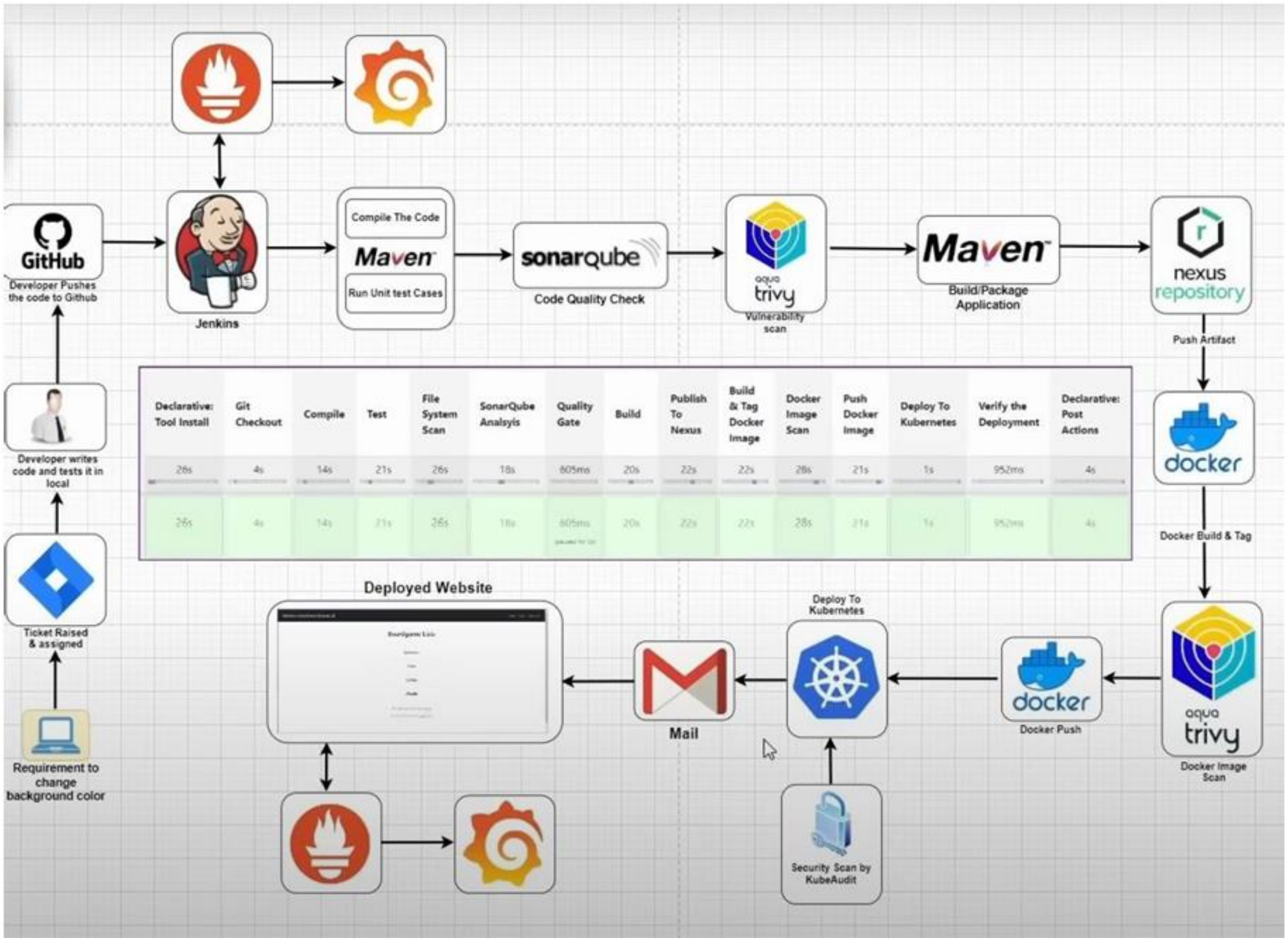
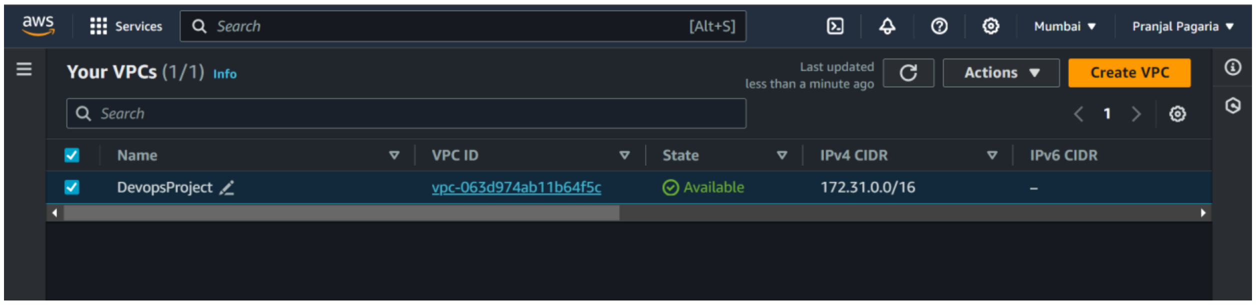
END TO END CI/CD PROJECT



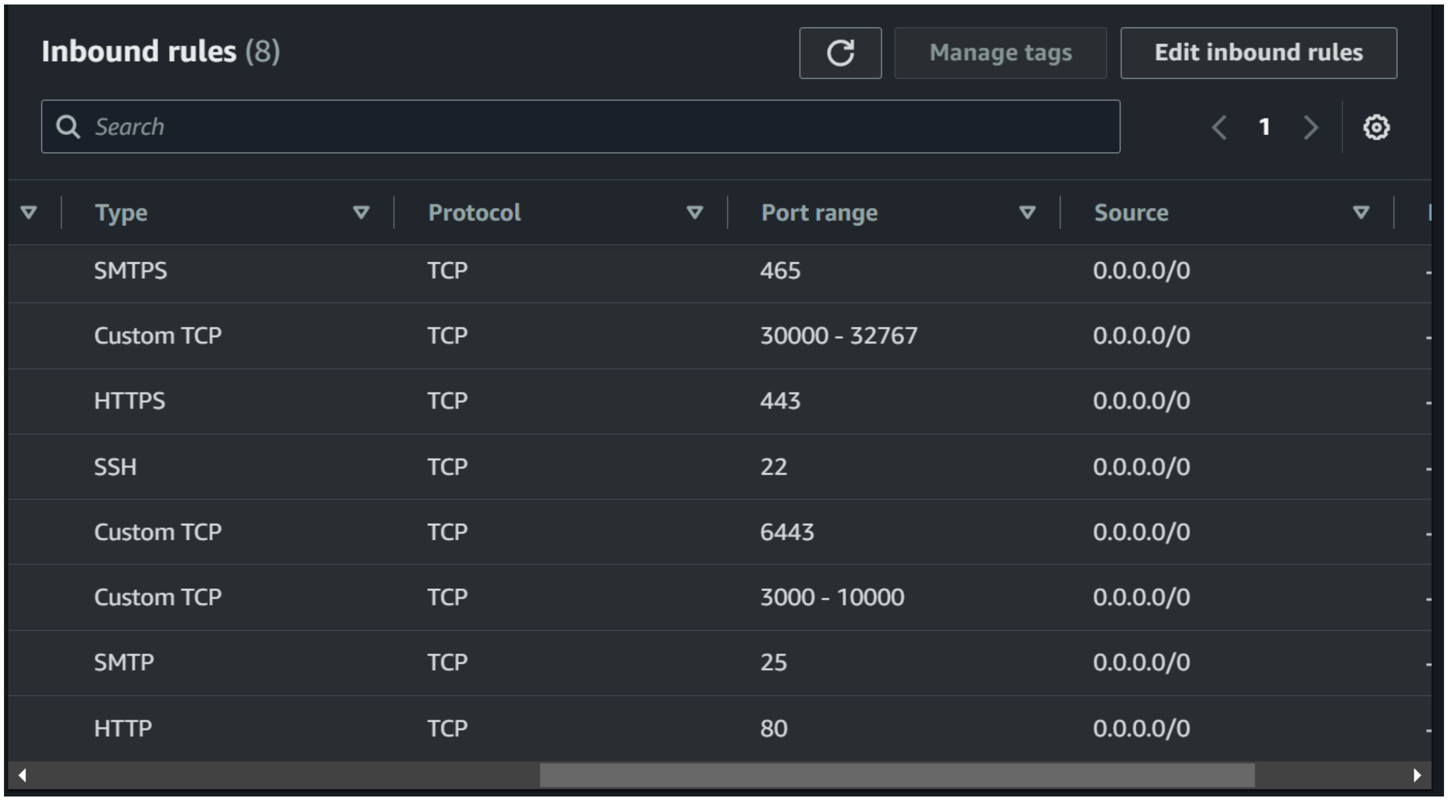
Phase-1

Firstly we have to create our VPC. Here I am using a default vpc.



Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR
DevopsProject	vpc-063d974ab11b64f5c	Available	172.31.0.0/16	–

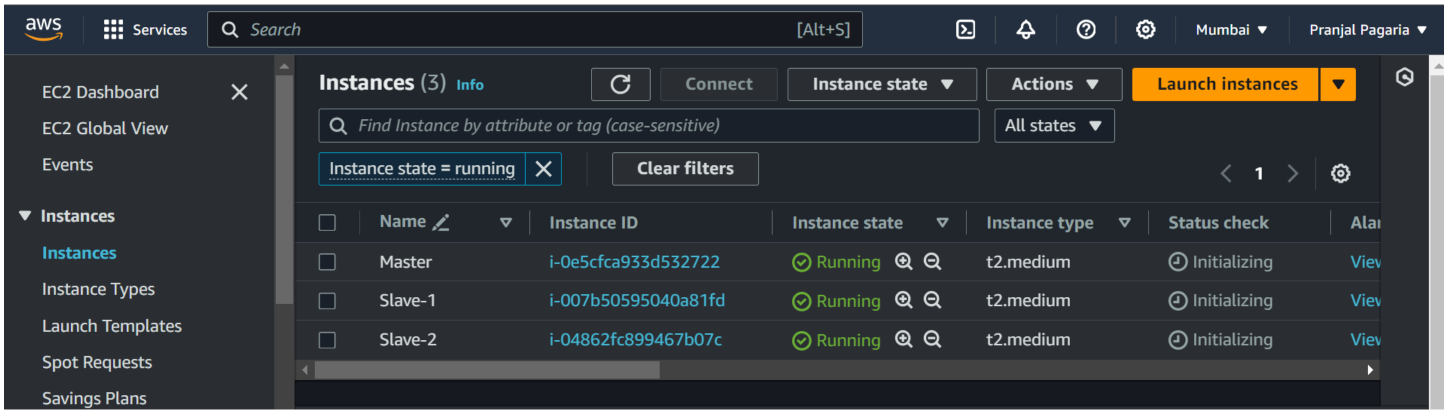
After this create your own security Group as below



Inbound rules (8)

Type	Protocol	Port range	Source
SMTPS	TCP	465	0.0.0.0/0
Custom TCP	TCP	30000 - 32767	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0
Custom TCP	TCP	6443	0.0.0.0/0
Custom TCP	TCP	3000 - 10000	0.0.0.0/0
SMTP	TCP	25	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0

Now we have to create a 3 virtual machine with instance type as t2-medium with 25 Gb root volume each of Ubuntu machine.

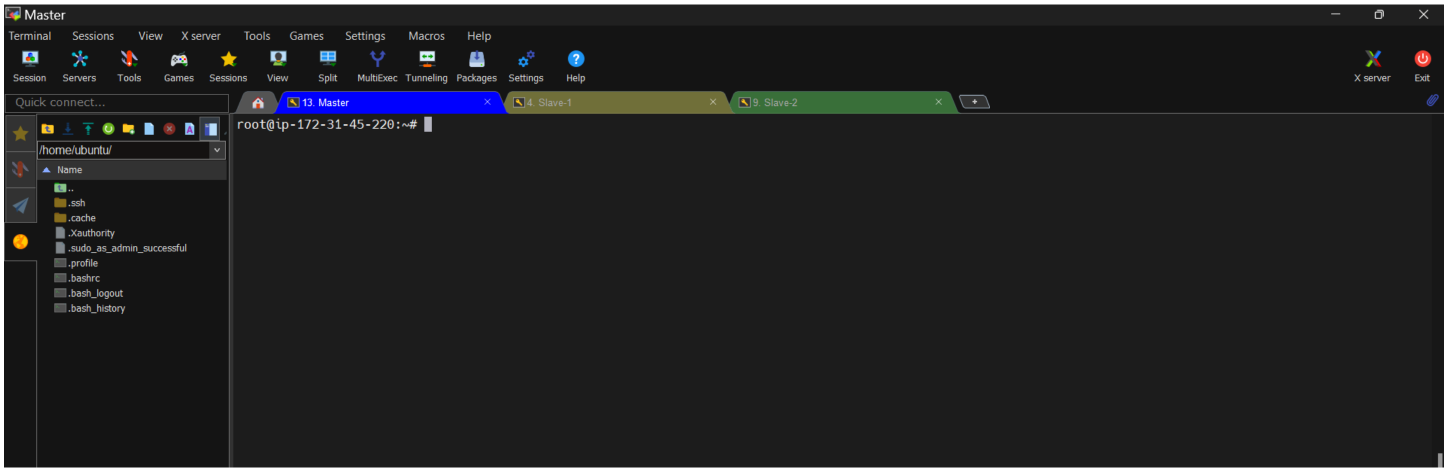


Instances (3) Info

Instance state = running

Name	Instance ID	Instance state	Instance type	Status check	Alar
Master	i-0e5cfca933d532722	Running	t2.medium	Initializing	View
Slave-1	i-007b50595040a81fd	Running	t2.medium	Initializing	View
Slave-2	i-04862fc899467b07c	Running	t2.medium	Initializing	View

Now we have to take access in all three machines


Master
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
X server Exit
Quick connect...
13. Master
4. Slave-1
9. Slave-2
/home/ubuntu/
Name
..
.ssh
.cache
.Xauthority
.sudo_as_admin_successful
.profile
.bashrc
.bash_logout
.bash_history

```
root@ip-172-31-45-220:~#
```

