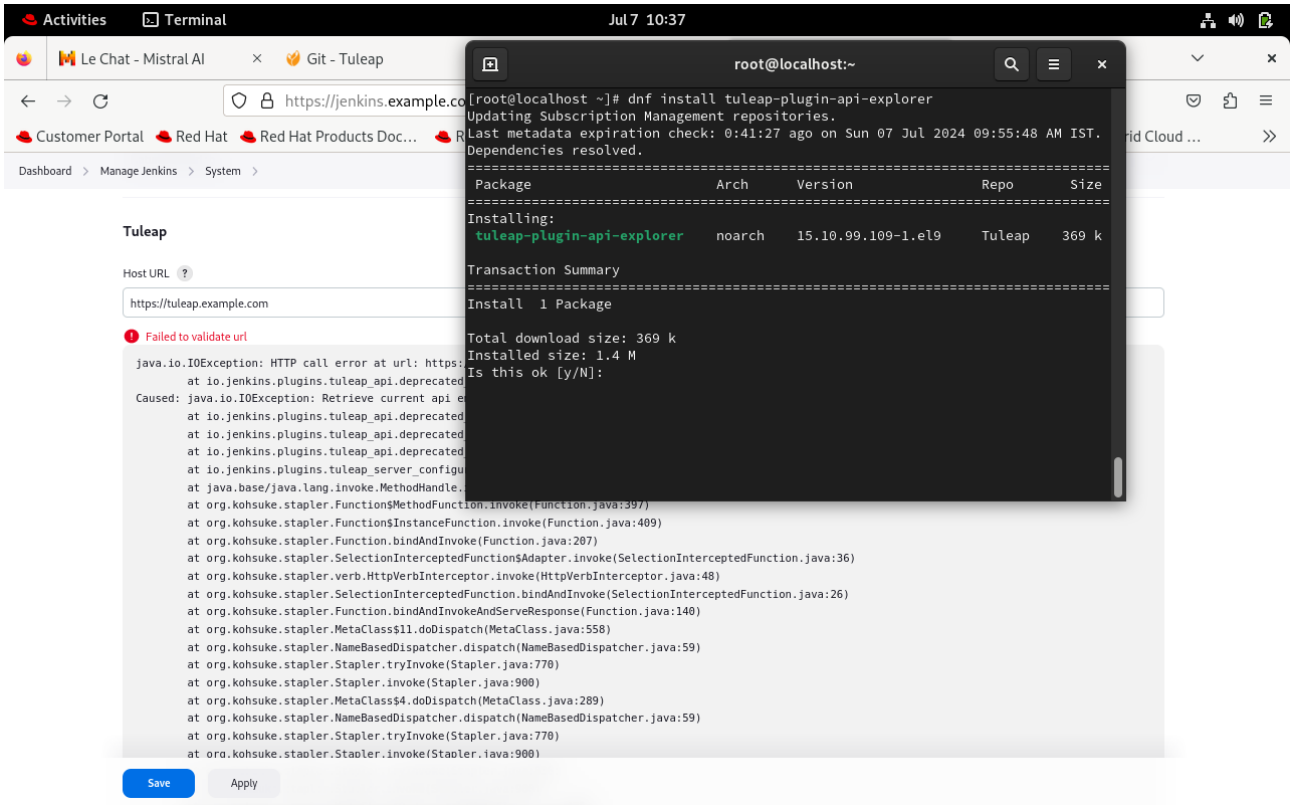


Tuleap:

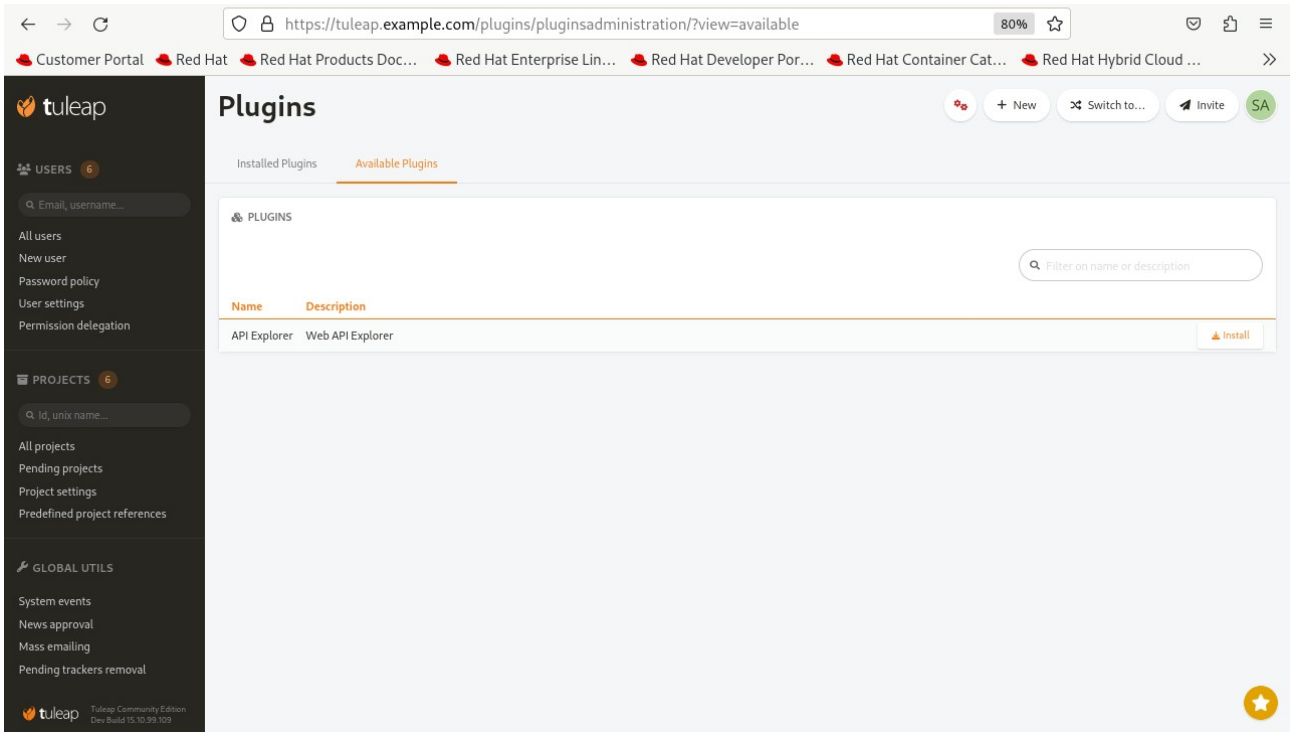
$ dnf install tuleap-plugin-api-explorer (In terminal)
# After install the tuleap api enable the plugin inside the tuleap

Restart the all servers:

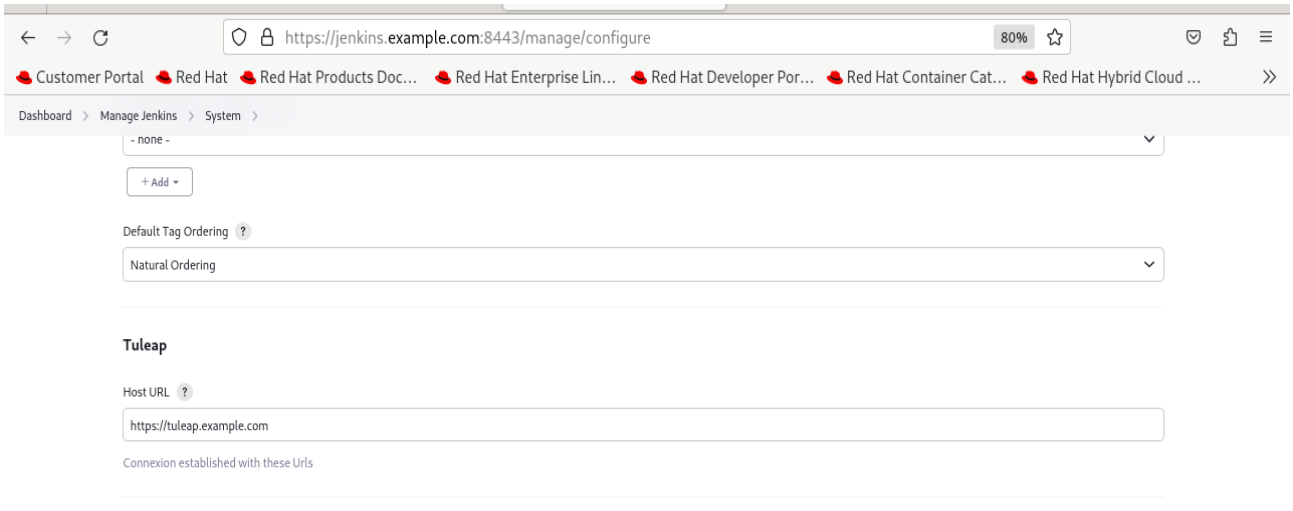
$ sudo systemctl restart jenkins.service
$ sudo systemctl restart tuleap
$ sudo systemctl restart nginx



Enable the plugin:



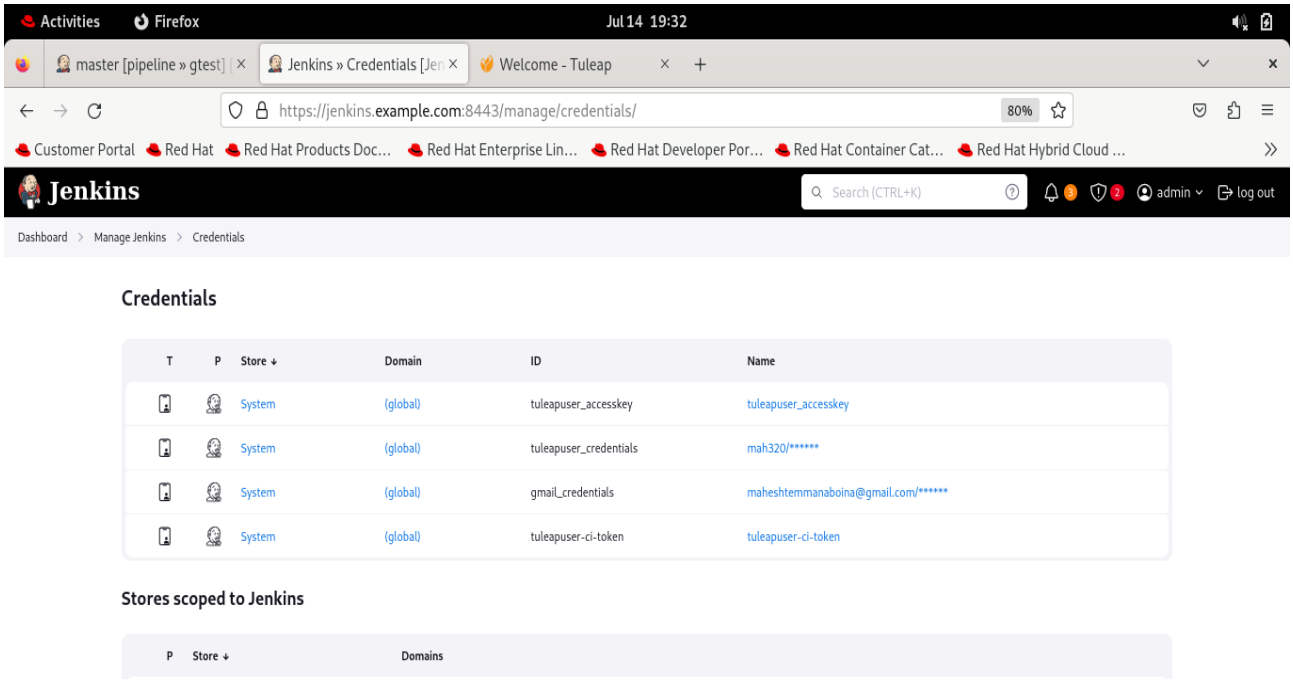
Verify the tuleap connection in Jenkins system configuration inside the Tuleap section:



Add Credentials for Tuleap in Jenkins:

Go to----manage Jenkins—select credentials option---inside select global credentials and add

- Tuleap Access key
- Tuleap user credentials
- E-mail credentials
- CI-Token



Install Docker and Docker Compose on Red Hat 9.4

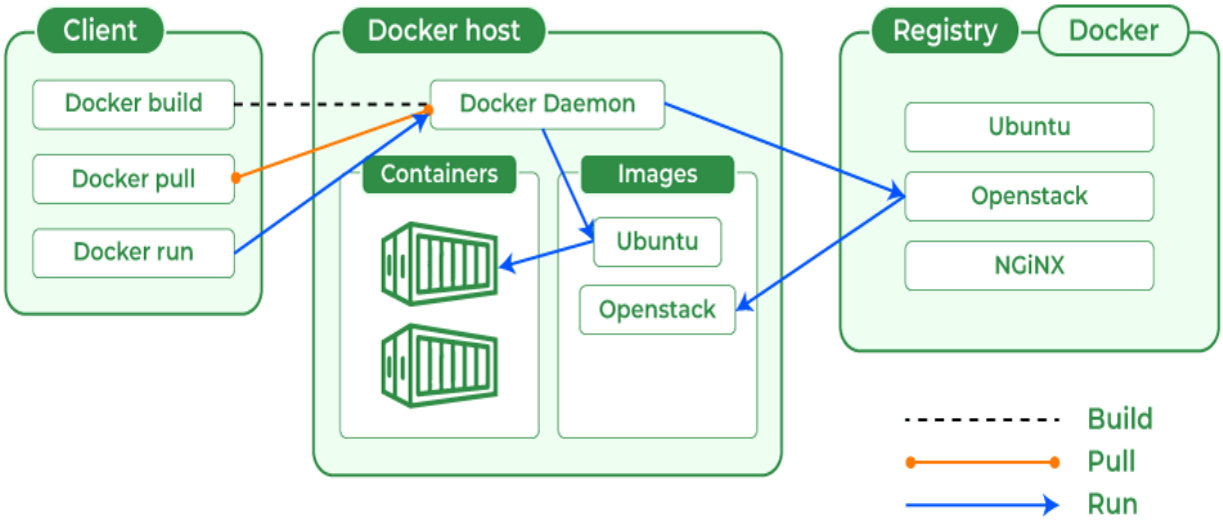
Docker is an open-source platform that automates the deployment, scaling, and management of applications using containerization. Containers are lightweight, portable, and can run on any system that supports Docker, ensuring consistency across different environments.

Or

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Docker Compose is a tool for defining and running multi-container Docker applications. It uses a YAML file to configure the application's services, networks, and volumes, allowing users to manage multiple containers as a single application.

Architecture of Docker:



Docker Components:

Docker Client:

- The interface through which users interact with Docker.
- Commands like `docker build`, `docker run` are issued from the client.

Docker Daemon (docker d):

- The background service running on the host that manages building, running, and distributing Docker containers.

Docker Images:

- Read-only templates used to create containers.
- Can be stored in a Docker registry like Docker Hub.

Docker Containers:

- Lightweight, standalone, executable packages that include everything needed to run a piece of software, including the code, runtime, libraries, and dependencies.

Docker Registries:

- Storage and distribution system for Docker images.
- Docker Hub is a public registry; private registries can also be set up.

Docker Engine:

- The core part of Docker, consisting of the Docker Daemon, API, and CLI.
- Manages the life cycle of containers.

Docker Commands:-

$ sudo yum update -y
$ sudo yum install -y yum-utils
$ sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
$ sudo yum install -y docker-ce docker-ce-cli containerd.io
$ sudo systemctl start docker
$ sudo systemctl enable docker
$ docker --version

Docker Compose:

#Download the binary file from the project's GitHub page
$ curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o docker-compose

$ sudo mv docker-compose /usr/local/bin && sudo chmod +x /usr/local/bin/docker-compose
$ sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

$ docker-compose --version

Basic Commands of Docker:

- docker info
- docker --version
- docker pull <image>
- docker run <image>
- docker ps
- docker ps -a
- docker stop <container>
- docker start <container>
- docker restart <container>
- docker rm <container>
- docker rmi <image>
- docker build -t <image> .
- docker images
- docker exec -it <container> /bin/bash
- docker logs <container>
- docker inspect <container>
- docker network ls
- docker network create <network>
- docker network inspect <network>
- docker volume ls
- docker volume create <volume>
- docker volume inspect <volume>
- docker-compose up
- docker-compose down
- docker-compose build
- docker-compose ps
- docker-compose logs
- docker-compose start
- docker-compose stop
- docker-compose restart
- docker-compose pause
- docker-compose unpause
- docker-compose down -v
- docker-compose logs -f
- docker-compose exec <service> <command>

- docker-compose scale <service>=<number>
- docker-compose build --no-cache
- docker-compose pull
- docker-compose push
- docker-compose config
- docker-compose events
- docker container prune
- docker rm -f <container_id>
- docker rm -f $(docker ps -aq)

Install ROS2- Red Hat humble image on local host Machine

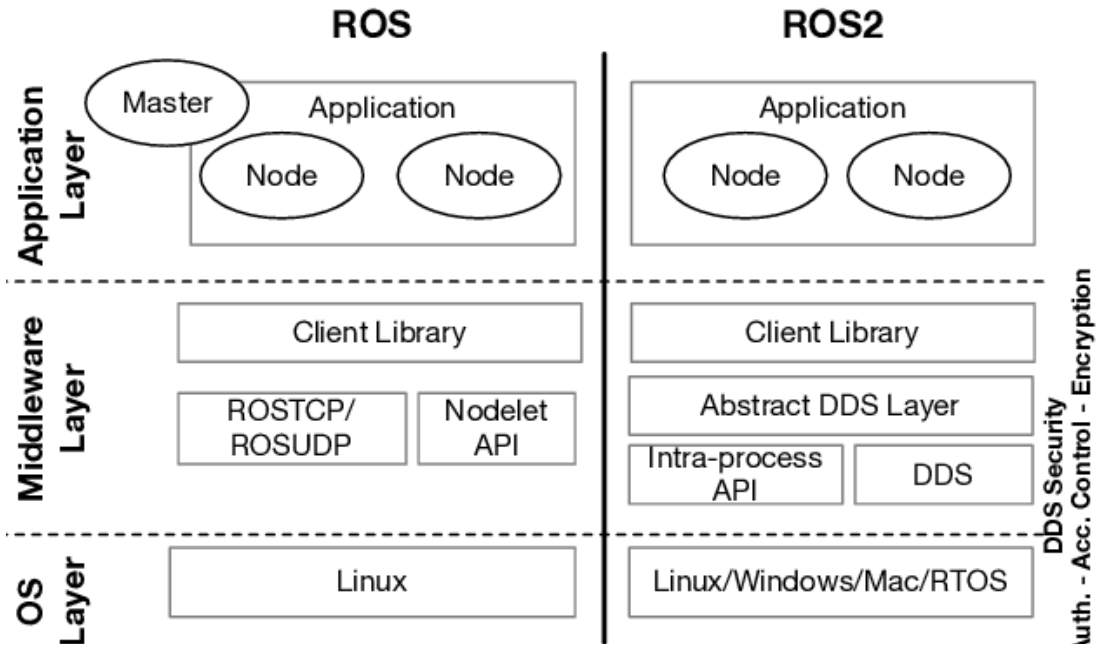
Website:

https://docs.ros.org/en/humble/How-To-Guides/Run-2-nodes-in-single-or-separate-docker-containers.html (Redhat Ros2 image for humble)

What is ROS2:

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.

ROS2 Architecture:



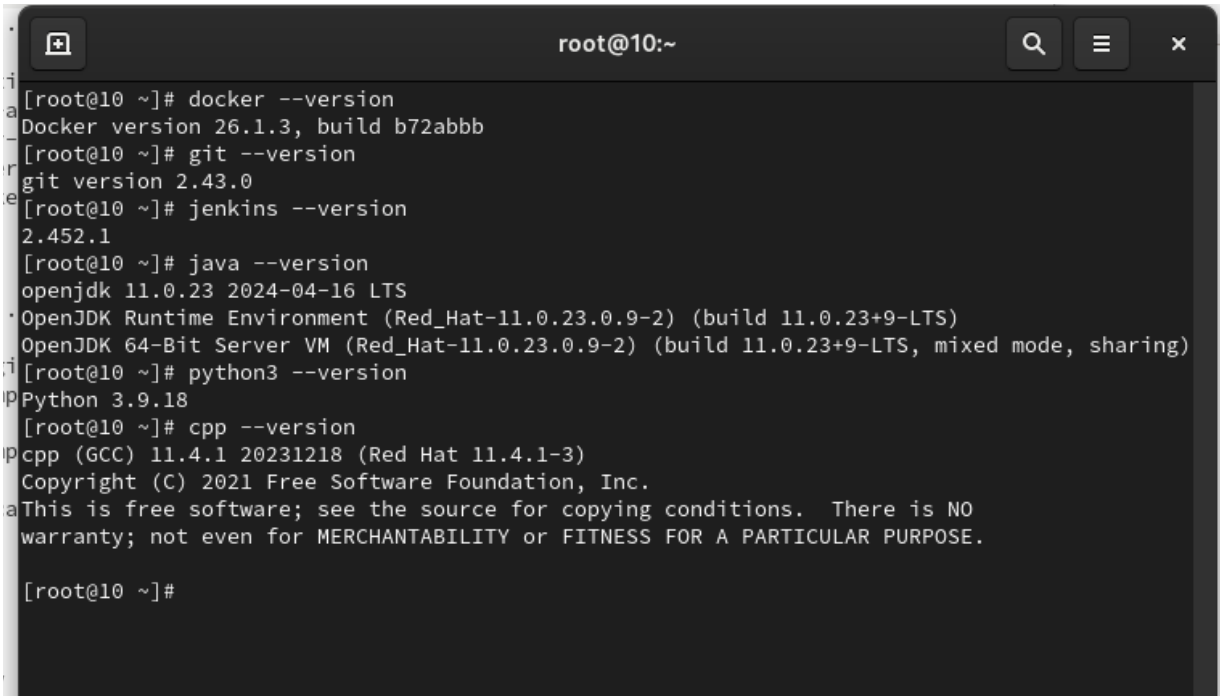
Install ros2 humble docker image:

$docker pull osrf/ros:humble-desktop
$ docker images
$ docker run -it <id>
$ docker ps
$ docker ps -a