Now we have to configure Kubernetes cluster in all three machines

# Setup K8-Cluster using kubeadm [K8 Version-->1.28.1]

### 1. Update System Packages [On Master & Worker Node]

sudo apt-get update

### 2. Install Docker[On Master & Worker Node]

sudo apt install docker.io -y

sudo chmod 666 /var/run/docker.sock

### 3. Install Required Dependencies for Kubernetes[On Master & Worker Node]

sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

sudo mkdir -p -m 755 /etc/apt/keyrings

### 4. Add Kubernetes Repository and GPG Key[On Master & Worker Node]

curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

### 5. Update Package List[On Master & Worker Node]

sudo apt update

### 6. Install Kubernetes Components[On Master & Worker Node]

sudo apt install -y kubeadm=1.28.1-1.1 kubelet=1.28.1-1.1 kubectl=1.28.1-1.1

### 7. Initialize Kubernetes Master Node [On MasterNode]

sudo kubeadm init --pod-network-cidr=10.244.0.0/16

### 8. Configure Kubernetes Cluster [On MasterNode]

mkdir -p $HOME/.kube

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

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

### 9. Deploy Networking Solution (Calico) [On MasterNode]

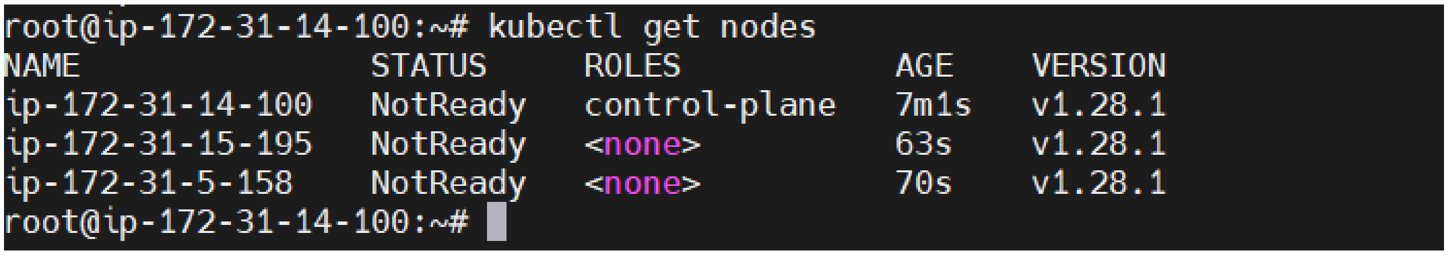
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

### 10. Deploy Ingress Controller (NGINX) [On MasterNode]

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v0.49.0/deploy/static/provider/baremetal/deploy.yaml

This we have to run in our slave machines
kubeadm join 172.31.14.100:6443 --token rue25m.0pt3qpgsm9ltpnwy \

--discovery-token-ca-cert-hash sha256:5b09ccc5ad6a61f79ef7d4e9691d0f3d592940159f1ade44f7b4ed21aaaf1b52

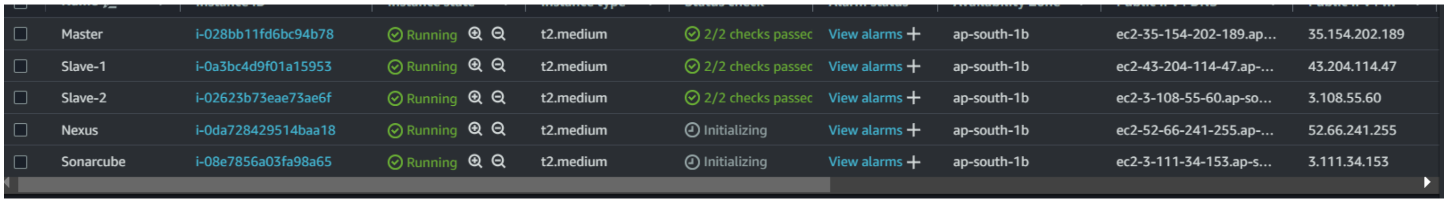


```
root@ip-172-31-14-100:~# kubectl get nodes
NAME               STATUS     ROLES           AGE    VERSION
ip-172-31-14-100   NotReady   control-plane   7m1s   v1.28.1
ip-172-31-15-195   NotReady   <none>          63s    v1.28.1
ip-172-31-5-158    NotReady   <none>          70s    v1.28.1
root@ip-172-31-14-100:~#
```

For security check we are using kubeaudit
steps:
https://github.com/Shopify/kubeaudit/releases
Then select amd 64 and copy the link and paste it with wget command as
wget https://github.com/Shopify/kubeaudit/releases/download/v0.22.1/kubeaudit_0.22.1_linux_amd64.tar.gz
sudo mv kubeaudit /usr/local/bin/
kubeaudit all

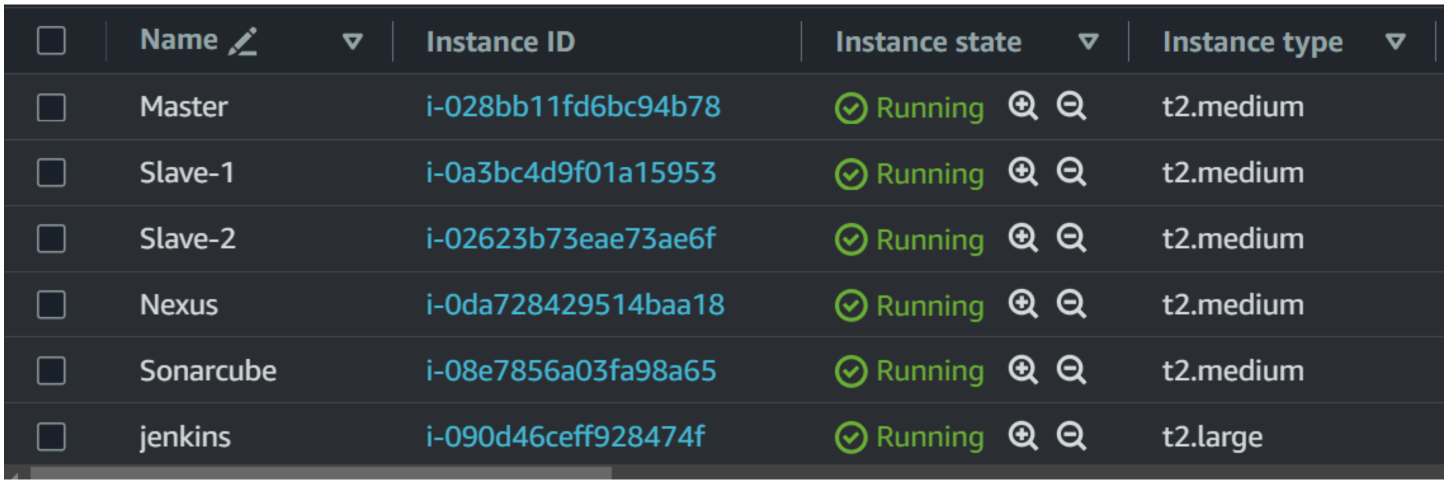
Now we have to setup Three more servers where we can configure Jenkins , Nexus and SonarQube.

Firstly we have two create two servers for Sonarqube and Nexus
Ubuntu server of T2-medium type with 20 gb root volume.



☐	Master	i-028bb11fd6bc94b78	✓ Running	t2.medium	✓ 2/2 checks passec	View alarms +	ap-south-1b	ec2-35-154-202-189.ap...	35.154.202.189
☐	Slave-1	i-0a3bc4d9f01a15953	✓ Running	t2.medium	✓ 2/2 checks passec	View alarms +	ap-south-1b	ec2-43-204-114-47.ap-...	43.204.114.47
☐	Slave-2	i-02623b73eae73ae6f	✓ Running	t2.medium	✓ 2/2 checks passec	View alarms +	ap-south-1b	ec2-3-108-55-60.ap-so...	3.108.55.60
☐	Nexus	i-0da728429514baa18	✓ Running	t2.medium	Initializing	View alarms +	ap-south-1b	ec2-52-66-241-255.ap-...	52.66.241.255
☐	Sonarcube	i-08e7856a03fa98a65	✓ Running	t2.medium	Initializing	View alarms +	ap-south-1b	ec2-3-111-34-153.ap-s...	3.111.34.153

Now we have to configure Jenkins : Ubuntu machine of t2-large type and with 30 gb volume.



☐	Name	Instance ID	Instance state	Instance type
☐	Master	i-028bb11fd6bc94b78	✓ Running	t2.medium
☐	Slave-1	i-0a3bc4d9f01a15953	✓ Running	t2.medium
☐	Slave-2	i-02623b73eae73ae6f	✓ Running	t2.medium
☐	Nexus	i-0da728429514baa18	✓ Running	t2.medium
☐	Sonarcube	i-08e7856a03fa98a65	✓ Running	t2.medium
☐	jenkins	i-090d46ceff928474f	✓ Running	t2.large

SonarQube Setup and Nexus setup

First run this command in both the servers ->Sudo apt update
Now we have to install docker in both
sudo apt update

Make one file as vi dock.sh

use this script to install docker

## script starts from here

#!/bin/bash

# Update package manager repositories

sudo apt-get update

# Install necessary dependencies

```
sudo apt-get install -y ca-certificates curl

# Create directory for Docker GPG key
sudo install -m 0755 -d /etc/apt/keyrings

# Download Docker's GPG key
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

# Ensure proper permissions for the key
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add Docker repository to Apt sources
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# Update package manager repositories
sudo apt-get update

sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

## script ends here
chmod +x dock.sh
```

To give access to other usser to run docker use the following command

```
sudo chmod 666 /var/run/docker.sock
```

For SonarQube
Create Sonarqube Docker container

To run SonarQube in a Docker container with the provided command, you can follow these steps:

Open your terminal or command prompt.

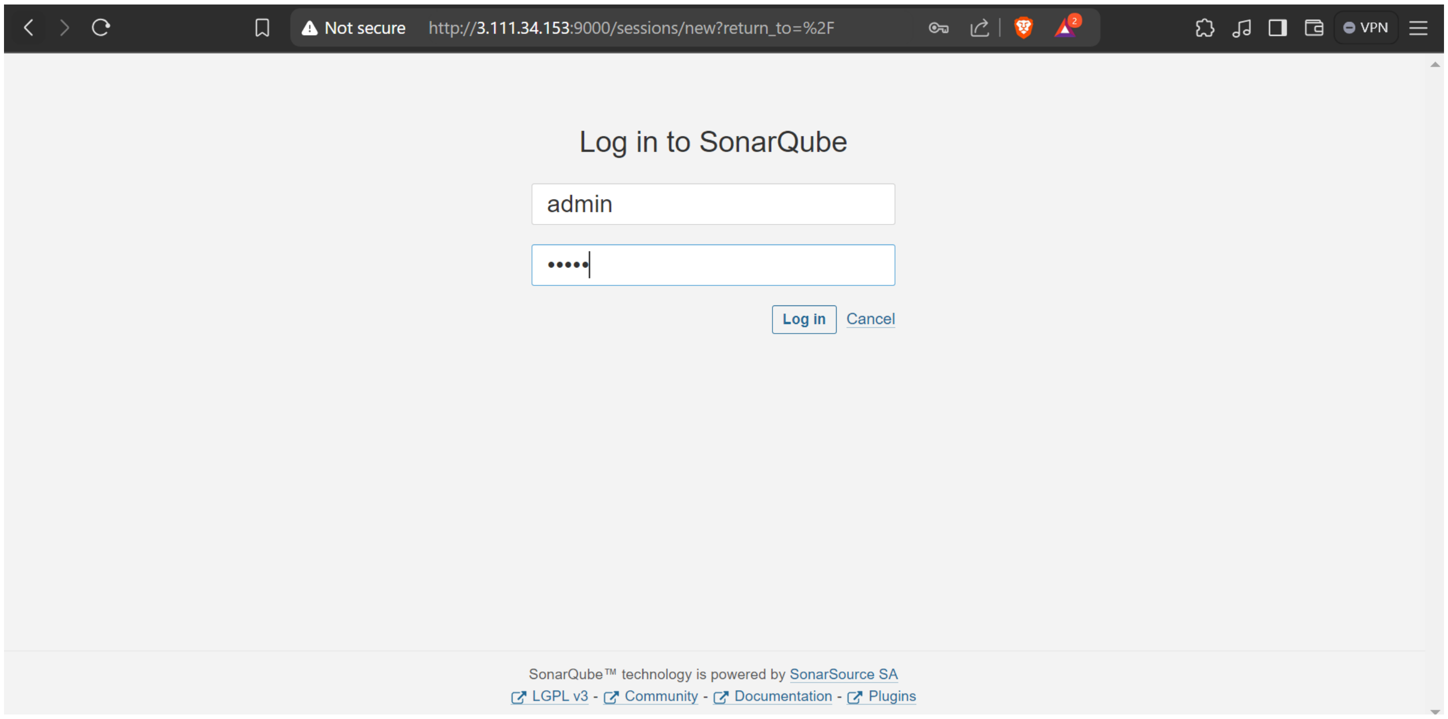
Run the following command:

```
docker run -d --name sonar -p 9000:9000 sonarqube:lts-community
```

This command will download the sonarqube:lts-community Docker image from Docker Hub if it's not already available locally. Then, it will create a container named "sonar" from this image, running it in detached mode (-d flag) and mapping port 9000 on the host machine to port 9000 in the container (-p 9000:9000 flag).

Access SonarQube by opening a web browser and navigating to http://VmIP:9000.

This will start the SonarQube server, and you should be able to access it using the provided URL. If you're running Docker on a remote server or a different port, replace localhost with the appropriate hostname or IP address and adjust the port accordingly.



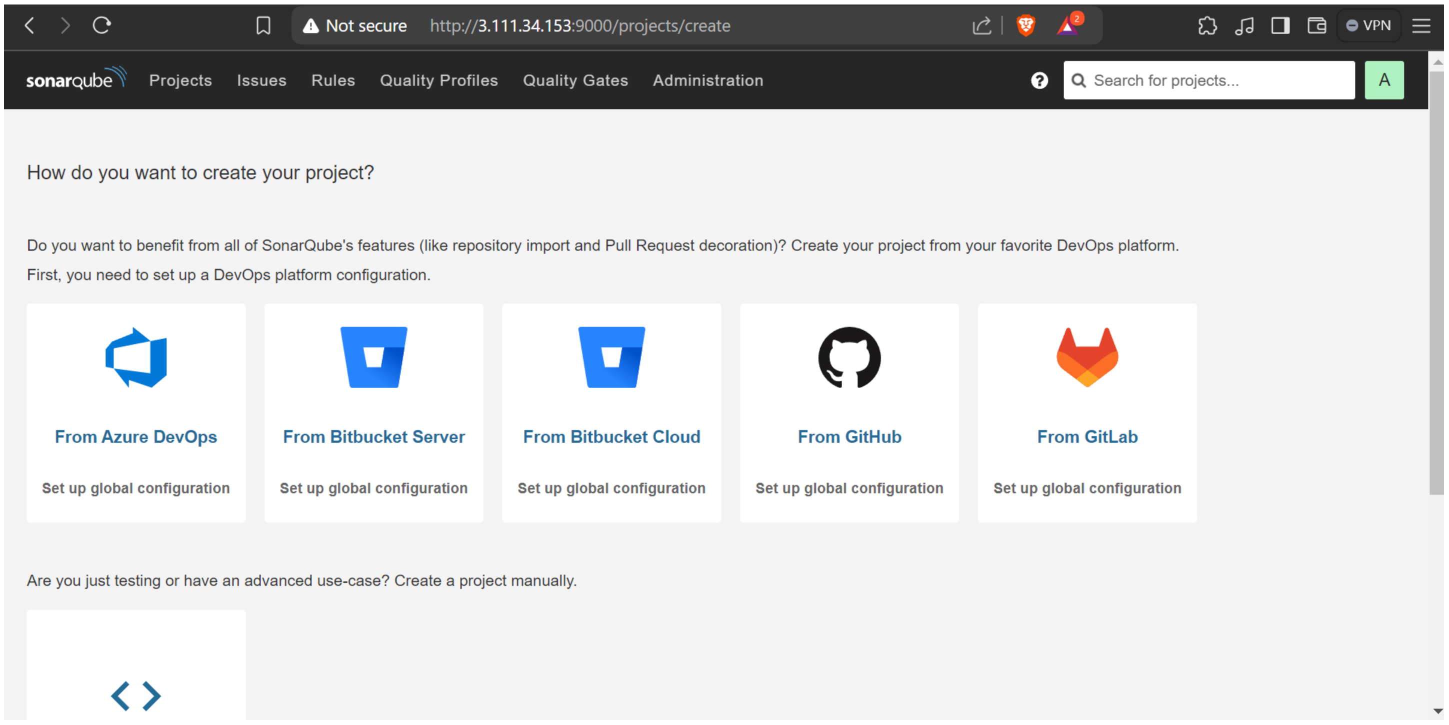
Log in to SonarQube

admin

•••••

Log in Cancel

SonarQube™ technology is powered by SonarSource SA
LGPL v3 - Community - Documentation - Plugins


Not secure http://3.111.34.153:9000/projects/create

How do you want to create your project?