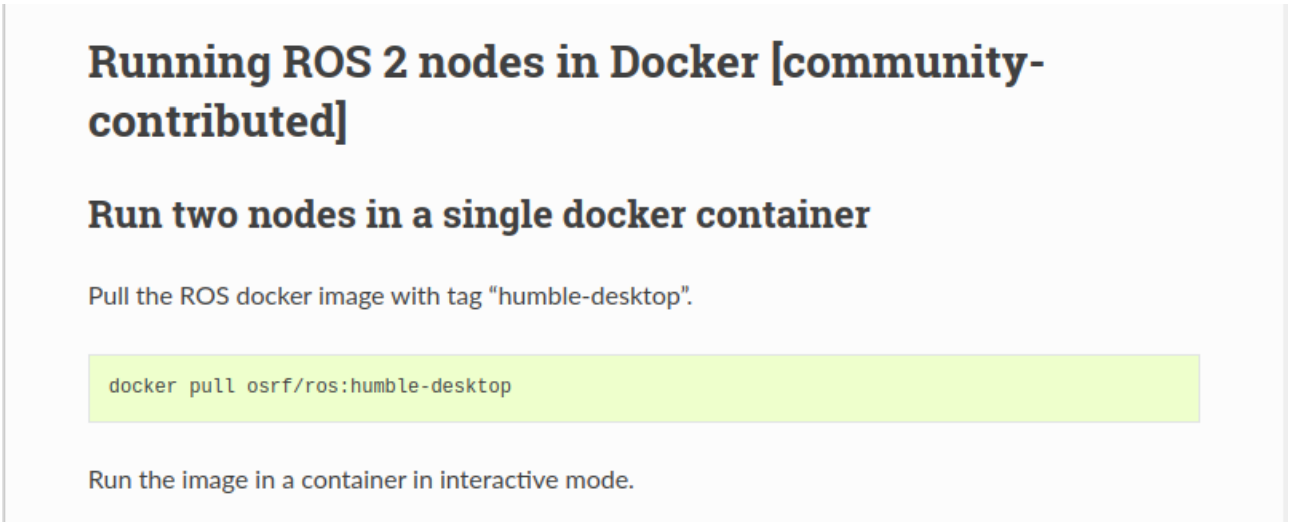
#user permissions Access for Jenkins, docker:

$ sudo usermod -aG jenkins cair
$ sudo chown -R cair:cair /var/lib/jenkins
$ id jenkins
$ sudo systemctl restart jenkins
$ dnf update -y

Installed Git, Docker, Jenkins, Tuleap, Python, C++, C, Java, Docker Compose



Ros2 Humble Image:



Running ROS 2 nodes in Docker [community-contributed]

Run two nodes in a single docker container

Pull the ROS docker image with tag "humble-desktop".

```
docker pull osrf/ros:humble-desktop
```

Run the image in a container in interactive mode.

Install Grafana, Node-Exporter and Prometheus on Red Hat Docker (Monitoring Tools)

Prometheus:

Prometheus is an open-source systems monitoring and alerting toolkit. It is designed to collect metrics from various sources, store them efficiently, and provide powerful querying capabilities. However, Prometheus itself does not gather metrics directly from the system; it relies on exporters for this purpose.(node exporter)

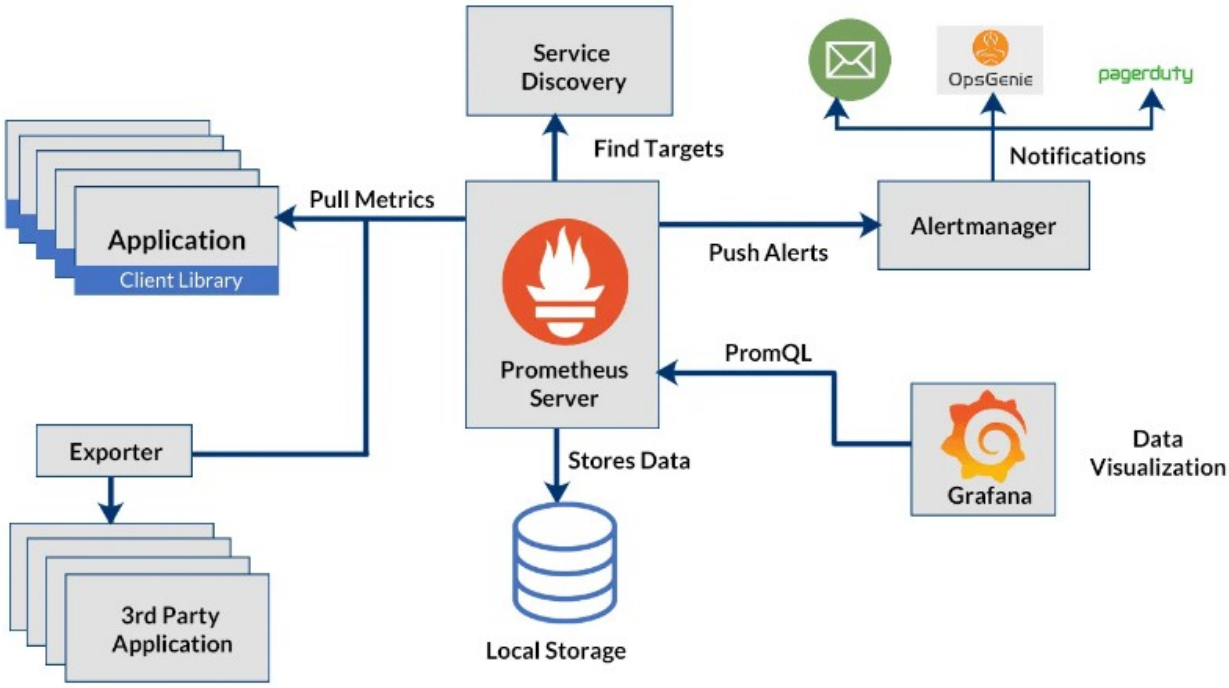
Grafana:

Grafana is an open-source platform for monitoring and observability. It provides a powerful and flexible dashboarding capability to visualize metrics collected by Prometheus. Grafana integrates with Prometheus to display metrics in various formats (graphs, tables, etc.) and allows users to set up alerts based on these metrics.

Node Exporter:

Node Exporter is a Prometheus exporter specifically designed to expose system-level metrics. These metrics include detailed information about the machine's hardware and operating system. Node Exporter is crucial for a comprehensive monitoring setup because it provides the raw data about the system's performance and health.

Architecture of Grafana and Prometheus:



Advantages of Node Exporter, Prometheus, and Grafana:

1. Provides granular details on CPU, memory, disk I/O, file system, and network statistics. Essential for diagnosing issues and optimizing system performance. (node-exporter)

2. Seamless Integration (Prometheus and Node Exporter):

Node Exporter exposes system metrics over HTTP, which Prometheus scrapes at regular intervals. Ensures consistent and reliable data collection from various sources.

3. Comprehensive Monitoring Solution:

Combines system-level metrics (Node Exporter) with application-level metrics (Prometheus). Offers a complete view of infrastructure health and performance.

4. Powerful Data Storage and Querying (**Prometheus**):

.Efficiently stores time-series data.
.Provides powerful querying capabilities to analyze metrics.

5. Flexible Visualization (**Grafana**):

.Visualize metrics collected by Prometheus in Grafana.
.Create customizable dashboards with various formats (graphs, tables, etc.).
.Import and customize pre-built dashboards for Node Exporter metrics.

Install prom, grafana, node-expo on docker redhat localhost machine:

$docker pull grafana/grafana
$ docker pull prom/node-exporter
$ docker pull prom/prometheus

Create a directory in terminal:

$ mkdir <>
$ cd <>
$ cd /home/cair/monitor

Create two files inside the above directory:

docker-compose.yml prometheus.yml

$ nano docker-compose.yml

```
version: '3.7'
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    ports:
      - "9090:9090"
    networks:
      - monitoring
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml

  node-exporter:
    image: prom/node-exporter
    container_name: node-exporter
    networks:
      - monitoring
    ports:
      - "9100:9100"

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3100:3000"
    networks:
      - monitoring

networks:
  monitoring:
    driver: bridge
```

save and exit!

$ nano prometheus.yml

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node-exporter:9100']
```

save and exit!

```
$ docker-compose up -d #Run the containers
$ docker ps
$ docker ps -a
```

Access in browser:

http://localhost:9090/
http://localhost:9100
http://172.17.0.1:3100/

Grafana:

user: admin
passwd: admin

user: admin
passwd:cair

Grafana basic process:

grafana:

grafana--goto--data sources--add source---add prome data source

url section give prometheus url--http://172.17.0.1:9090

Dashboard-new-import-<14513>--add prome—import--

Errors:

$ docker-compose restart prometheus
$ docker-compose ps

Resolve the issue:prome, grafana, node-exporter:

Dont compose every time do restart the container and will come status automatically

$ docker restart <>
$ docker ps

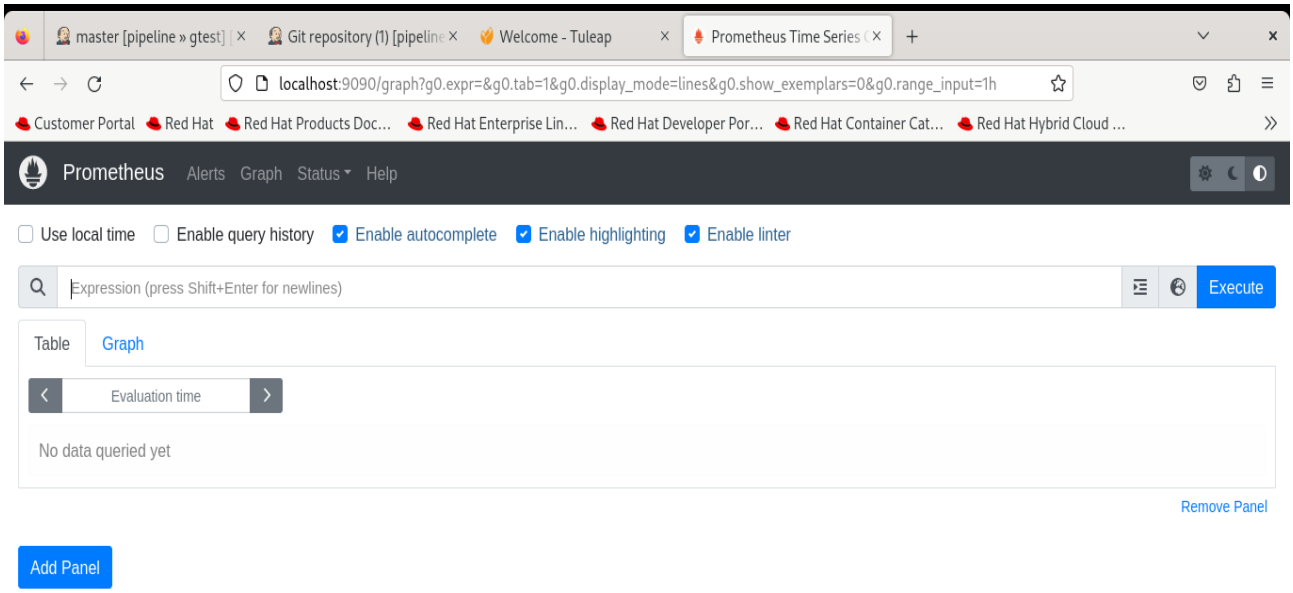
login to grafana
username: admin
passwd: cair

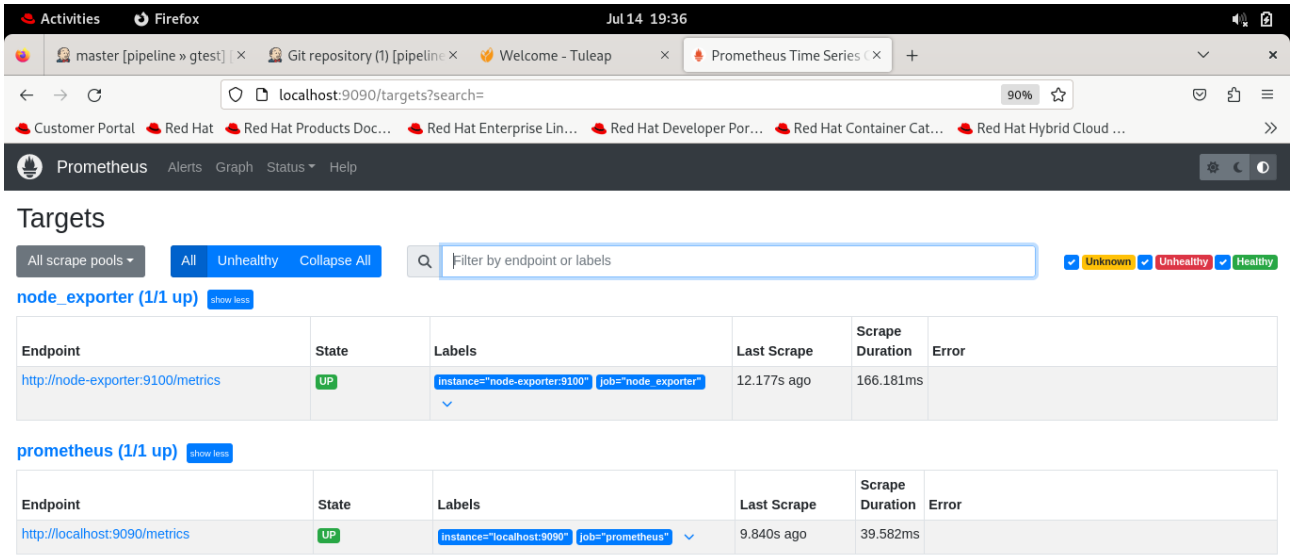
Check the server status:

Commands:

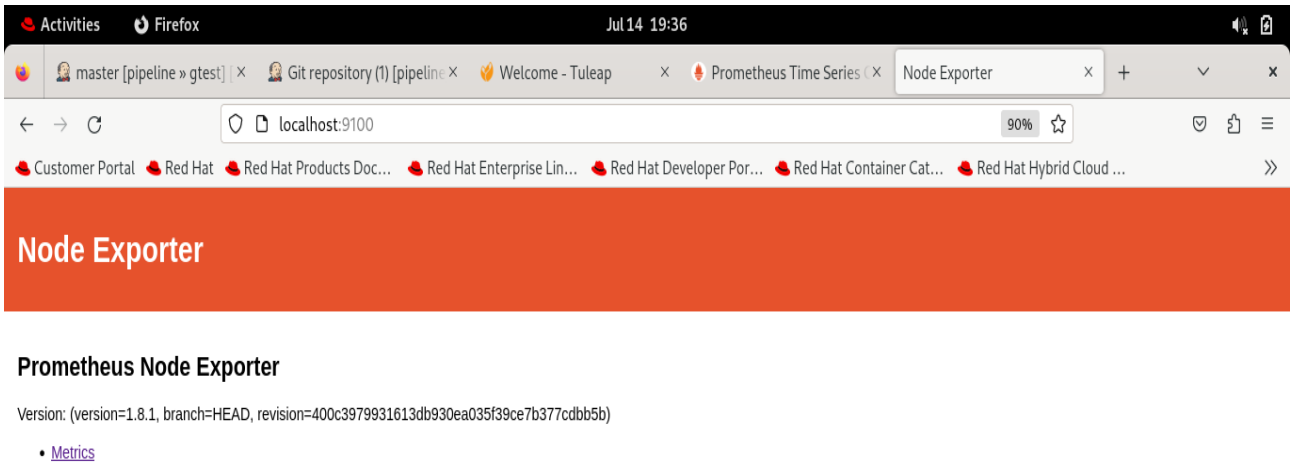
$ top (cpu)
$ uptime
$ free -h (ram and swap)
$ df -h /
$ nproc
$ uptime -p (system uptime)
$ grep MemTotal /proc/meminfo (ram total)
$ grep SwapTotal /proc/meminfo (swap)

System Server Monitor:

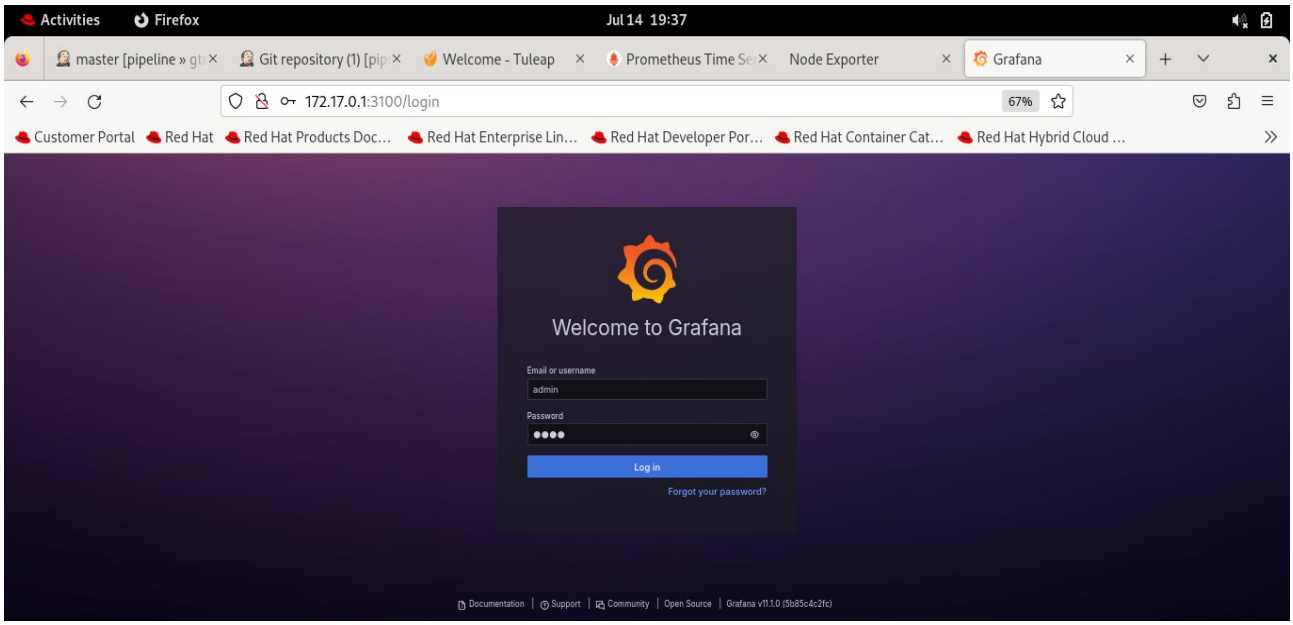




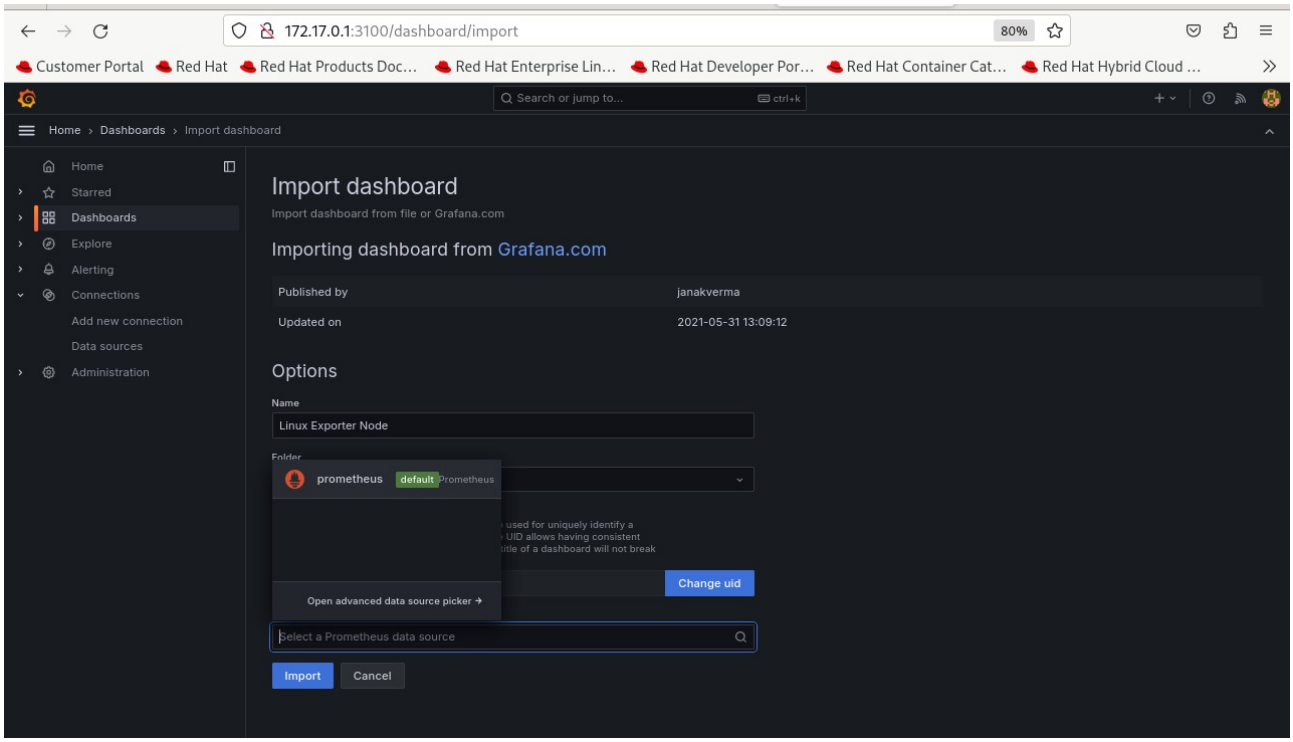
Node-Exporter:



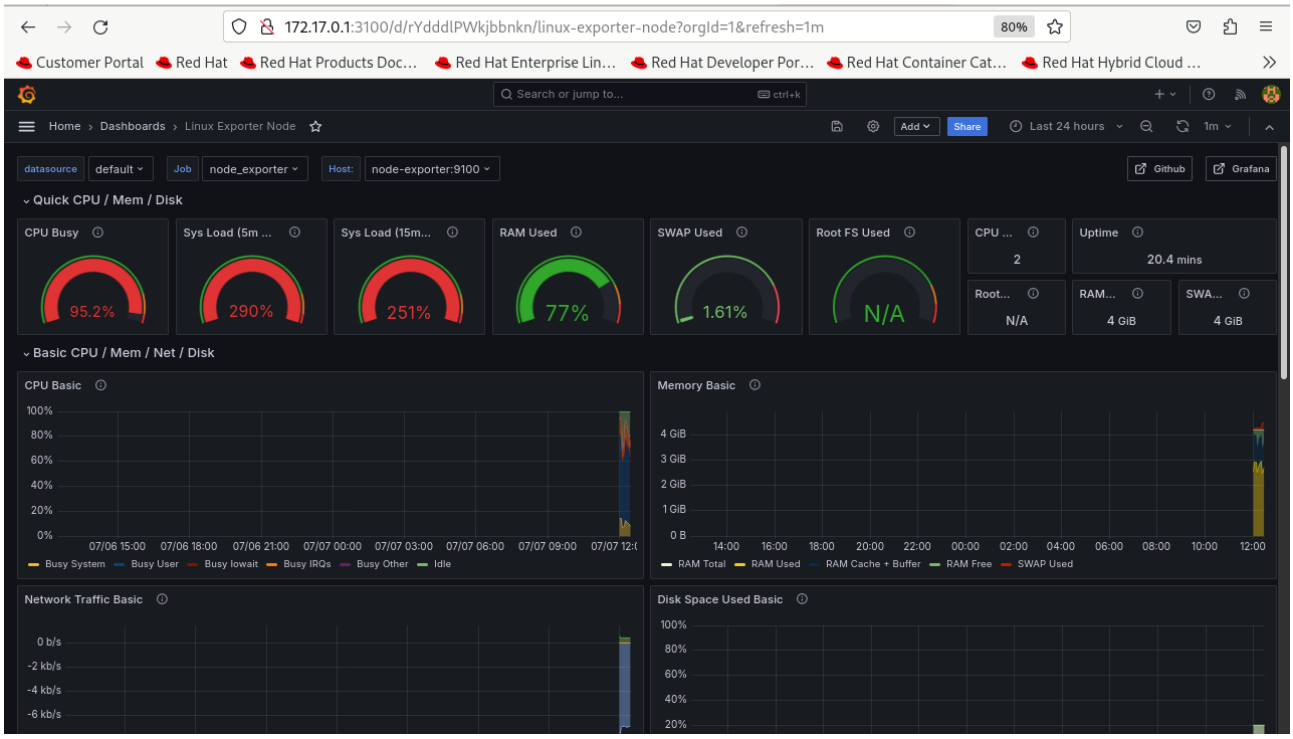
Grafana:



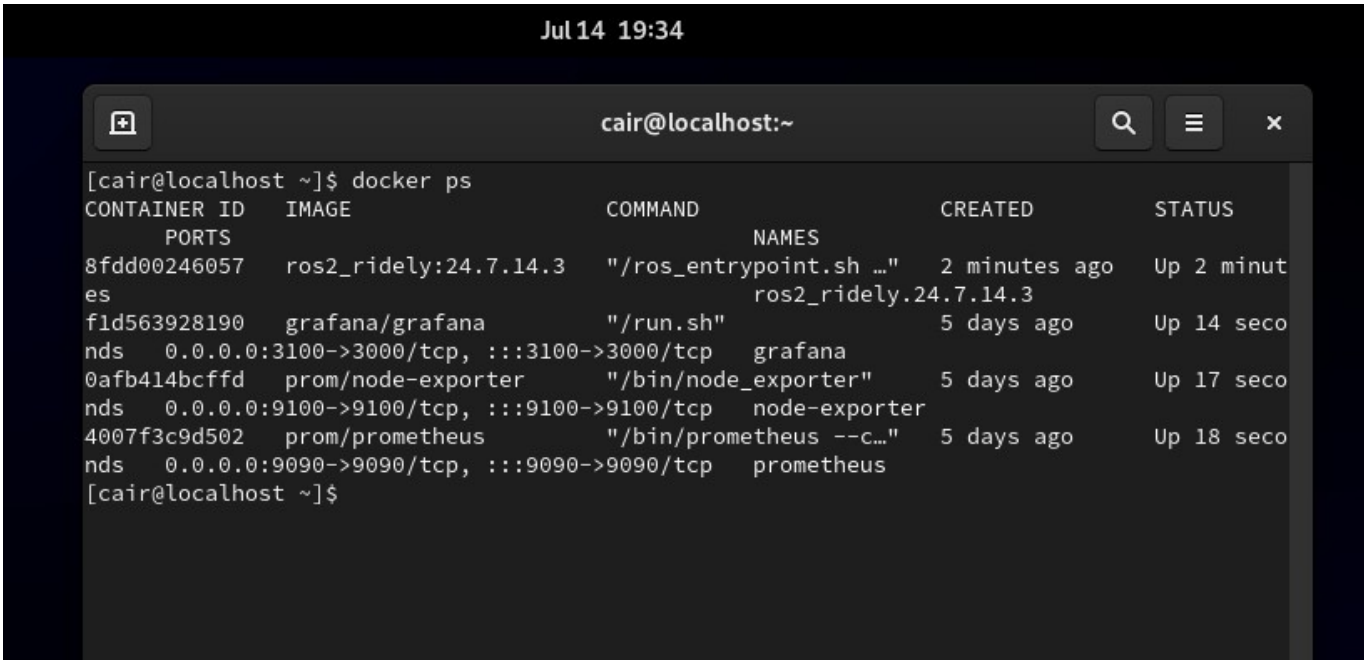
Import the Prometheus dashboard:



Server Monitor Status:

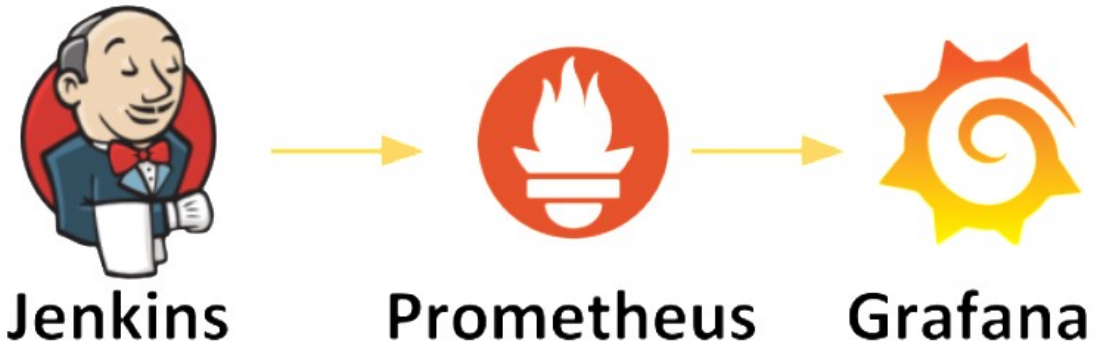


Verify the docker containers status:



How to monitor Jenkins with Grafana and Prometheus