Do you want to benefit from all of SonarQube's features (like repository import and Pull Request decoration)? Create your project from your favorite DevOps platform.
First, you need to set up a DevOps platform configuration.

From Azure DevOps
Set up global configuration
From Bitbucket Server
Set up global configuration
From Bitbucket Cloud
Set up global configuration
From GitHub
Set up global configuration
From GitLab
Set up global configuration

Are you just testing or have an advanced use-case? Create a project manually.

For nexus server

Create Nexus using docker container

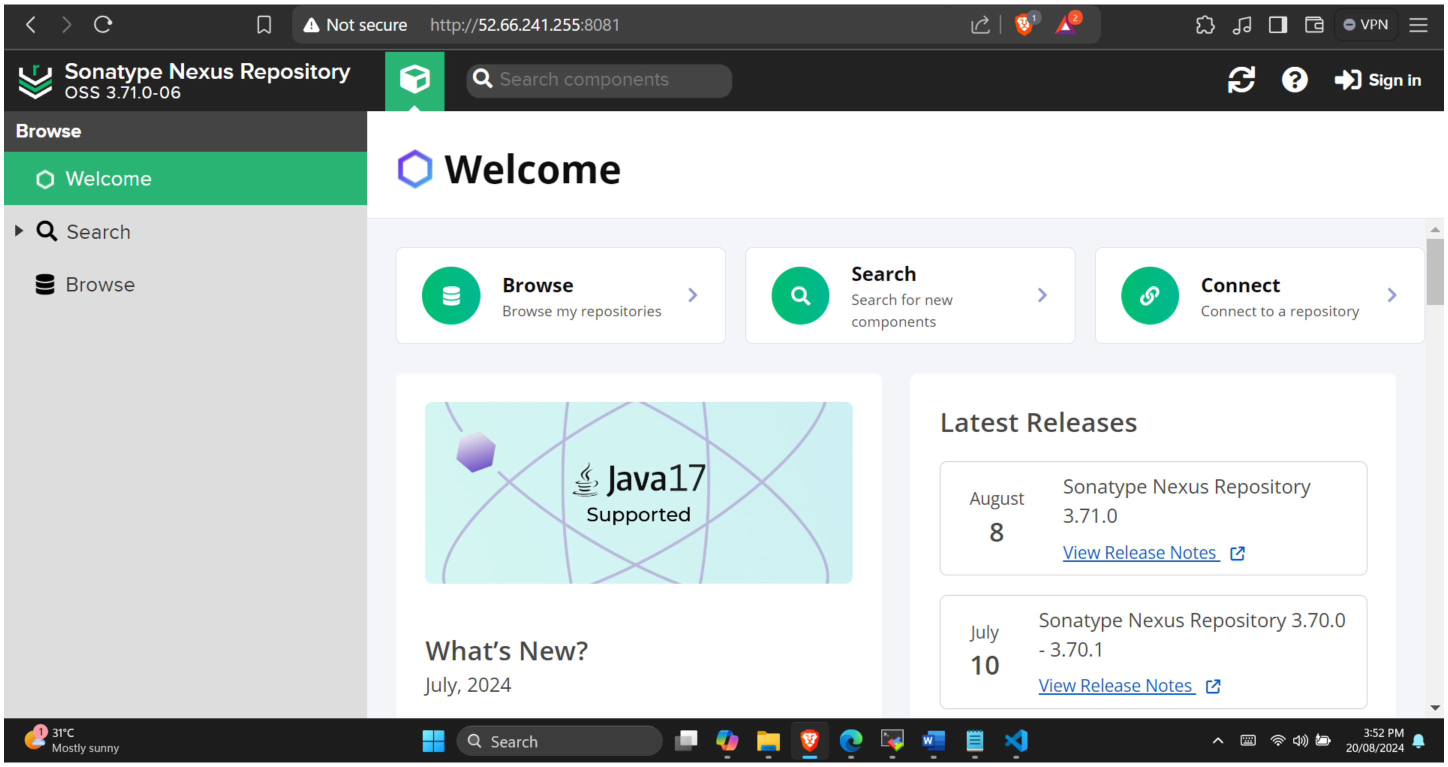
To create a Docker container running Nexus 3 and exposing it on port 8081, you can use the following command:

```
docker run -d --name nexus -p 8081:8081 sonatype/nexus3:latest
```

This command does the following:

- -d: Detaches the container and runs it in the background.
- --name nexus: Specifies the name of the container as "nexus".
- -p 8081:8081: Maps port 8081 on the host to port 8081 on the container, allowing access to Nexus through port 8081.
- sonatype/nexus3:latest: Specifies the Docker image to use for the container, in this case, the latest version of Nexus 3 from the Sonatype repository.

After running this command, Nexus will be accessible on your host machine at http://IP:8081.



Get Nexus initial password

Your provided commands are correct for accessing the Nexus password stored in the container. Here's a breakdown of the steps:

1. **Get Container ID**: You need to find out the ID of the Nexus container. You can do this by running:

docker ps

This command lists all running containers along with their IDs, among other information.

2. **Access Container's Bash Shell**: Once you have the container ID, you can execute the docker exec command to access the container's bash shell:

docker exec -it <container_ID> /bin/bash

Replace <container_ID> with the actual ID of the Nexus container.

3. **Navigate to Nexus Directory**: Inside the container's bash shell, navigate to the directory where Nexus stores its configuration:

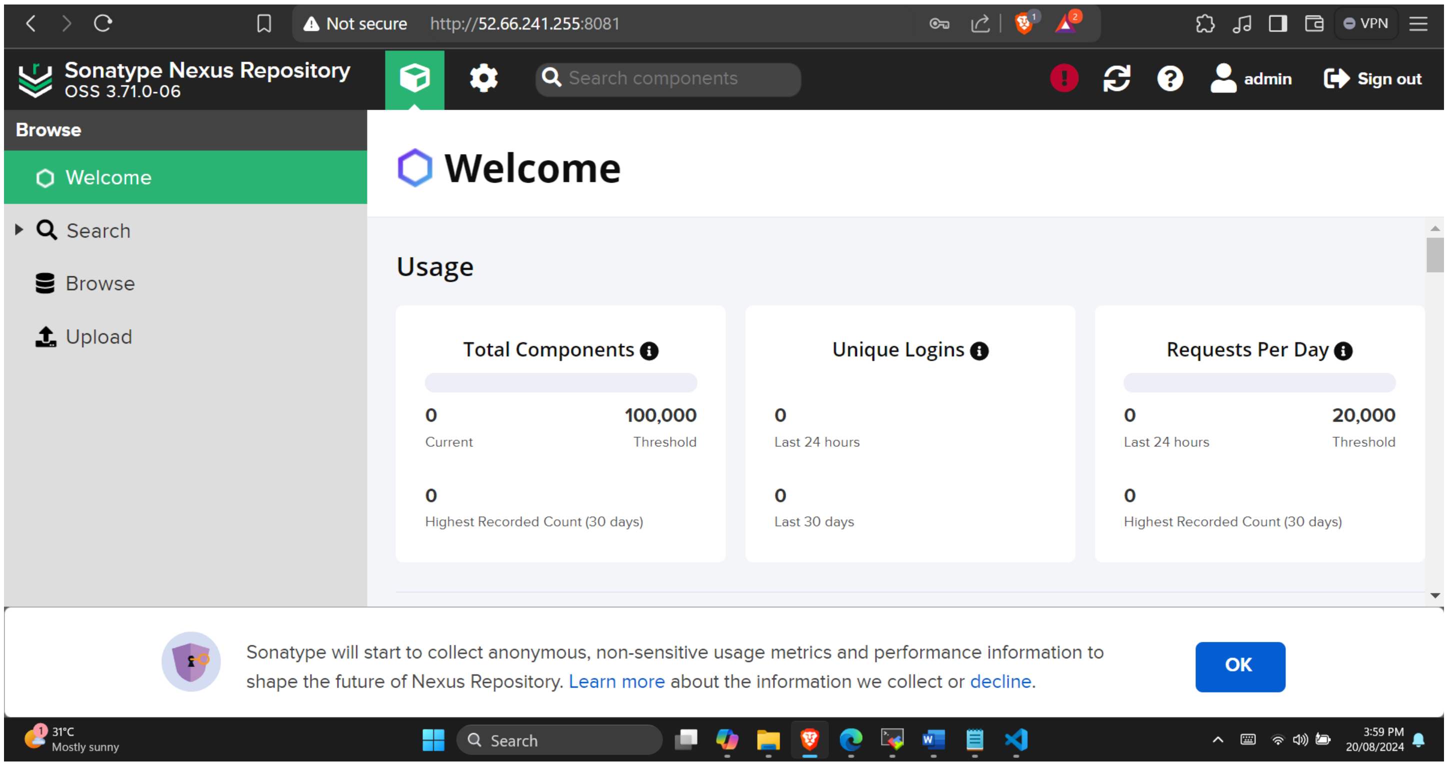
cd sonatype-work/nexus3

4. **View Admin Password**: Finally, you can view the admin password by displaying the contents of the admin.password file:

cat admin.password

5. **Exit the Container Shell**: Once you have retrieved the password, you can exit the container's bash shell:

```
exit
```


Not secure http://52.66.241.255:8081
Sonatype Nexus Repository OSS 3.71.0-06
Search components
admin
Sign out
Browse
Welcome
Search
Browse
Upload

Welcome

Usage

Total Components		Unique Logins	Requests Per Day	
0 Current	100,000 Threshold	0 Last 24 hours	0 Last 24 hours	20,000 Threshold
0 Highest Recorded Count (30 days)		0 Last 30 days	0 Highest Recorded Count (30 days)	

Sonatype will start to collect anonymous, non-sensitive usage metrics and performance information to shape the future of Nexus Repository. Learn more about the information we collect or decline.

Now we have to setup Jenkins
Installing Jenkins on Ubuntu

```
#!/bin/bash


# Install OpenJDK 17 JRE Headless

sudo apt install openjdk-17-jre-headless -y


# Download Jenkins GPG key

sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \

  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key


# Add Jenkins repository to package manager sources

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \

  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null

# Update package manager repositories
sudo apt-get update

# Install Jenkins
sudo apt-get install jenkins -y
```

Save this script in a file, for example, install_jenkins.sh, and make it executable using:

```
chmod +x install_jenkins.sh
```

Then, you can run the script using:

```
./install_jenkins.sh
```

This script will automate the installation process of OpenJDK 17 JRE Headless and Jenkins.

Install docker for future use

```
#!/bin/bash

# Update package manager repositories
sudo apt-get update

# Install necessary dependencies
sudo apt-get install -y ca-certificates curl

# Create directory for Docker GPG key
sudo install -m 0755 -d /etc/apt/keyrings

# Download Docker's GPG key
```

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc


# Ensure proper permissions for the key

sudo chmod a+r /etc/apt/keyrings/docker.asc


# Add Docker repository to Apt sources

echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null


# Update package manager repositories

sudo apt-get update


sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

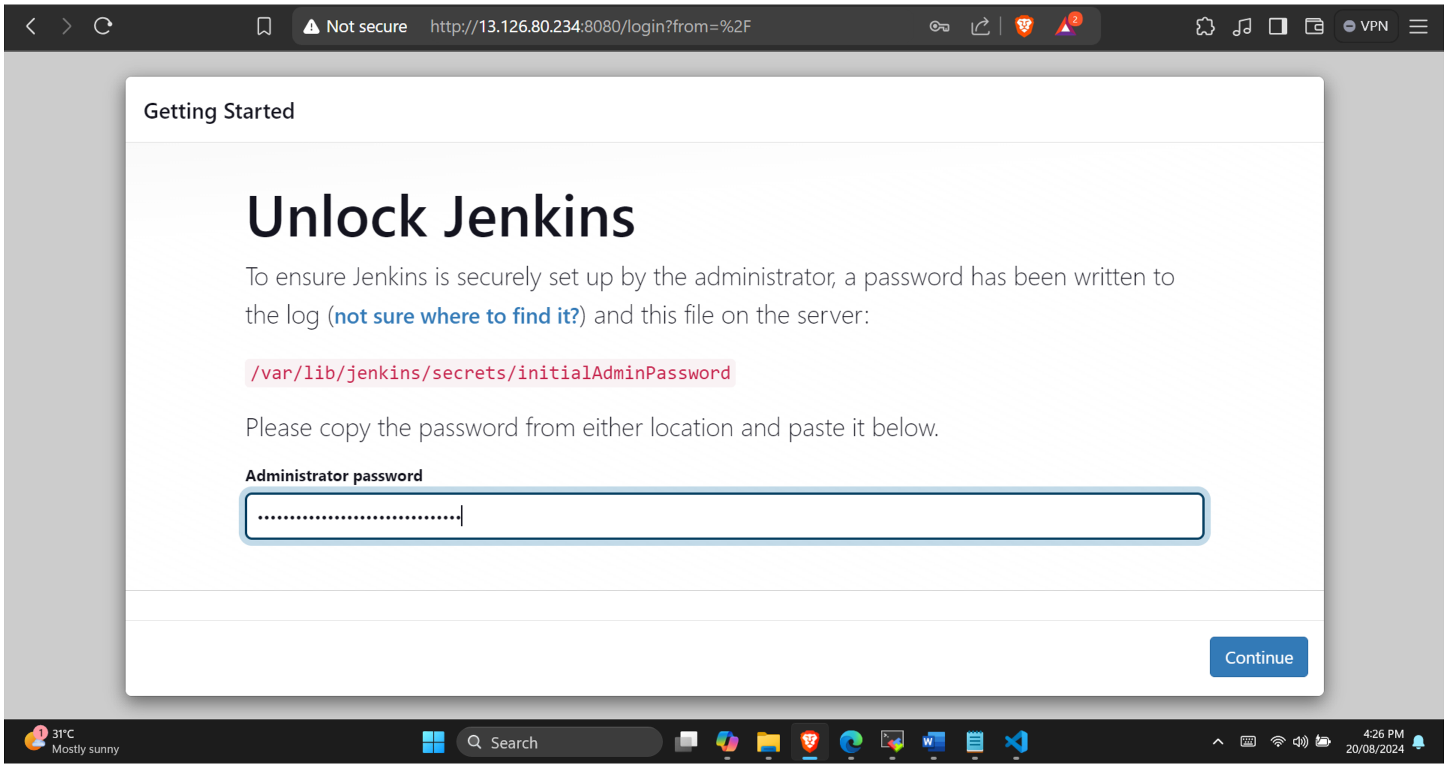
Save this script in a file, for example, install_docker.sh, and make it executable using:

```
chmod +x install_docker.sh
```

Then, you can run the script using:

```
./install_docker.sh
```

```
sudo chmod 666 /var/run/docker.sock
```



Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

```
/var/lib/jenkins/secrets/initialAdminPassword
```

Please copy the password from either location and paste it below.