This setup involves integrating Jenkins, an automation server, with Grafana, a data visualization tool, and Prometheus, a monitoring and alerting toolkit. Prometheus collects metrics from Jenkins, such as build times, job statuses, and resource usage, while Grafana provides visualizations and dashboards to analyze these metrics in real-time. This integration allows teams to monitor Jenkins' performance, detect bottlenecks, and optimize workflows for better efficiency and reliability.



Advantages of Monitoring Jenkins with Grafana and Prometheus:

1. Centralized monitoring
2. Real-time visibility
3. Historical analysis
4. Customization dashboards
5. Alerting and notifications
6. Integration flexibility
7. Open source community

Configure the Jenkins in Prometheus location:

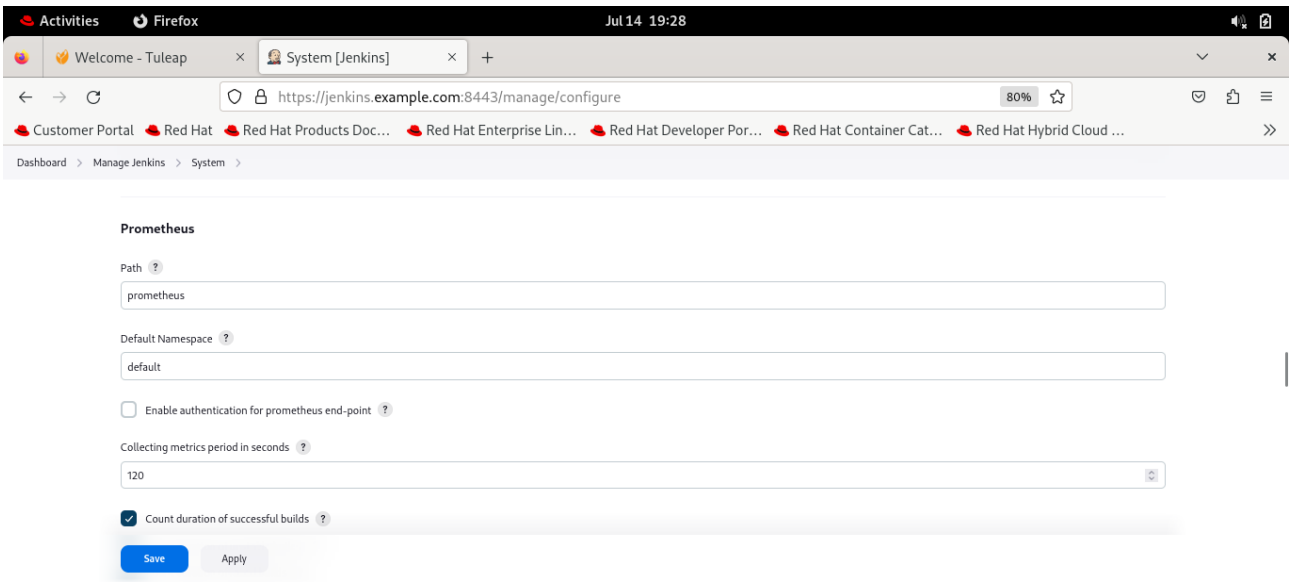
Install plugins in jenkins:

https://jenkins.example.com:8443/prometheus/ (metrics data)

Ensure the Prometheus Metrics plugin is correctly configured in Jenkins:

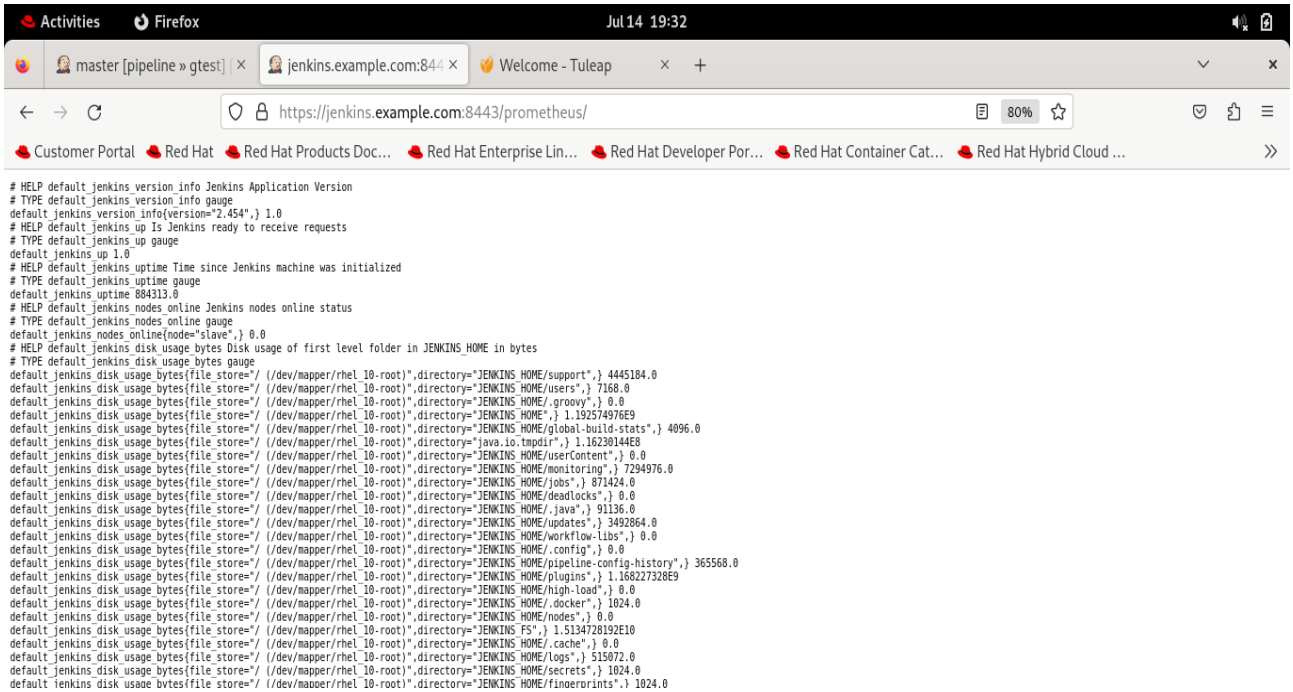
Go to Jenkins -> Manage Jenkins -> Configure System -> Prometheus.

Check that the "Enable Prometheus metrics" option is enabled and the endpoint is correctly set (default /prometheus).



Jenkins Metrics:

https://jenkins.example.com:8443/prometheus/ (metrics data)



Now add Jenkins in Prometheus location

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node-exporter:9100']

  - job_name: 'jenkins'
    metrics_path: /prometheus/
    static_configs:
      - targets: ['localhost:8443']  # Assuming localhost can access Jenkins directly
```

save and exit!

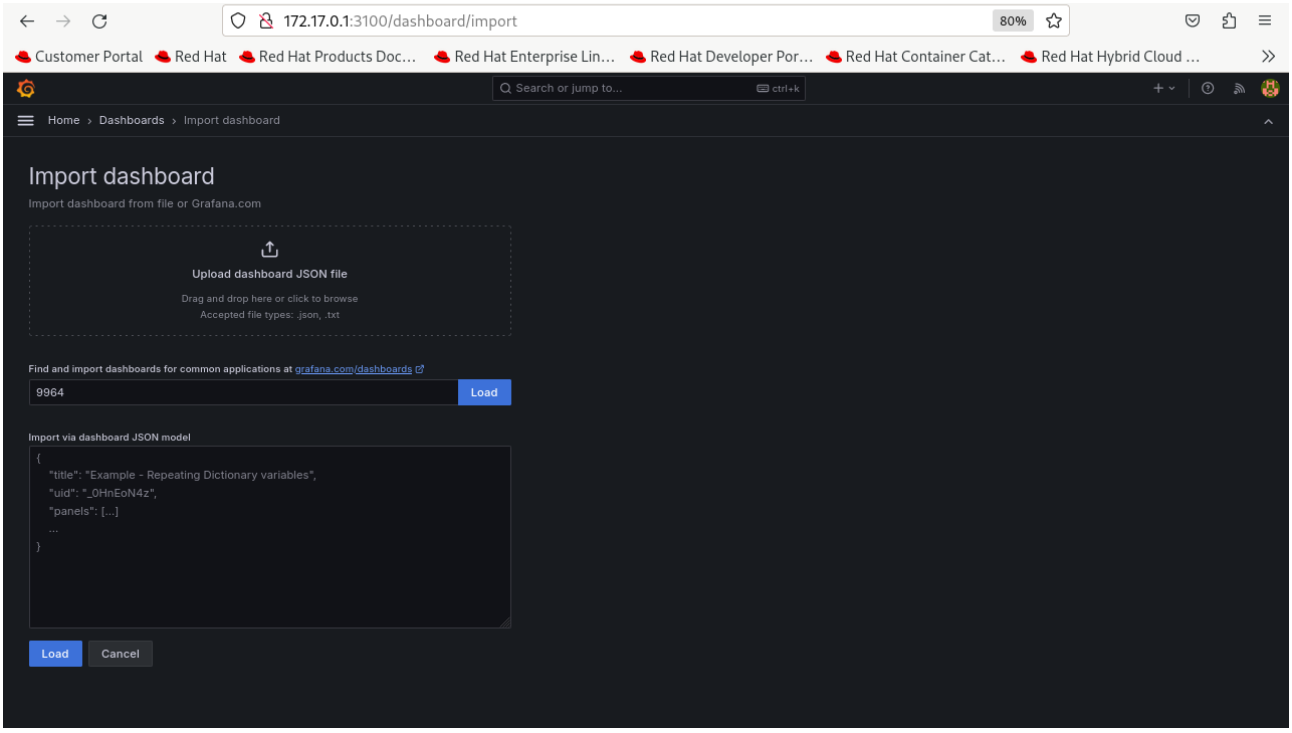
```
$ docker restart <>
$ docker start <>
$ docker ps
```

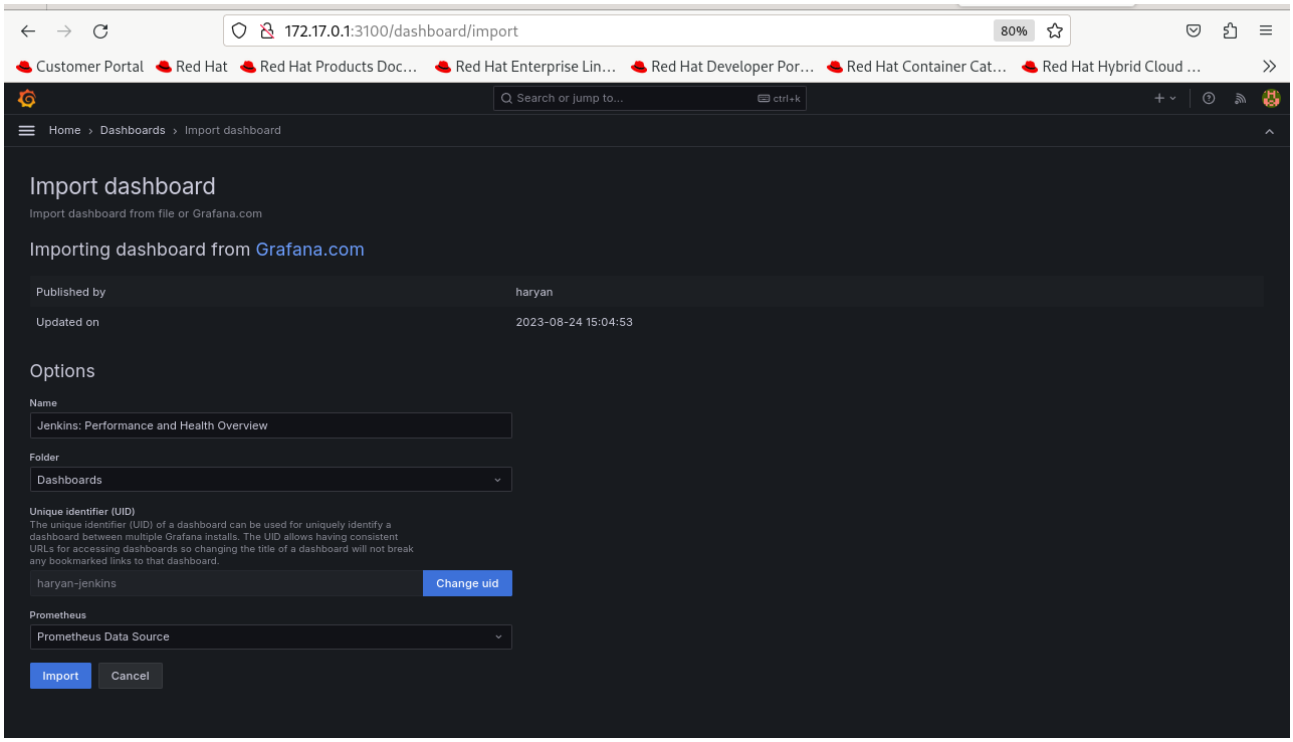
Web Access:

http://localhost:9090/
http://localhost:9100
http://172.17.0.1:3100/

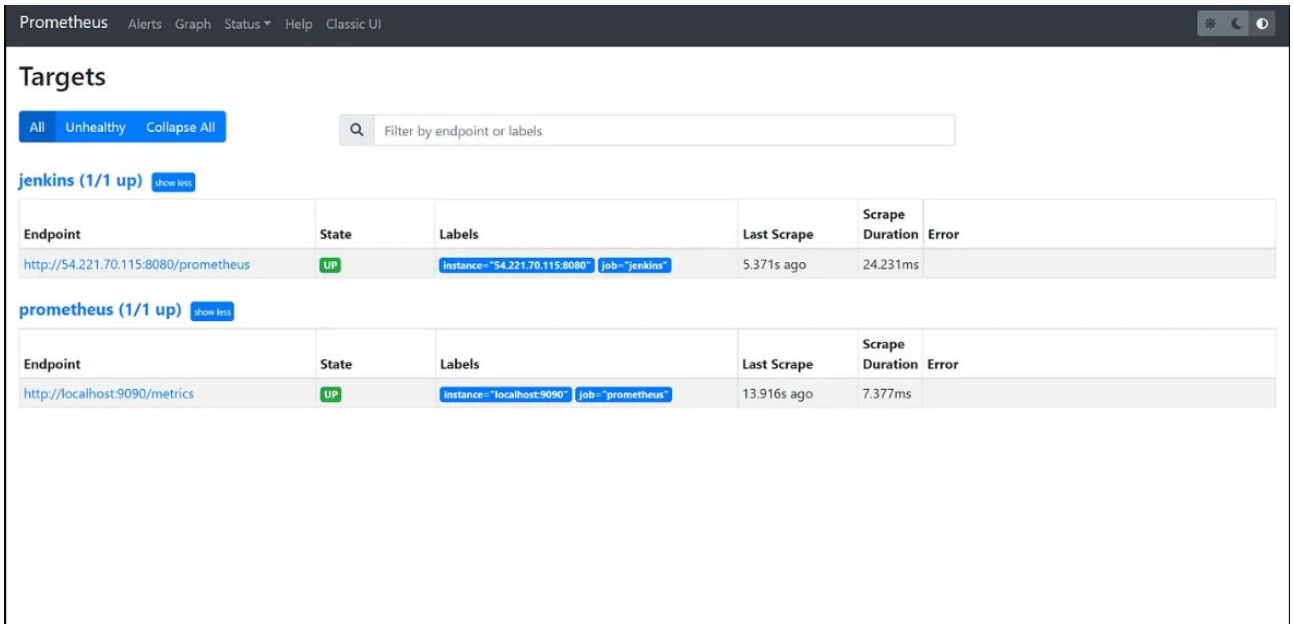
Goto--status-configuration
Goto--status--targets