Administrator password

Phase 2

Steps to create a private Git repository, generate a personal access token, connect to the repository, and push code to it:

1. Create a Private Git Repository:
   - Go to your preferred Git hosting platform (e.g., GitHub, GitLab, Bitbucket).
   - Log in to your account or sign up if you don't have one.
   - Create a new repository and set it as private.
2. Generate a Personal Access Token:
   - Navigate to your account settings or profile settings.
   - Look for the "Developer settings" or "Personal access tokens" section.
   - Generate a new token, providing it with the necessary permissions (e.g., repo access).
3. Clone the Repository Locally:
   - Open Git Bash or your terminal.
   - Navigate to the directory where you want to clone the repository.
   - Use the git clone command followed by the repository's URL. For example:

git clone <repository_URL>

4. Replace <repository_URL> with the URL of your private repository.
5. Add Your Source Code Files:
   - Navigate into the cloned repository directory.
   - Paste your source code files or create new ones inside this directory.
6. Stage and Commit Changes:
   - Use the git add command to stage the changes:

git add .

   - Use the git commit command to commit the staged changes along with a meaningful message:

git commit -m "Your commit message here"

7. Push Changes to the Repository:
   - Use the git push command to push your committed changes to the remote repository:

git push

   - If it's your first time pushing to this repository, you might need to specify the remote and branch:

git push -u origin master

8. Replace master with the branch name if you're pushing to a different branch.
9. Enter Personal Access Token as Authentication:
   - When prompted for credentials during the push, enter your username (usually your email) and use your personal access token as the password.

By following these steps, you'll be able to create a private Git repository, connect to it using Git Bash, and push your code changes securely using a personal access token for authentication.

Phase-3

We have to install certain plugins in Jenkins
Install Plugins in Jenkins

1. **Eclipse Temurin Installer**:
   - This plugin enables Jenkins to automatically install and configure the Eclipse Temurin JDK (formerly known as AdoptOpenJDK).

- To install, go to Jenkins dashboard -> Manage Jenkins -> Manage Plugins -> Available tab.
- Search for "Eclipse Temurin Installer" and select it.
- Click on the "Install without restart" button.

2. **Pipeline Maven Integration**:
   - This plugin provides Maven support for Jenkins Pipeline.
   - It allows you to use Maven commands directly within your Jenkins Pipeline scripts.
   - To install, follow the same steps as above, but search for "Pipeline Maven Integration" instead.
3. **Config File Provider**:
   - This plugin allows you to define configuration files (e.g., properties, XML, JSON) centrally in Jenkins.
   - These configurations can then be referenced and used by your Jenkins jobs.
   - Install it using the same procedure as mentioned earlier.
4. **SonarQube Scanner**:
   - SonarQube is a code quality and security analysis tool.
   - This plugin integrates Jenkins with SonarQube by providing a scanner that analyzes code during builds.
   - You can install it from the Jenkins plugin manager as described above.
5. **Kubernetes CLI**:
   - This plugin allows Jenkins to interact with Kubernetes clusters using the Kubernetes command-line tool (kubectl).
   - It's useful for tasks like deploying applications to Kubernetes from Jenkins jobs.
   - Install it through the plugin manager.
6. **Kubernetes**:
   - This plugin integrates Jenkins with Kubernetes by allowing Jenkins agents to run as pods within a Kubernetes cluster.
   - It provides dynamic scaling and resource optimization capabilities for Jenkins builds.
   - Install it from the Jenkins plugin manager.
7. **Docker**:
   - This plugin allows Jenkins to interact with Docker, enabling Docker builds and integration with Docker registries.

- You can use it to build Docker images, run Docker containers, and push/pull images from Docker registries.
- Install it from the plugin manager.

8. **Docker Pipeline Step**:
    - This plugin extends Jenkins Pipeline with steps to build, publish, and run Docker containers as part of your Pipeline scripts.
    - It provides a convenient way to manage Docker containers directly from Jenkins Pipelines.
    - Install it through the plugin manager like the others.

After installing these plugins, you may need to configure them according to your specific environment and requirements. This typically involves setting up credentials, configuring paths, and specifying options in Jenkins global configuration or individual job configurations. Each plugin usually comes with its own set of documentation to guide you through the configuration process.

We also need to install trivy on Jenkins
sudo apt-get install wget apt-transport-https gnupg lsb-release

wget -q0 https://aquasecurity.github.io/trivy-repo/deb/public.key | gpg --dearmor |sudo tee /usr/share/key rings/trivy.gpg> /dev/null

echo "deb [signed-by=/usr/share/keyrings/trivy.gpg] https://aquasecurity.github.io/trivy-repo/deb $(lsb_relea se -sc) main" | sudo tee -a /etc/apt/sources.list.d/trivy.list

sudo apt-get update

sudo apt-get install trivy -y

~

~

Now we installed the tools and Now we need to configure them

Go to → manage Jenkins→Tools→

1. Jdk→ name= jdk17 , install automatically from adoptium.net, version= jdk17 latest

2. Sonarqube scanner → name=sonar-scanner, Install automatically

3. Maven → name= maven3, version= 3.6.3

4. Docker→ name=docker, install automatically from docker.com

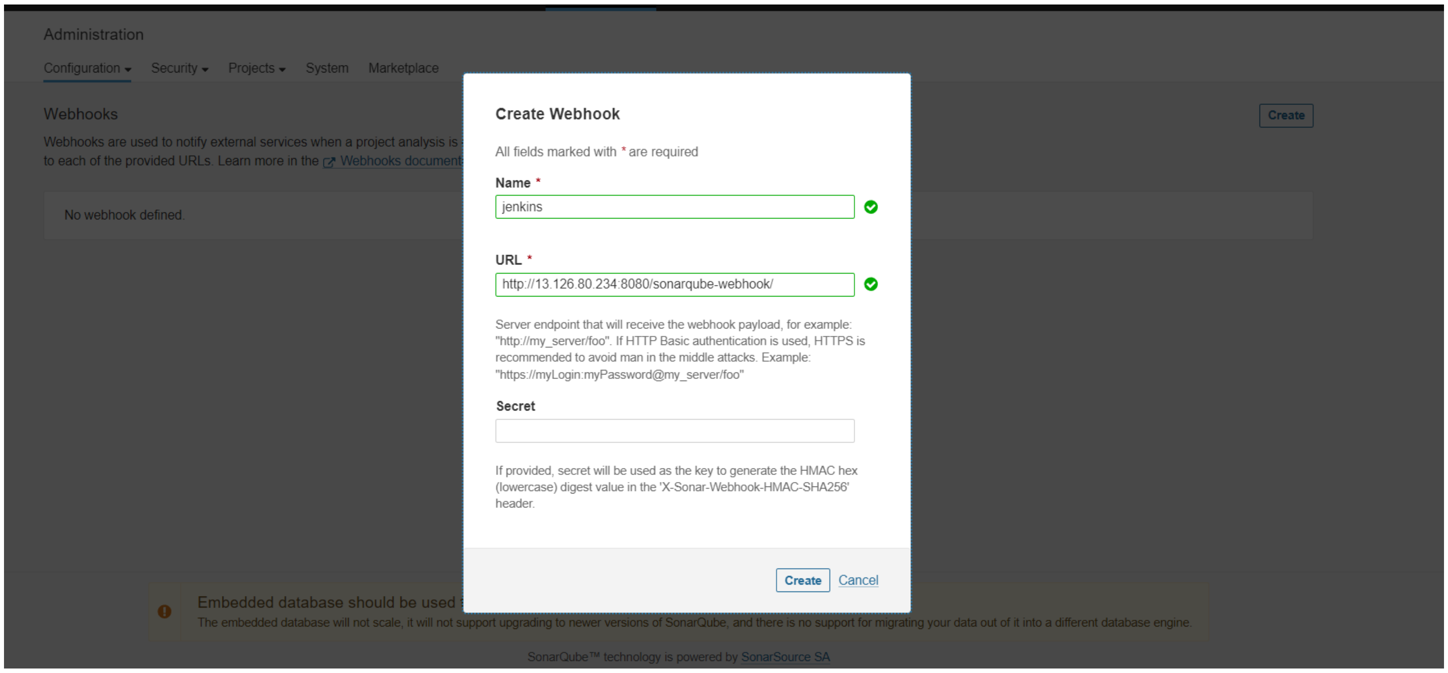
Now configure the sonarqube server in Jenkins

Firstly generate the token in sonarqube

Goto → Administaration→ security→ users→update token→ name= sonartoken and

Generate

Create webhook in sonarqube



Webhooks are used to notify external services when a project analysis is … to each of the provided URLs. Learn more in the Webhooks document…

No webhook defined.

Create Webhook

All fields marked with * are required

Name * jenkins

URL * http://13.126.80.234:8080/sonarqube-webhook/

Server endpoint that will receive the webhook payload, for example: "http://my_server/foo". If HTTP Basic authentication is used, HTTPS is recommended to avoid man in the middle attacks. Example: "https://myLogin:myPassword@my_server/foo"

Secret

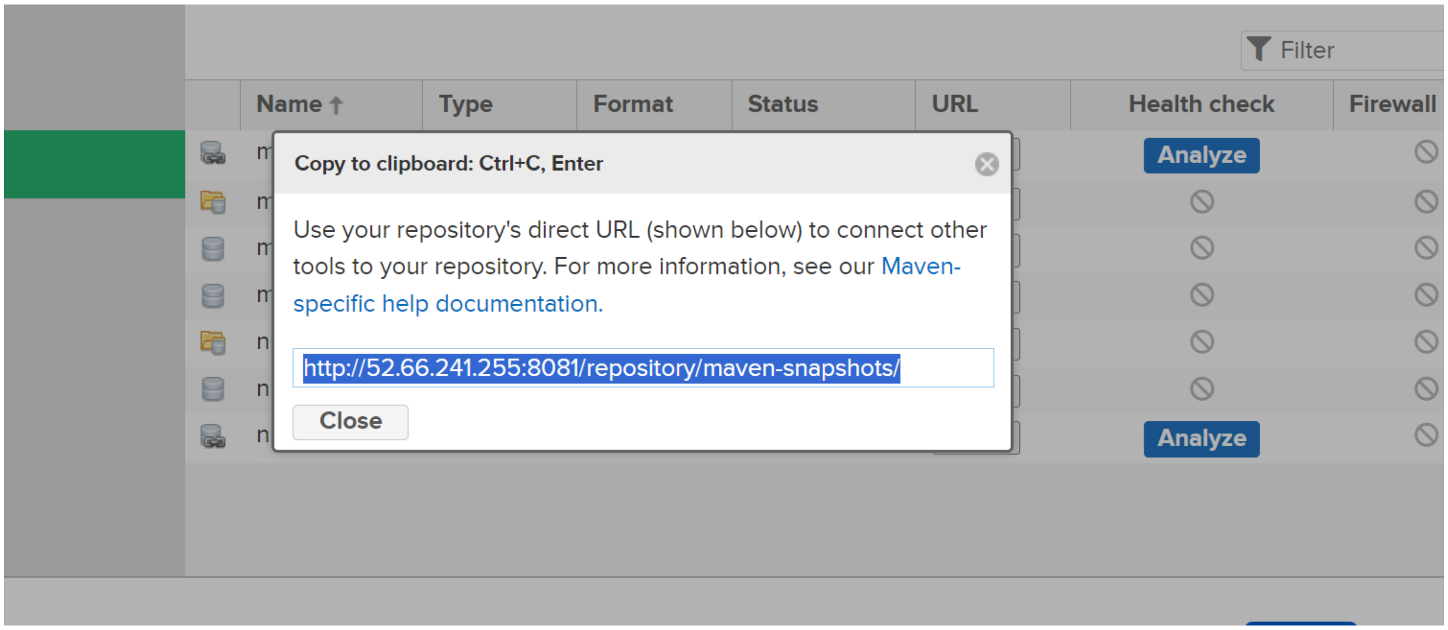
If provided, secret will be used as the key to generate the HMAC hex (lowercase) digest value in the 'X-Sonar-Webhook-HMAC-SHA256' header.

Create Cancel

Embedded database should be used …
The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA

Now we need to publish our artifacts to nexus

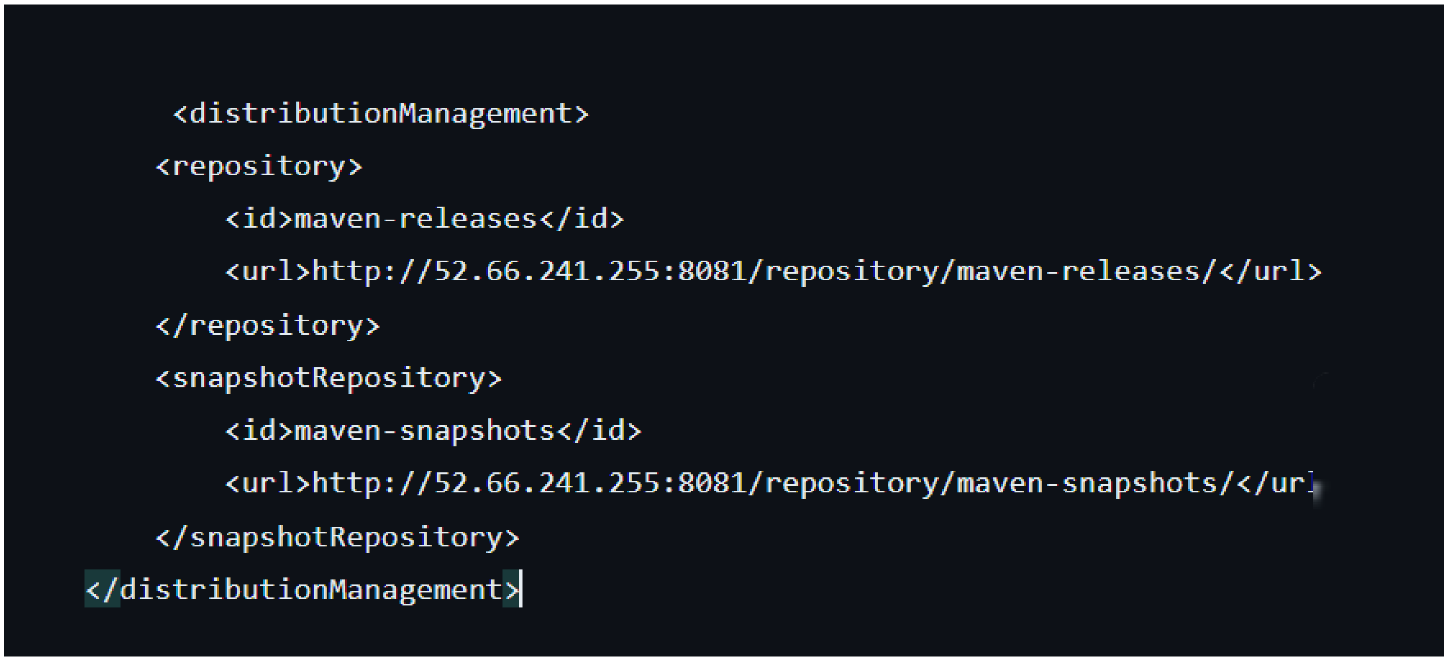


Copy to clipboard: Ctrl+C, Enter

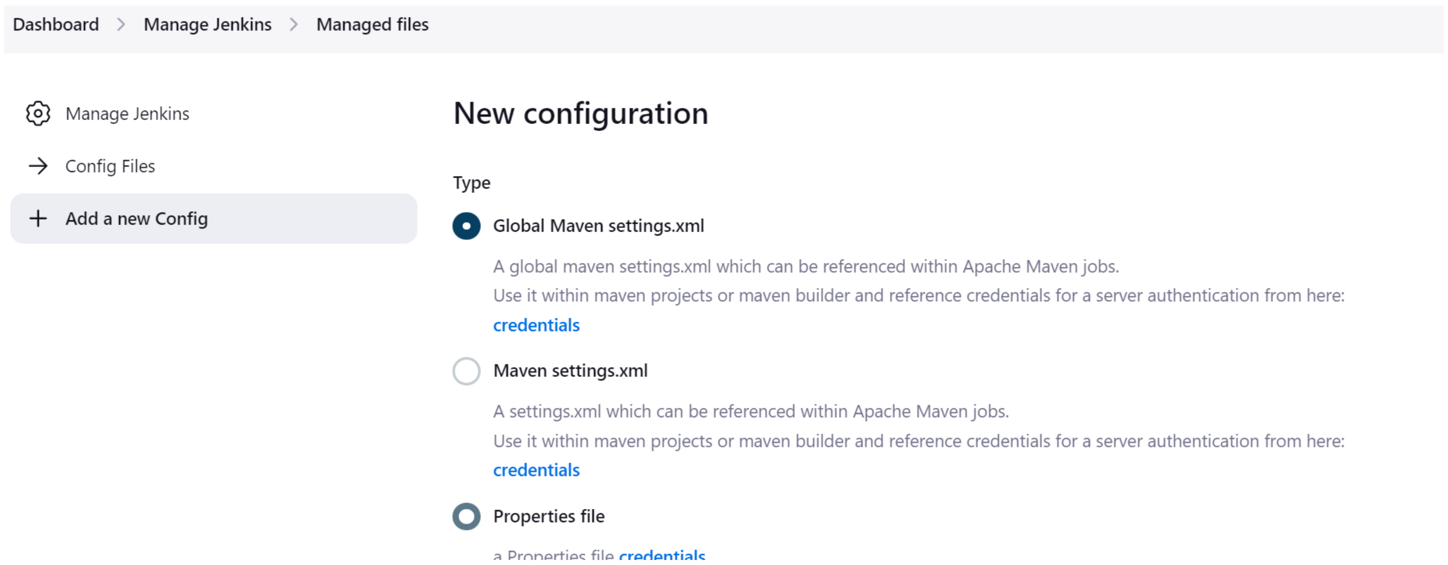
Use your repository's direct URL (shown below) to connect other tools to your repository. For more information, see our Maven-specific help documentation.

http://52.66.241.255:8081/repository/maven-snapshots/

do this changes in pom.xml



```
 <distributionManagement>
<repository>
    <id>maven-releases</id>
    <url>http://52.66.241.255:8081/repository/maven-releases/</url>
</repository>
<snapshotRepository>
    <id>maven-snapshots</id>
    <url>http://52.66.241.255:8081/repository/maven-snapshots/</ur
</snapshotRepository>
</distributionManagement>
```



Dashboard > Manage Jenkins > Managed files

Manage Jenkins
Config Files
Add a new Config

New configuration

Type

[✓ Global Maven settings.xml]

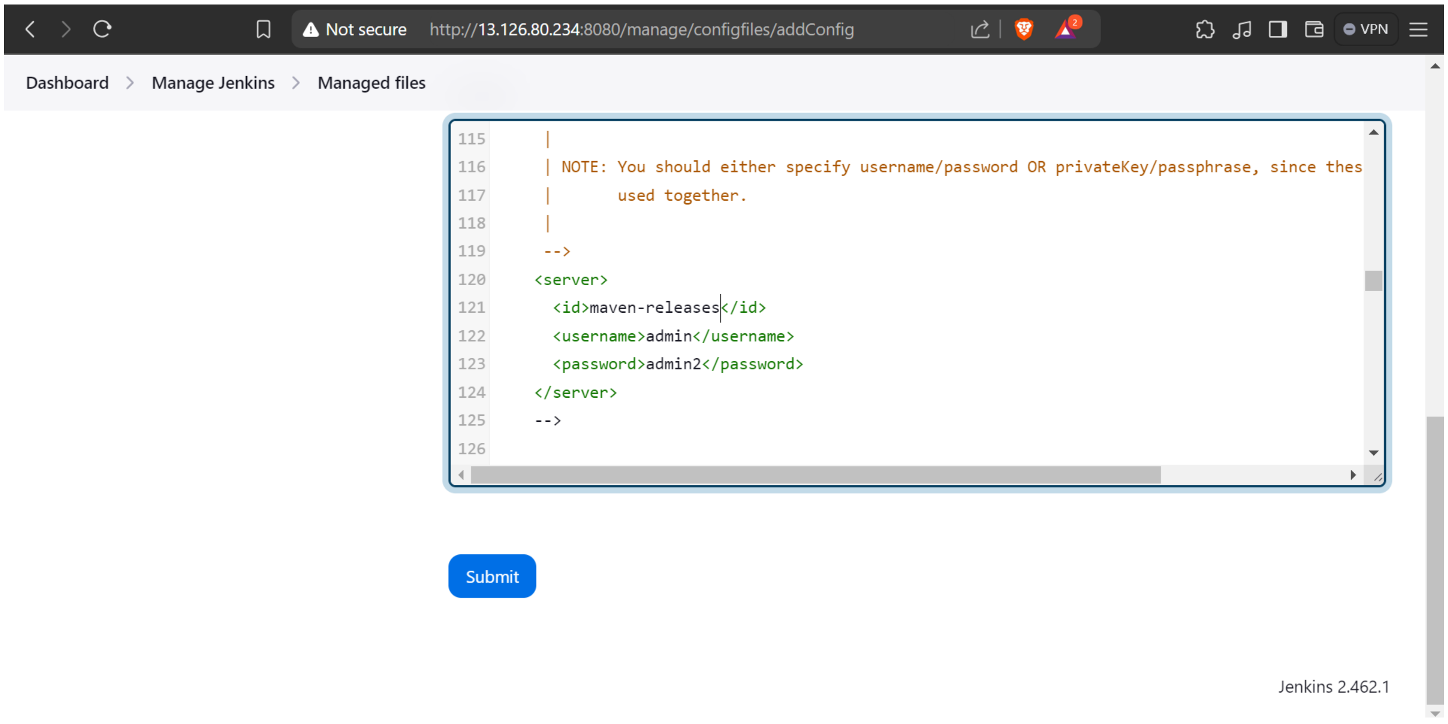
A global maven settings.xml which can be referenced within Apache Maven jobs.
Use it within maven projects or maven builder and reference credentials for a server authentication from here: credentials

Maven settings.xml

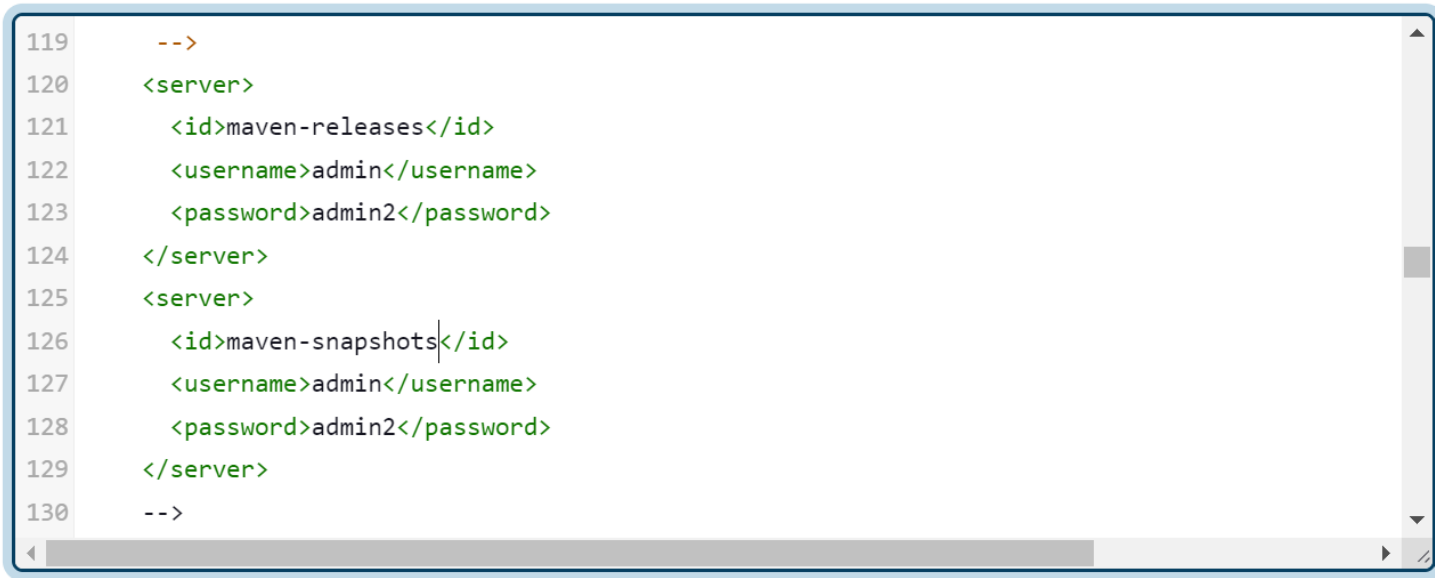
A settings.xml which can be referenced within Apache Maven jobs.
Use it within maven projects or maven builder and reference credentials for a server authentication from here: credentials

Properties file

a Properties file credentials


Not secure
http://13.126.80.234:8080/manage/configfiles/addConfig
Dashboard > Manage Jenkins > Managed files
Submit
Jenkins 2.462.1

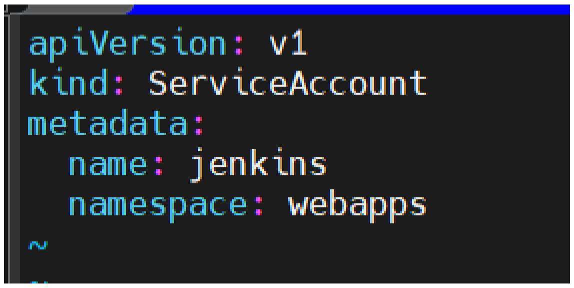
```
     |
     | NOTE: You should either specify username/password OR privateKey/passphrase, since thes
     |       used together.
     |
    -->
   <server>
     <id>maven-releases</id>
     <username>admin</username>
     <password>admin2</password>
   </server>
   -->
```



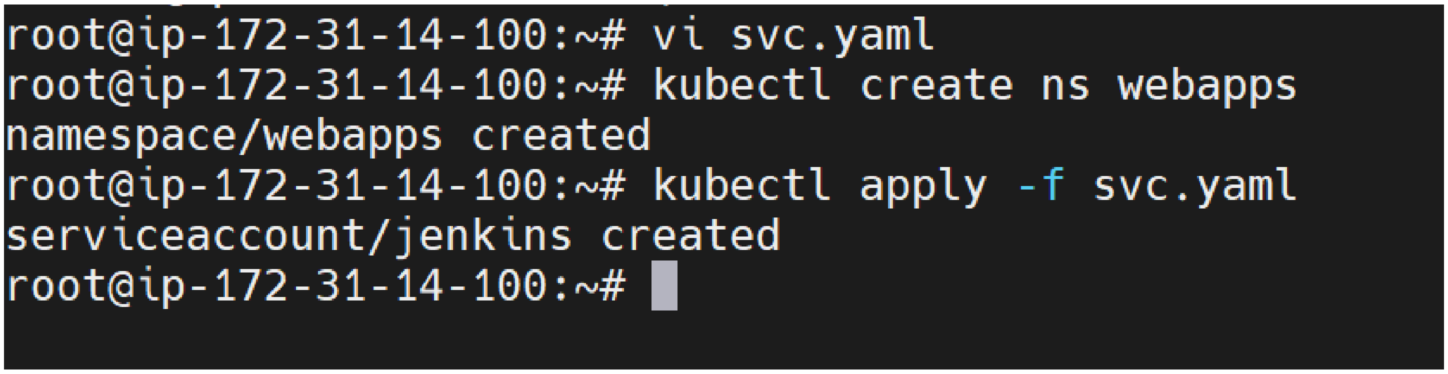
```
    -->
   <server>
     <id>maven-releases</id>
     <username>admin</username>
     <password>admin2</password>
   </server>
   <server>
     <id>maven-snapshots</id>
     <username>admin</username>
     <password>admin2</password>
   </server>
   -->
```