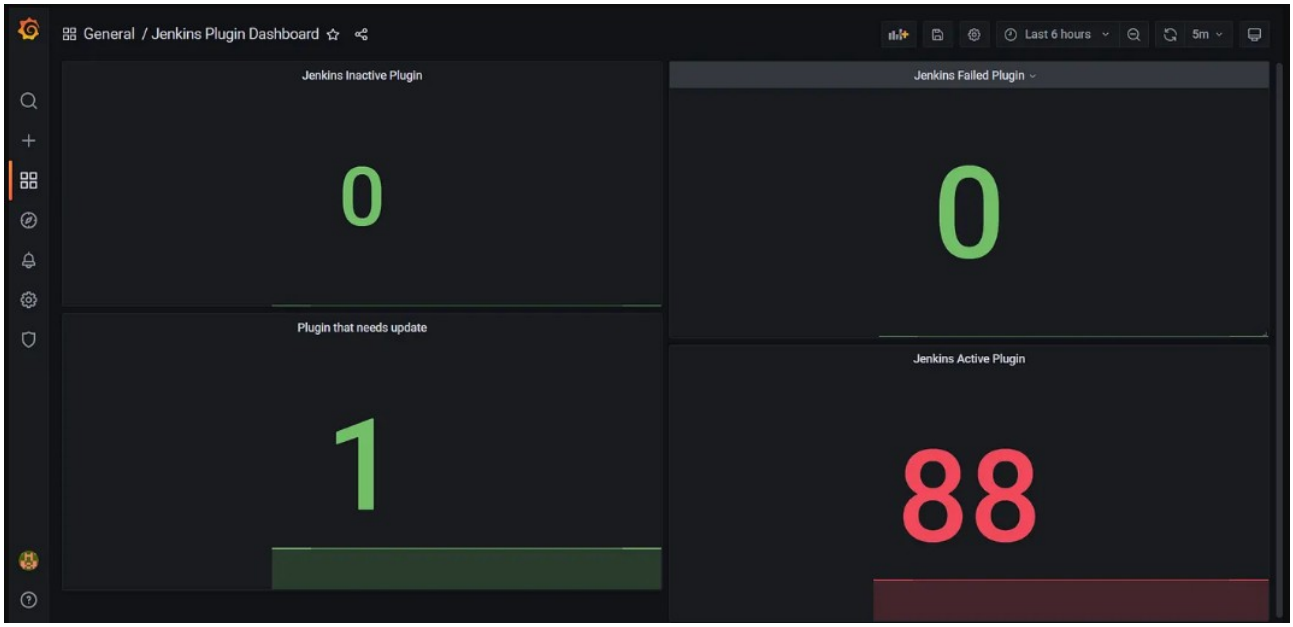
Dashboard-new-import-<9964>--add prome--import

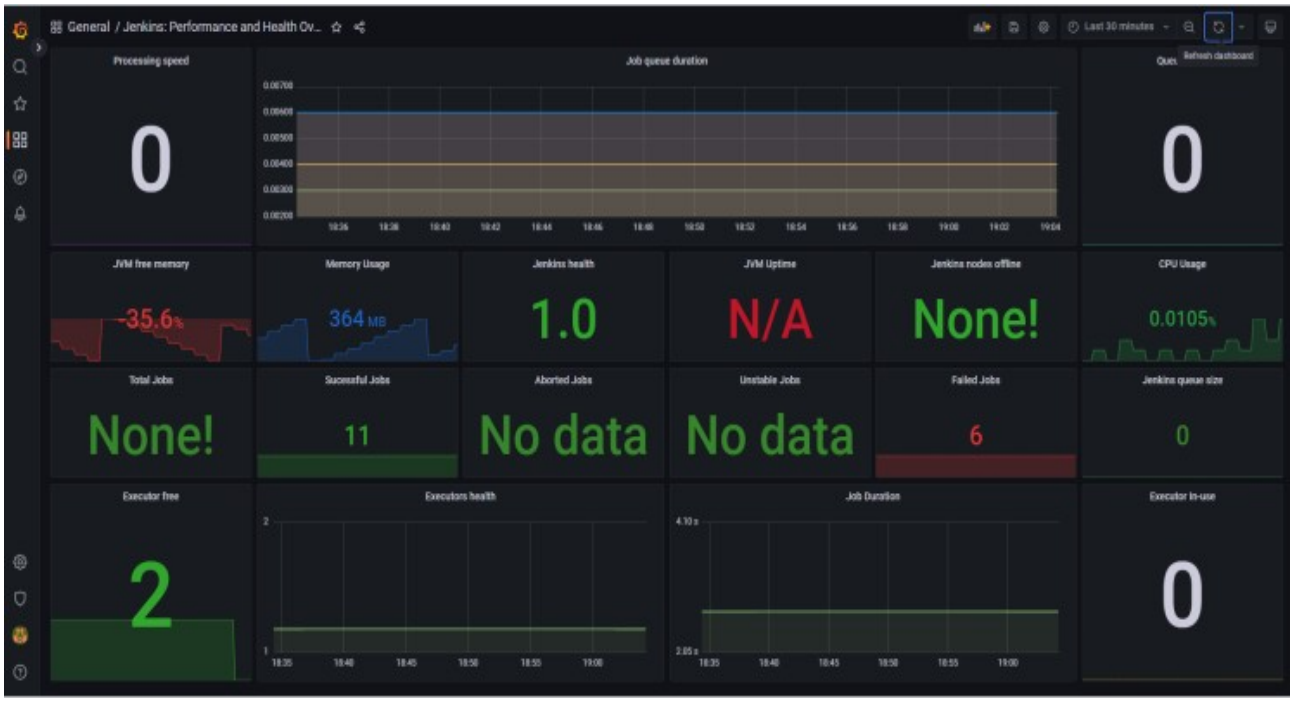

172.17.0.1:3100/dashboard/import
80%
Customer Portal
Red Hat
Red Hat Products Doc…
Red Hat Enterprise Lin…
Red Hat Developer Por…
Red Hat Container Cat…
Red Hat Hybrid Cloud …
Search or jump to…
ctrl+k
Home › Dashboards › Import dashboard
Import dashboard
Import dashboard from file or Grafana.com
Upload dashboard JSON file
Drag and drop here or click to browse
Accepted file types: .json, .txt
Find and import dashboards for common applications at grafana.com/dashboards
9964
Load
Import via dashboard JSON model
{
"title": "Example - Repeating Dictionary variables",
"uid": "_0HnEoN4z",
"panels": [...]
...
}
Load
Cancel


172.17.0.1:3100/dashboard/import
80%
Customer Portal
Red Hat
Red Hat Products Doc…
Red Hat Enterprise Lin…
Red Hat Developer Por…
Red Hat Container Cat…
Red Hat Hybrid Cloud …
Search or jump to…
ctrl+k
Home › Dashboards › Import dashboard
Import dashboard
Import dashboard from file or Grafana.com
Importing dashboard from Grafana.com
Published by
haryan
Updated on
2023-08-24 15:04:53
Options
Name
Jenkins: Performance and Health Overview
Folder
Dashboards
Unique identifier (UID)
The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.
haryan-jenkins
Change uid
Prometheus
Prometheus Data Source
Import
Cancel

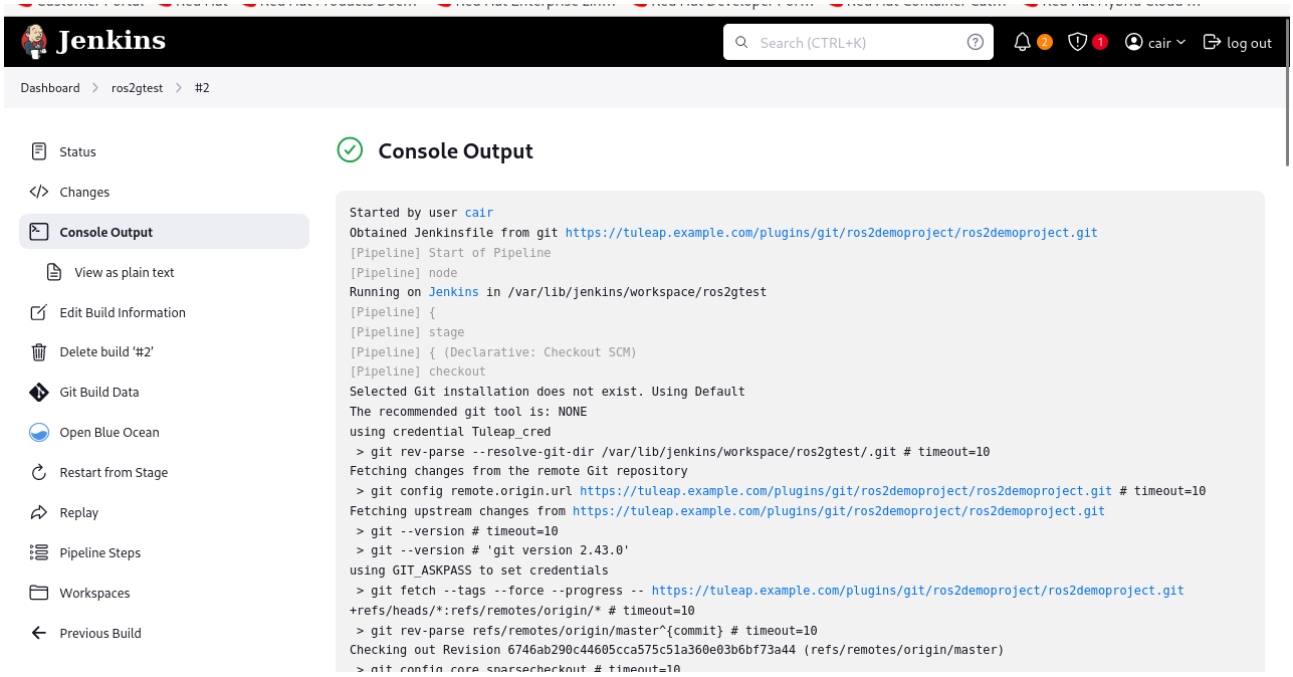
Monitoring Jenkins on Prometheus Grafana Dashboard:



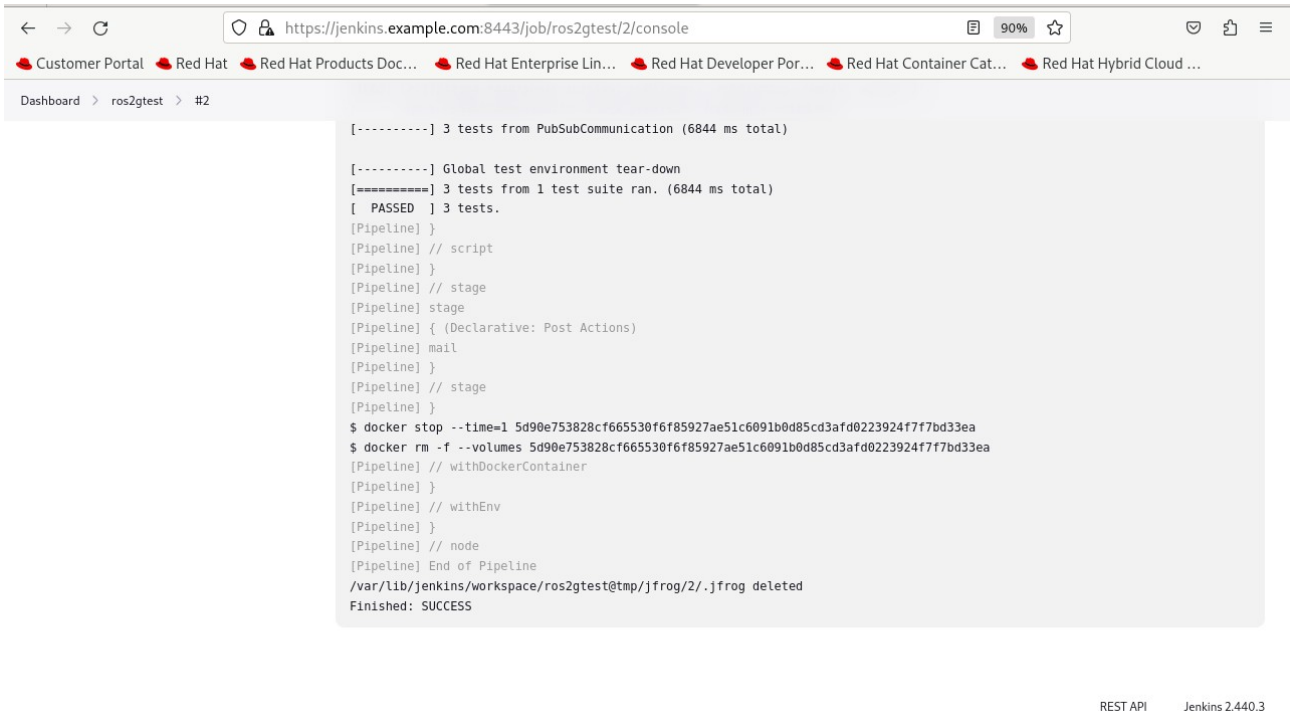


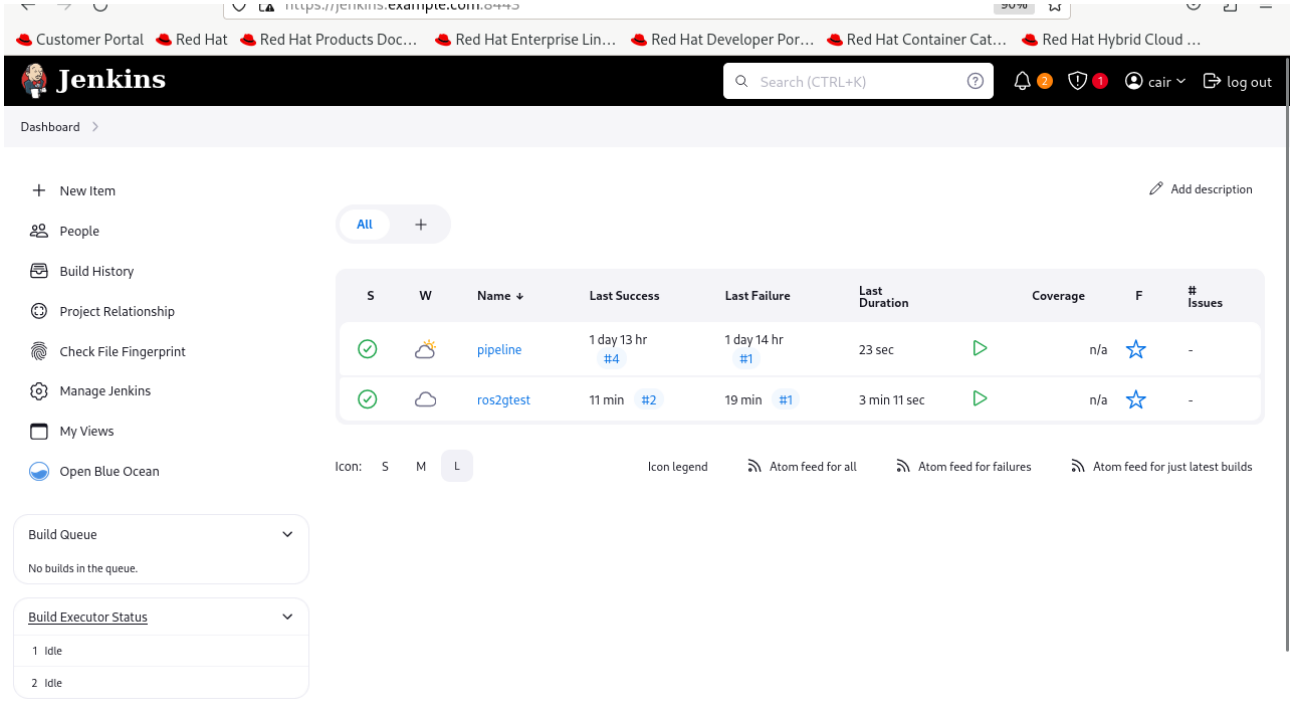


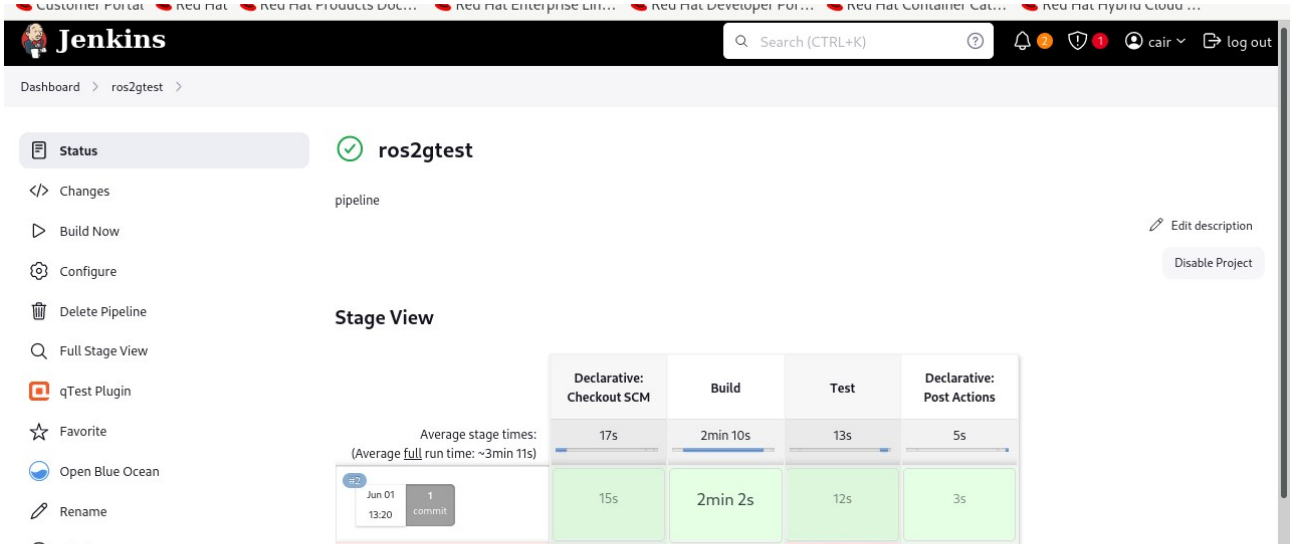
Final Project output:




Customer Portal
Red Hat
Red Hat Products Doc...
Red Hat Enterprise Lin...
Red Hat Developer Por...
Red Hat Container Cat...
Red Hat Hybrid Cloud ...
Dashboard
ros2gtest
#2
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Test)
[Pipeline] script
[Pipeline] {
[Pipeline] sh
+ cd /ros2_gtest/cpp_pubsub
+ ./test.sh
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from PubSubCommunication
[RUN] PubSubCommunication.pub_sub_communication
[INFO] [1717228378.469796712] [minimal_publisher]: Publishing: 'Hello, world! 0'
[OK] PubSubCommunication.pub_sub_communication (1616 ms)
[RUN] PubSubCommunication.check_received_msg
[INFO] [1717228380.604806245] [minimal_publisher]: Publishing: 'Hello, world! 0'
[OK] PubSubCommunication.check_received_msg (2118 ms)
[RUN] PubSubCommunication.check_publisher_frequency
[INFO] [1717228382.281160052] [minimal_publisher]: Publishing: 'Hello, world! 0'
[INFO] [1717228383.044398978] [minimal_publisher]: Publishing: 'Hello, world! 1'
[INFO] [1717228384.044473863] [minimal_publisher]: Publishing: 'Hello, world! 2'
[OK] PubSubCommunication.check_publisher_frequency (3110 ms)
[----------] 3 tests from PubSubCommunication (6844 ms total)
[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (6844 ms total)
[PASSED] 3 tests.
[Pipeline] }
[Pipeline] // script


https://jenkins.example.com:8443/job/ros2gtest/2/console
90%
Customer Portal
Red Hat
Red Hat Products Doc...
Red Hat Enterprise Lin...
Red Hat Developer Por...
Red Hat Container Cat...
Red Hat Hybrid Cloud ...
Dashboard
ros2gtest
#2
[----------] 3 tests from PubSubCommunication (6844 ms total)
[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (6844 ms total)
[PASSED] 3 tests.
[Pipeline] }
[Pipeline] // script
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] mail
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
$ docker stop --time=1 5d90e753828cf665530f6f85927ae51c6091b0d85cd3afd0223924f7f7bd33ea
$ docker rm -f --volumes 5d90e753828cf665530f6f85927ae51c6091b0d85cd3afd0223924f7f7bd33ea
[Pipeline] // withDockerContainer
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
/var/lib/jenkins/workspace/ros2gtest@tmp/jfrog/2/.jfrog deleted
Finished: SUCCESS


Customer Portal
Red Hat
Red Hat Products Doc…
Red Hat Enterprise Lin…
Red Hat Developer Por…
Red Hat Container Cat…
Red Hat Hybrid Cloud …
Jenkins
Search (CTRL+K)
cair
log out
Dashboard
New Item
People
Build History
Project Relationship
Check File Fingerprint
Manage Jenkins
My Views
Open Blue Ocean
Add description
All
S
W
Name
Last Success
Last Failure
Last Duration
Coverage
F
Issues
pipeline
1 day 13 hr #4
1 day 14 hr #1
23 sec
n/a
ros2gtest
11 min #2
19 min #1
3 min 11 sec
n/a
Icon: S M L
Icon legend
Atom feed for all
Atom feed for failures
Atom feed for just latest builds
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle


Jenkins
Search (CTRL+K)
cair
log out
Dashboard
ros2gtest
Status
Changes
Build Now
Configure
Delete Pipeline
Full Stage View
qTest Plugin
Favorite
Open Blue Ocean
Rename
ros2gtest
pipeline
Edit description
Disable Project
Stage View
Declarative: Checkout SCM
Build
Test
Declarative: Post Actions
Average stage times:
(Average full run time: ~3min 11s)
17s
2min 10s
13s
5s
Jun 01
13:20
1 commit
15s
2min 2s
12s
3s