Now we want to deploy our service to the Kubernetes cluster

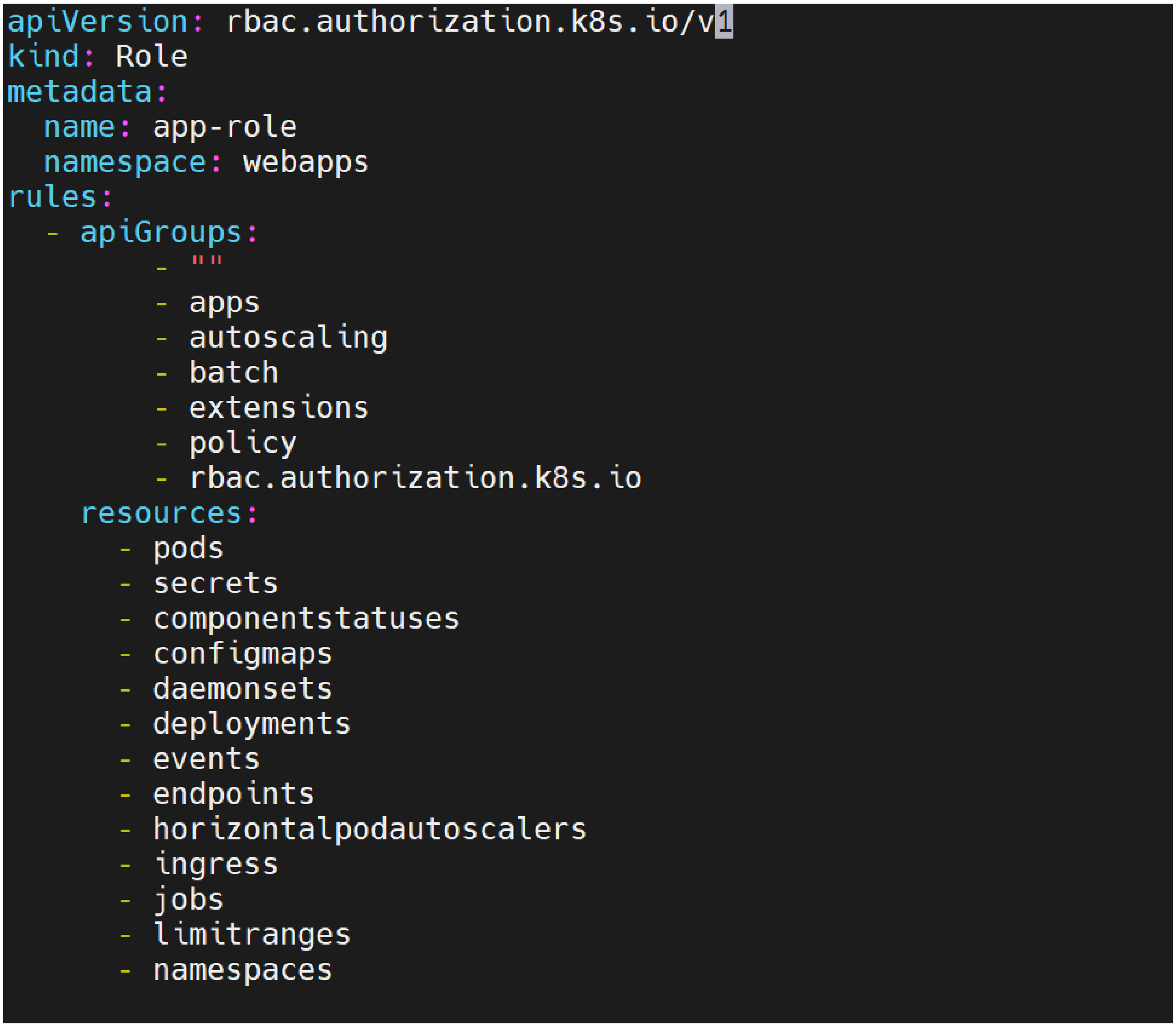
Creating service account

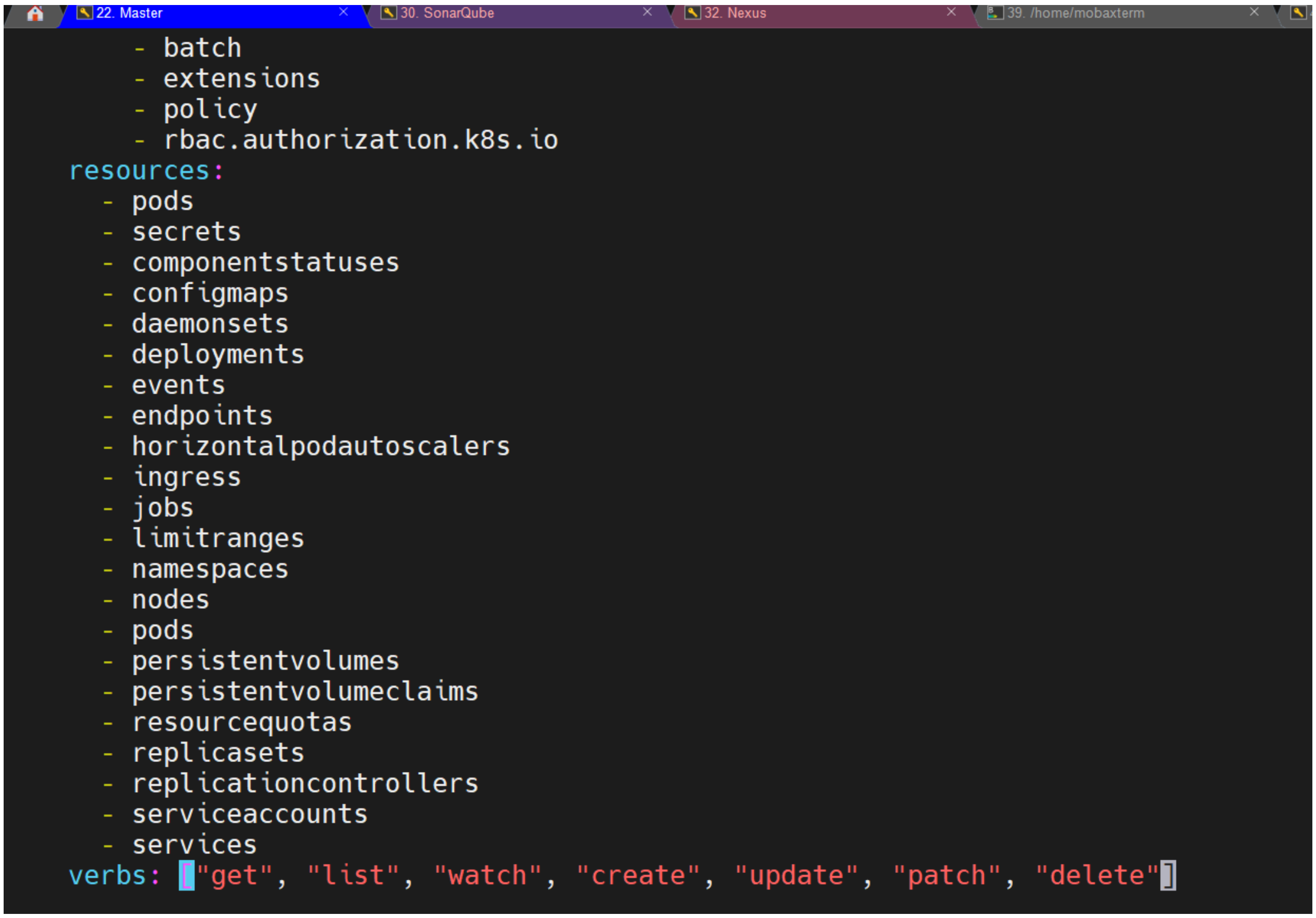


```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
  namespace: webapps
~
```

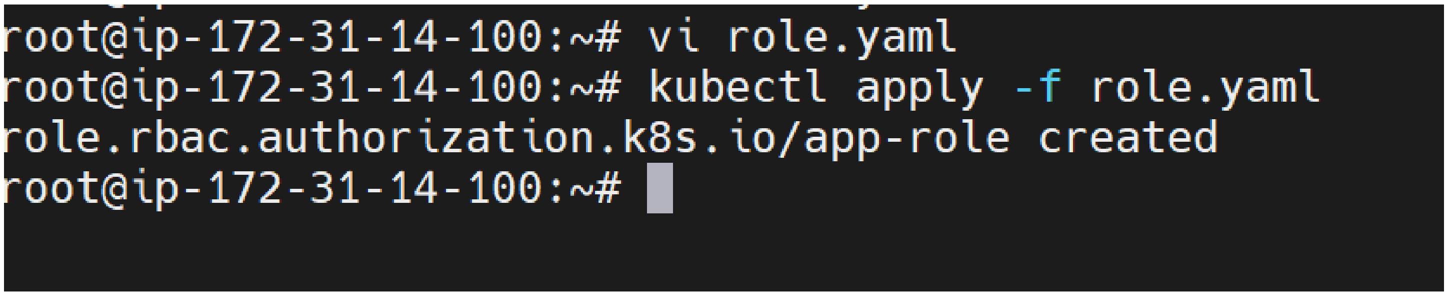

```
root@ip-172-31-14-100:~# vi svc.yaml
root@ip-172-31-14-100:~# kubectl create ns webapps
namespace/webapps created
root@ip-172-31-14-100:~# kubectl apply -f svc.yaml
serviceaccount/jenkins created
root@ip-172-31-14-100:~#
```

Roles


```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-role
  namespace: webapps
rules:
  - apiGroups:
        - ""
        - apps
        - autoscaling
        - batch
        - extensions
        - policy
        - rbac.authorization.k8s.io
    resources:
      - pods
      - secrets
      - componentstatuses
      - configmaps
      - daemonsets
      - deployments
      - events
      - endpoints
      - horizontalpodautoscalers
      - ingress
      - jobs
      - limitranges
      - namespaces
```

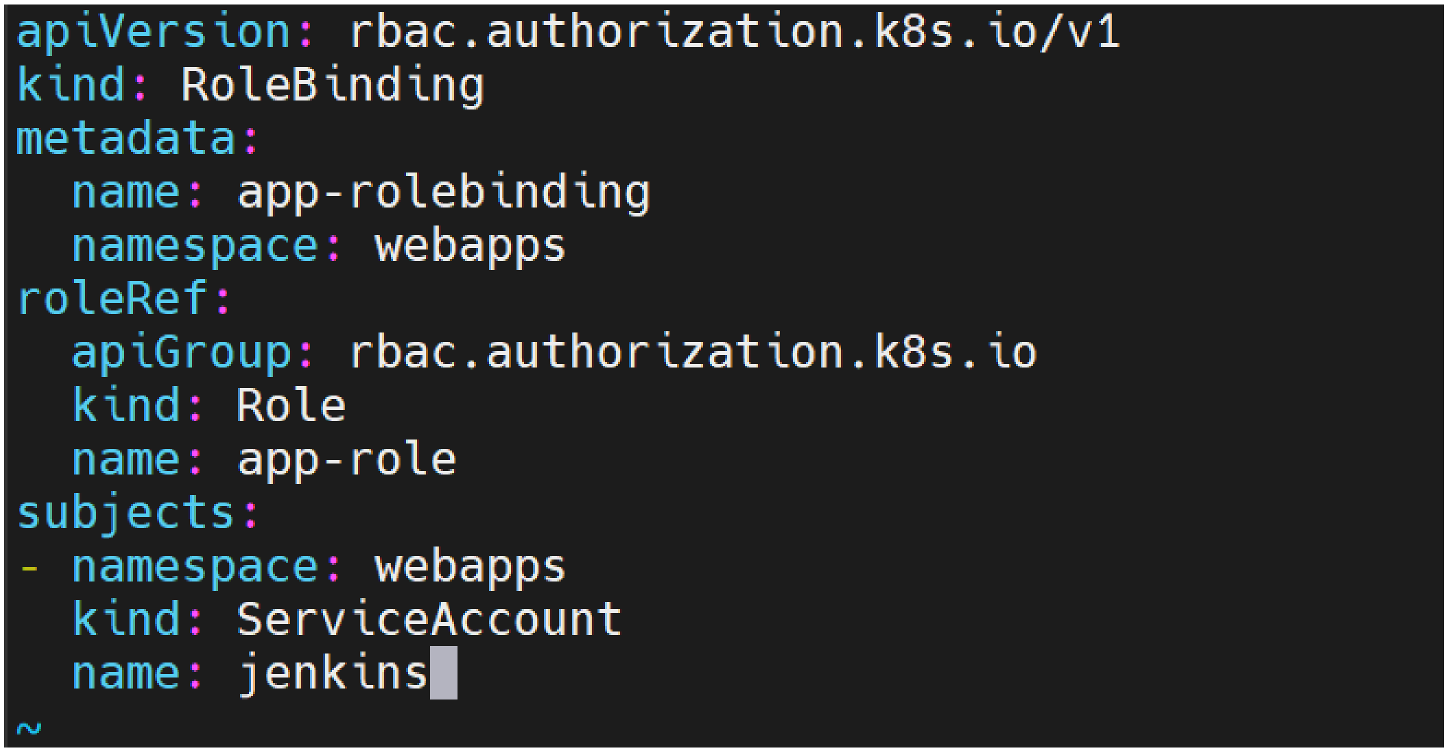


```
      - batch
      - extensions
      - policy
      - rbac.authorization.k8s.io
    resources:
    - pods
    - secrets
    - componentstatuses
    - configmaps
    - daemonsets
    - deployments
    - events
    - endpoints
    - horizontalpodautoscalers
    - ingress
    - jobs
    - limitranges
    - namespaces
    - nodes
    - pods
    - persistentvolumes
    - persistentvolumeclaims
    - resourcequotas
    - replicasets
    - replicationcontrollers
    - serviceaccounts
    - services
    verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```



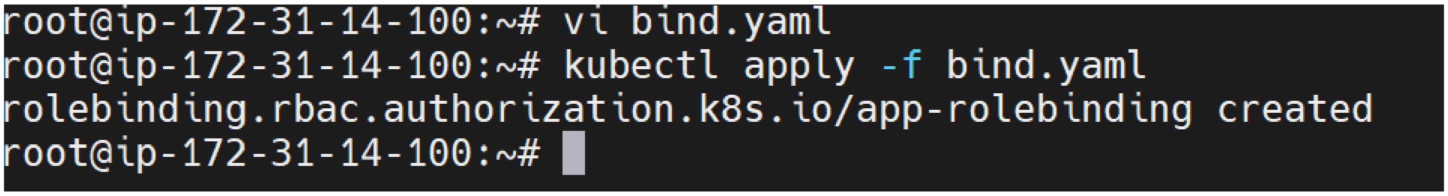
```
root@ip-172-31-14-100:~# vi role.yaml
root@ip-172-31-14-100:~# kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/app-role created
root@ip-172-31-14-100:~#
```

Now we have to bind the the role as below:



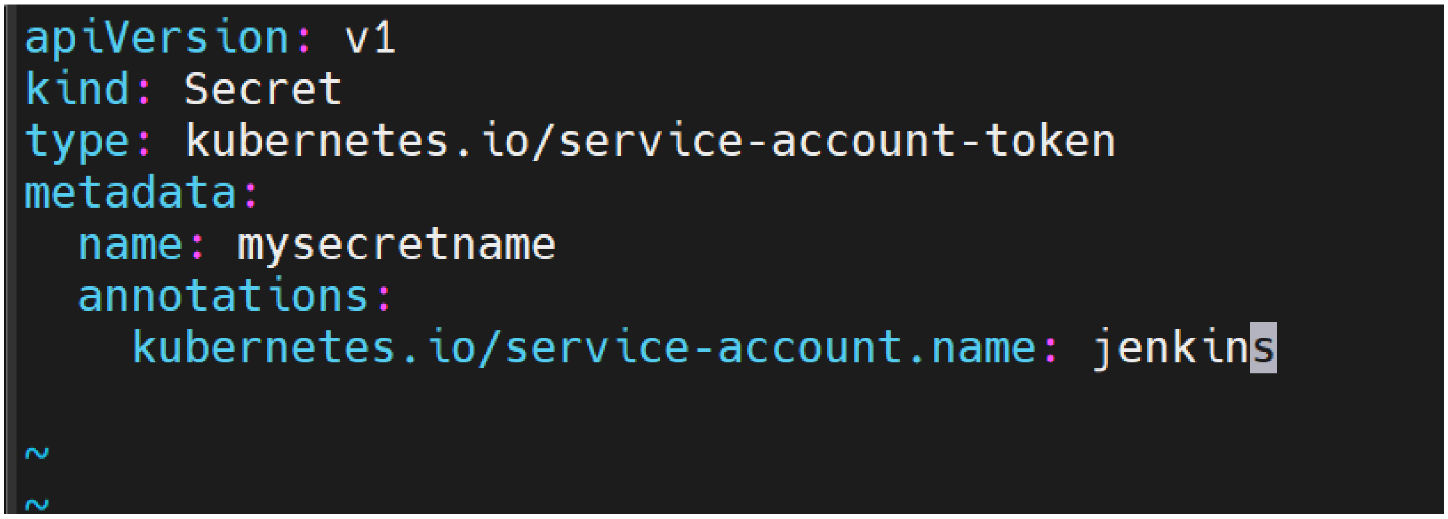
```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: app-rolebinding
  namespace: webapps
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: app-role
subjects:
- namespace: webapps
  kind: ServiceAccount
  name: jenkins
~
```

https://github.com/jaiswaladi246/EKS-Complete/blob/main/Steps-eks.md (for refference)

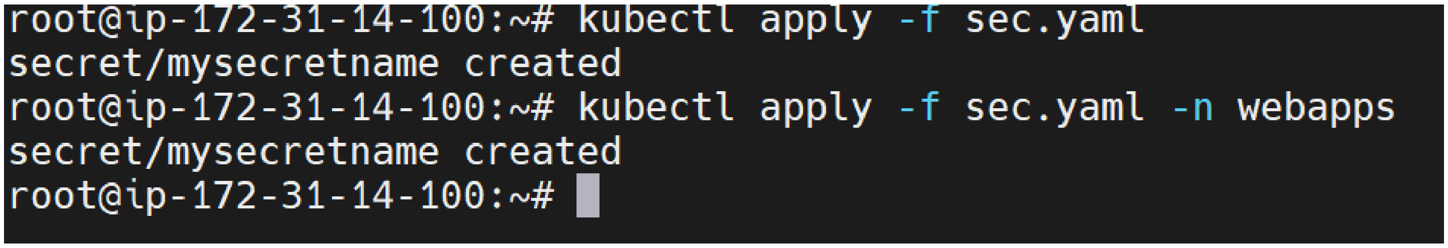


```
root@ip-172-31-14-100:~# vi bind.yaml
root@ip-172-31-14-100:~# kubectl apply -f bind.yaml
rolebinding.rbac.authorization.k8s.io/app-rolebinding created
root@ip-172-31-14-100:~#
```

Access token for authentification to connect with jenkins

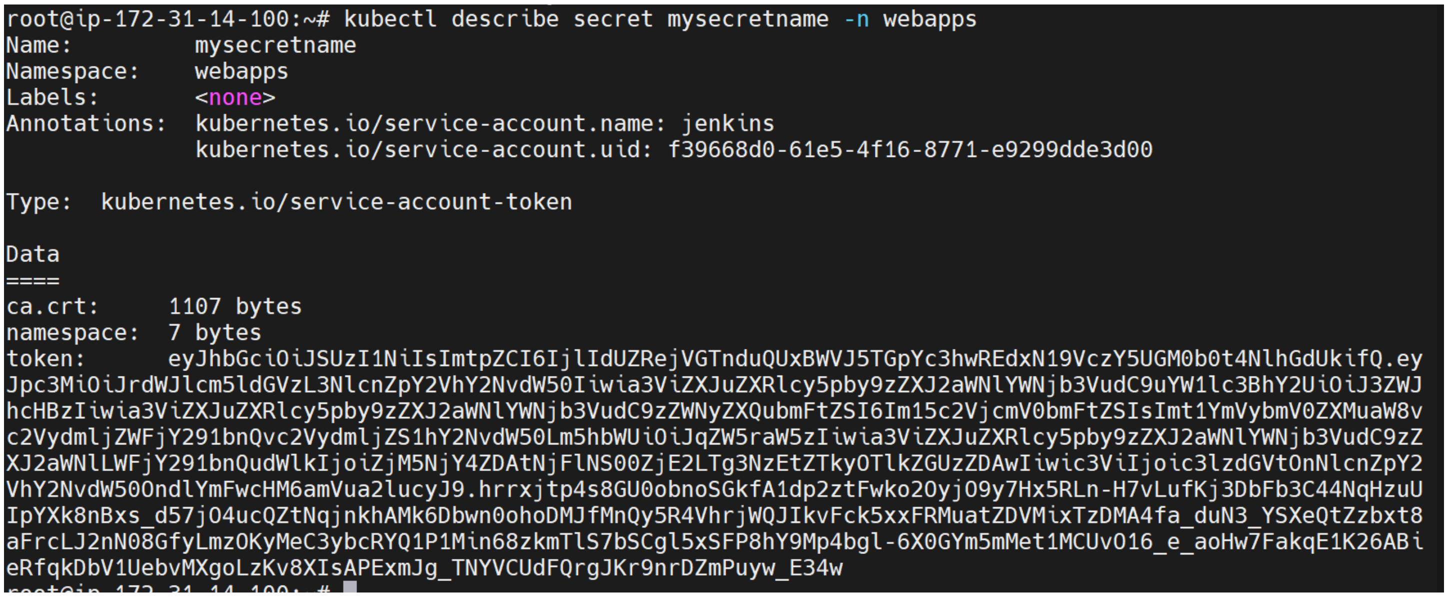


```
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: mysecretname
  annotations:
    kubernetes.io/service-account.name: jenkins

~
~
```



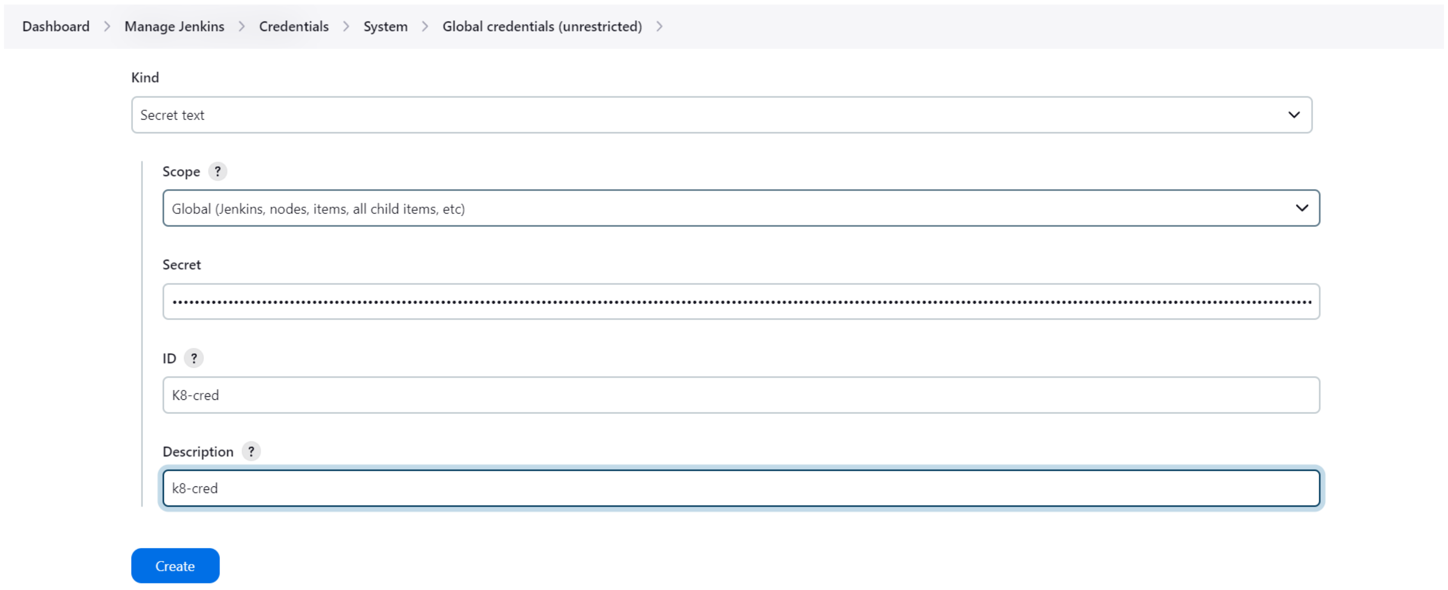
```
root@ip-172-31-14-100:~# kubectl apply -f sec.yaml
secret/mysecretname created
root@ip-172-31-14-100:~# kubectl apply -f sec.yaml -n webapps
secret/mysecretname created
root@ip-172-31-14-100:~#
```

For token



```
root@ip-172-31-14-100:~# kubectl describe secret mysecretname -n webapps
Name:         mysecretname
Namespace:    webapps
Labels:       <none>
Annotations:  kubernetes.io/service-account.name: jenkins
              kubernetes.io/service-account.uid: f39668d0-61e5-4f16-8771-e9299dde3d00

Type:  kubernetes.io/service-account-token

Data
====
ca.crt:     1107 bytes
namespace:  7 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IjlIdUZRejVGTnduQUxBWVJ5TGpYc3hwREdxN19VczY5UGM0b0t4NlhGdUkifQ.ey
Jpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJ3ZWJ
hcHBzIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZWNyZXQubmFtZSI6Im15c2VjcmV0bmFtZSIsImt1YmVybmV0ZXMuaW8v
c2VydmljZWFjY291bnQvc2VydmljZS1hY2NvdW50Lm5hbWUiOiJqZW5raW5zIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZ
XJ2aWNlLWFjY291bnQudWlkIjoiZjM5NjY4ZDAtNjFlNS00ZjE2LTg3NzEtZTkyOTlkZGUzZDAwIiwic3ViIjoic3lzdGVtOnNlcnZpY2
VhY2NvdW50OndlYmFwcHM6amVua2lucyJ9.hrrxjtp4s8GU0obnoSGkfA1dp2ztFwko2OyjO9y7Hx5RLn-H7vLufKj3DbFb3C44NqHzuU
IpYXk8nBxs_d57jO4ucQZtNqjnkhAMk6Dbwn0ohoDMJfMnQy5R4VhrjWQJIkvFck5xxFRMuatZDVMixTzDMA4fa_duN3_YSXeQtZzbxt8
aFrcLJ2nN08GfyLmzOKyMeC3ybcRYQ1P1Min68zkmTlS7bSCgl5xSFP8hY9Mp4bgl-6X0GYm5mMet1MCUvO16_e_aoHw7FakqE1K26ABi
eRfqkDbV1UebvMXgoLzKv8XIsAPExmJg_TNYVCUdFQrgJKr9nrDZmPuyw_E34w
```

Then go to Jenkins



Dashboard > Manage Jenkins > Credentials > System > Global credentials (unrestricted) >

Kind: Secret text

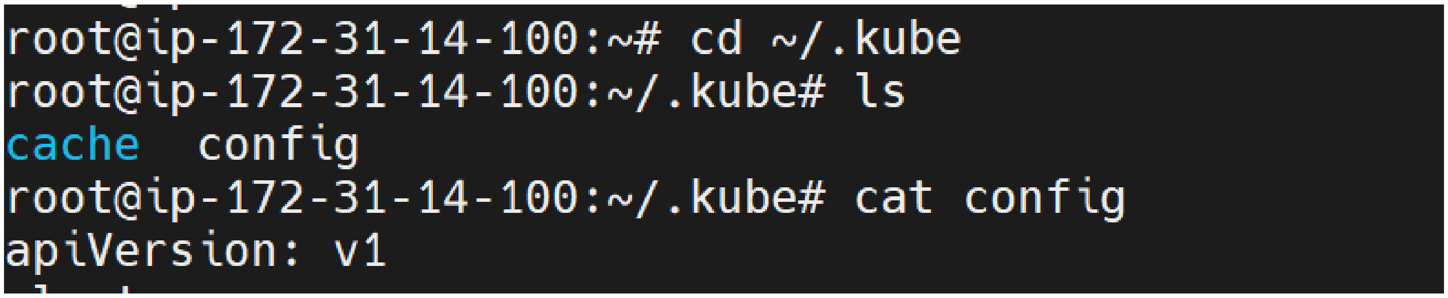
Scope: Global (Jenkins, nodes, items, all child items, etc)

Secret: ••

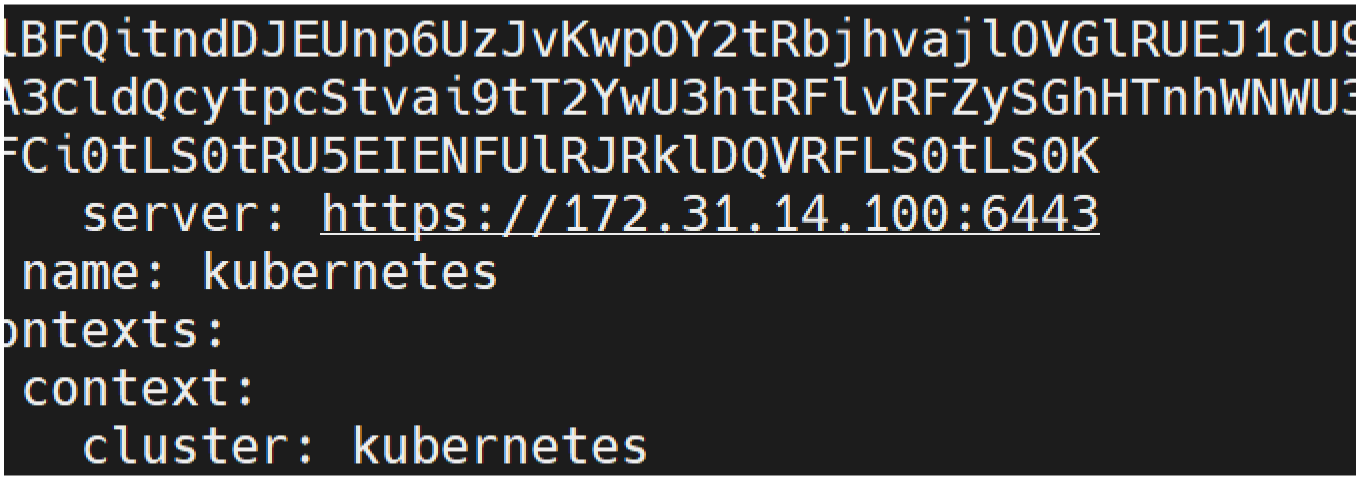
ID: K8-cred

Description: k8-cred

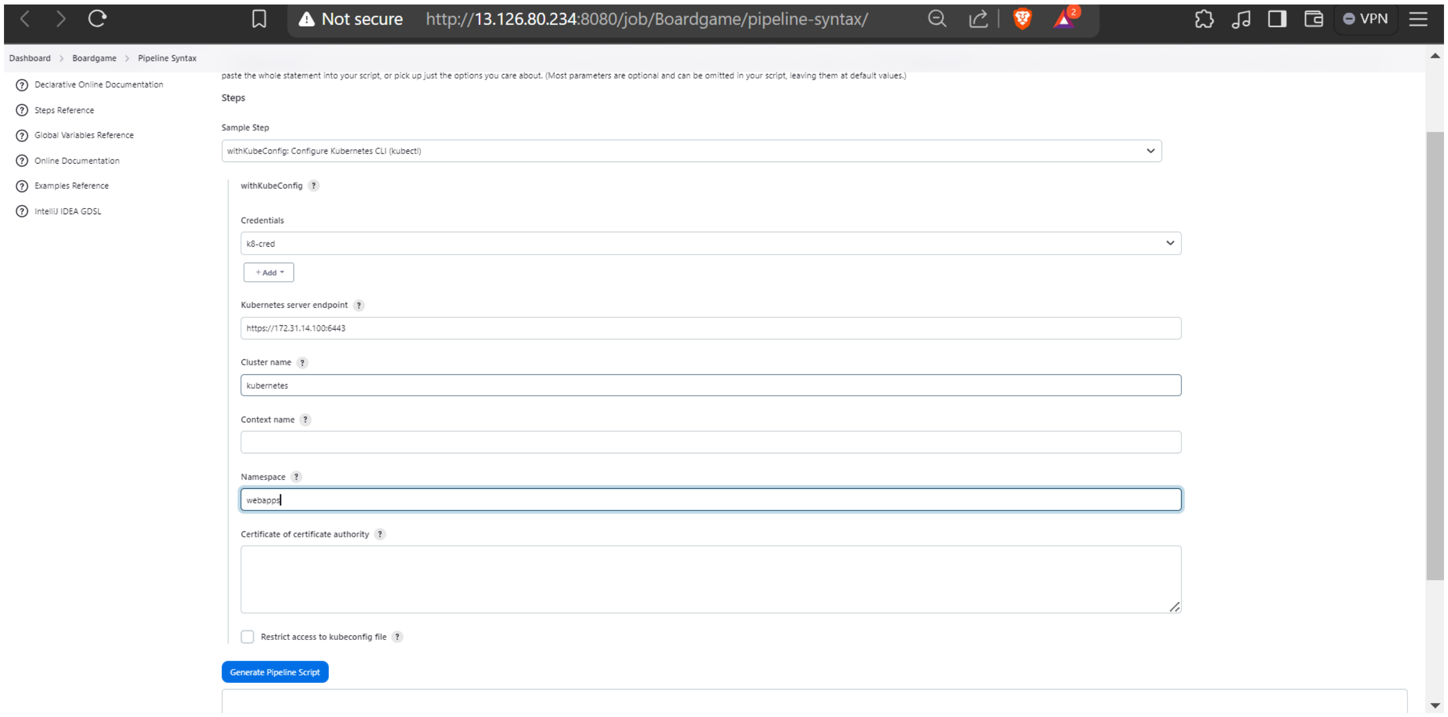
To find server endpoint



```
root@ip-172-31-14-100:~# cd ~/.kube
root@ip-172-31-14-100:~/.kube# ls
cache  config
root@ip-172-31-14-100:~/.kube# cat config
apiVersion: v1
```



```
lBFQitndDJEUnp6UzJvKwpOY2tRbjhvajlOVGlRUEJ1cU9
A3CldQcytpcStvai9tT2YwU3htRFlvRFZySGhHTnhWNWU3
FCi0tLS0tRU5EIENFUlRJRklDQVRFLS0tLS0K
    server: https://172.31.14.100:6443
 name: kubernetes
ontexts:
 context:
    cluster: kubernetes
```



Not secure http://13.126.80.234:8080/job/Boardgame/pipeline-syntax/

Dashboard > Boardgame > Pipeline Syntax

- Declarative Online Documentation
- Steps Reference
- Global Variables Reference
- Online Documentation
- Examples Reference
- IntelliJ IDEA GDSL

paste the whole statement into your script, or pick up just the options you care about. (Most parameters are optional and can be omitted in your script, leaving them at default values.)

Steps

Sample Step: withKubeConfig: Configure Kubernetes CLI (kubectl)

withKubeConfig

Credentials: k8-cred

Kubernetes server endpoint: https://172.31.14.100:6443

Cluster name: kubernetes

Context name:

Namespace: webapps

Certificate of certificate authority:

Restrict access to kubeconfig file

Generate Pipeline Script

pipeline successfully completed

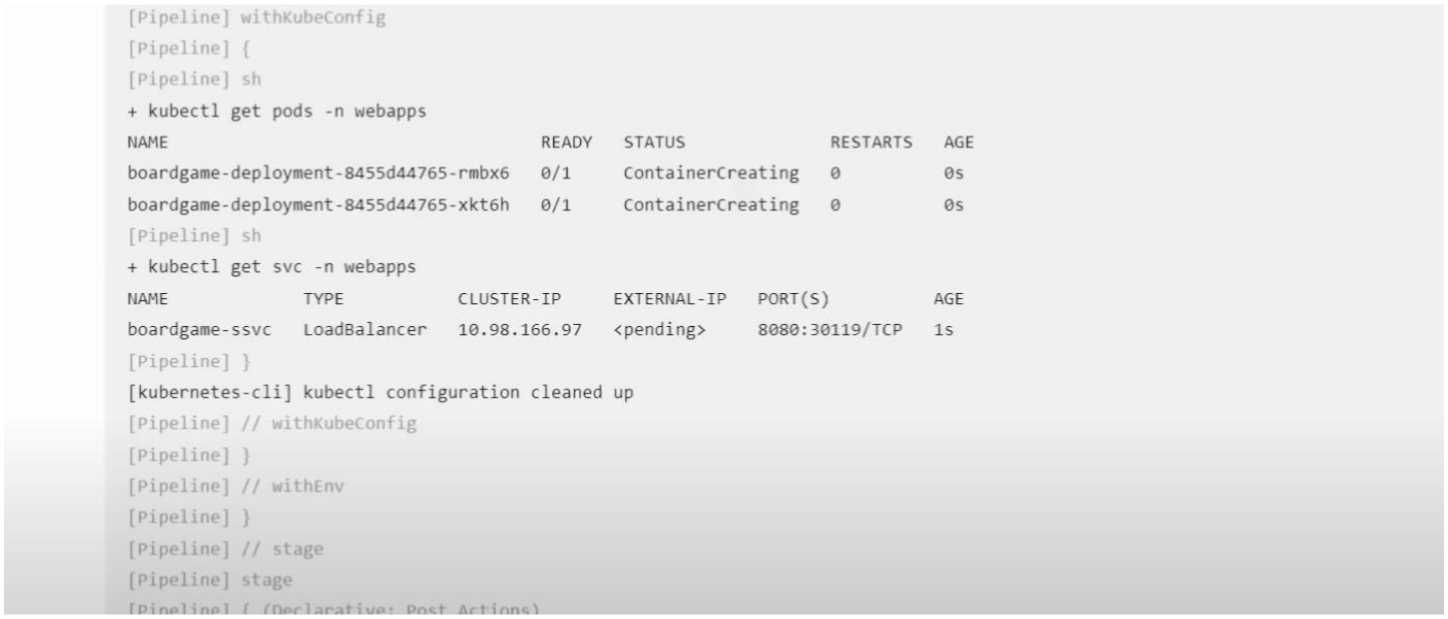
pipeline {

```
agent any


tools {
   jdk 'jdk17'
   maven 'maven3'
}


environment {
   SCANNER_HOME= tool 'sonar-scanner'
}


stages {
   stage('Git Checkout') {
      steps {
        git branch: 'main', credentialsId: 'git-cred', url: 'https://github.com/pranjalpagaria/boardgame.git'
      }
   }


   stage('Compile') {
      steps {
         sh "mvn compile"
      }
   }


   stage('Test') {
      steps {
         sh "mvn test"
      }
```

```
        }


        stage('File System Scan') {
            steps {
                sh "trivy fs --format table -o trivy-fs-report.html ."
            }
        }


        stage('SonarQube Analsyis') {
            steps {
                withSonarQubeEnv('sonar') {
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=BoardGame -
Dsonar.projectKey=BoardGame \
                            -Dsonar.java.binaries=. '''
                }
            }
        }


        stage('Quality Gate') {
            steps {
                script {
                  waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
                }
            }
        }


        stage('Build') {
            steps {
               sh "mvn package"
```

```
          }

        }


        stage('Publish To Nexus') {

          steps {

             withMaven(globalMavenSettingsConfig: 'global-settings', jdk: 'jdk17', maven: 'maven3',
mavenSettingsConfig: '', traceability: true) {

                  sh "mvn deploy"

              }

          }

        }


        stage('Build & Tag Docker Image') {

          steps {

             script {

                 withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {

                          sh "docker build -t pranjalpagaria/boardshack:latest ."

                  }

             }

          }

        }


        stage('Docker Image Scan') {

          steps {

              sh "trivy image --format table -o trivy-image-report.html pranjalpagaria/boardshack:latest "

          }

        }


        stage('Push Docker Image') {
          steps {
             script {
                 withDockerRegistry(credentialsId: 'docker-cred', toolName: 'docker') {
                          sh "docker push pranjalpagaria/boardshack:latest"
                  }
             }
          }
        }
        stage('Deploy To Kubernetes') {
          steps {
             withKubeConfig(caCertificate: '', clusterName: 'kubernetes', contextName: '', credentialsId: 'k8-cred',
namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'https://172.31.14.100:6443') {
                        sh "kubectl apply -f deployment-service.yaml"
              }
          }
        }


        stage('Verify the Deployment') {
          steps {
             withKubeConfig(caCertificate: '', clusterName: 'kubernetes', contextName: '', credentialsId: 'k8-cred',
namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'https://172.31.14.100:6443') {
                        sh "kubectl get pods -n webapps"
                        sh "kubectl get svc -n webapps"
              }
          }
        }



    }
```

```
post {
always {
   script {
      def jobName = env.JOB_NAME
      def buildNumber = env.BUILD_NUMBER
      def pipelineStatus = currentBuild.result ?: 'UNKNOWN'
      def bannerColor = pipelineStatus.toUpperCase() == 'SUCCESS' ? 'green' : 'red'

      def body = """
         <html>
         <body>
         <div style="border: 4px solid ${bannerColor}; padding: 10px;">
         <h2>${jobName} - Build ${buildNumber}</h2>
         <div style="background-color: ${bannerColor}; padding: 10px;">
         <h3 style="color: white;">Pipeline Status: ${pipelineStatus.toUpperCase()}</h3>
         </div>
         <p>Check the <a href="${BUILD_URL}">console output</a>.</p>
         </div>
         </body>
         </html>
      """

      emailext (
         subject: "${jobName} - Build ${buildNumber} - ${pipelineStatus.toUpperCase()}",
         body: body,
         to: 'pranjalpagaria20@gmai.com',
         from: 'jenkins@example.com',
         replyTo: 'jenkins@example.com',
                mimeType: 'text/html',

                attachmentsPattern: 'trivy-image-report.html'

            )

        }

    }

}


}
```

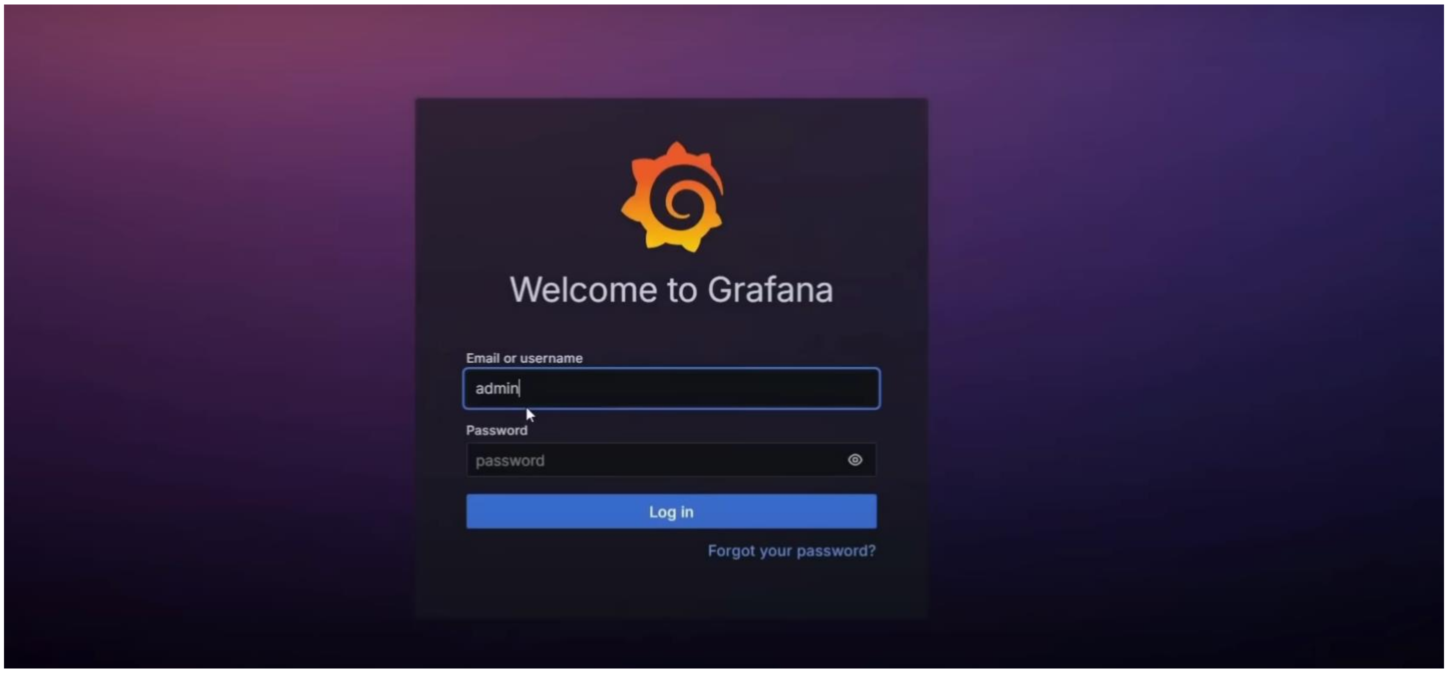


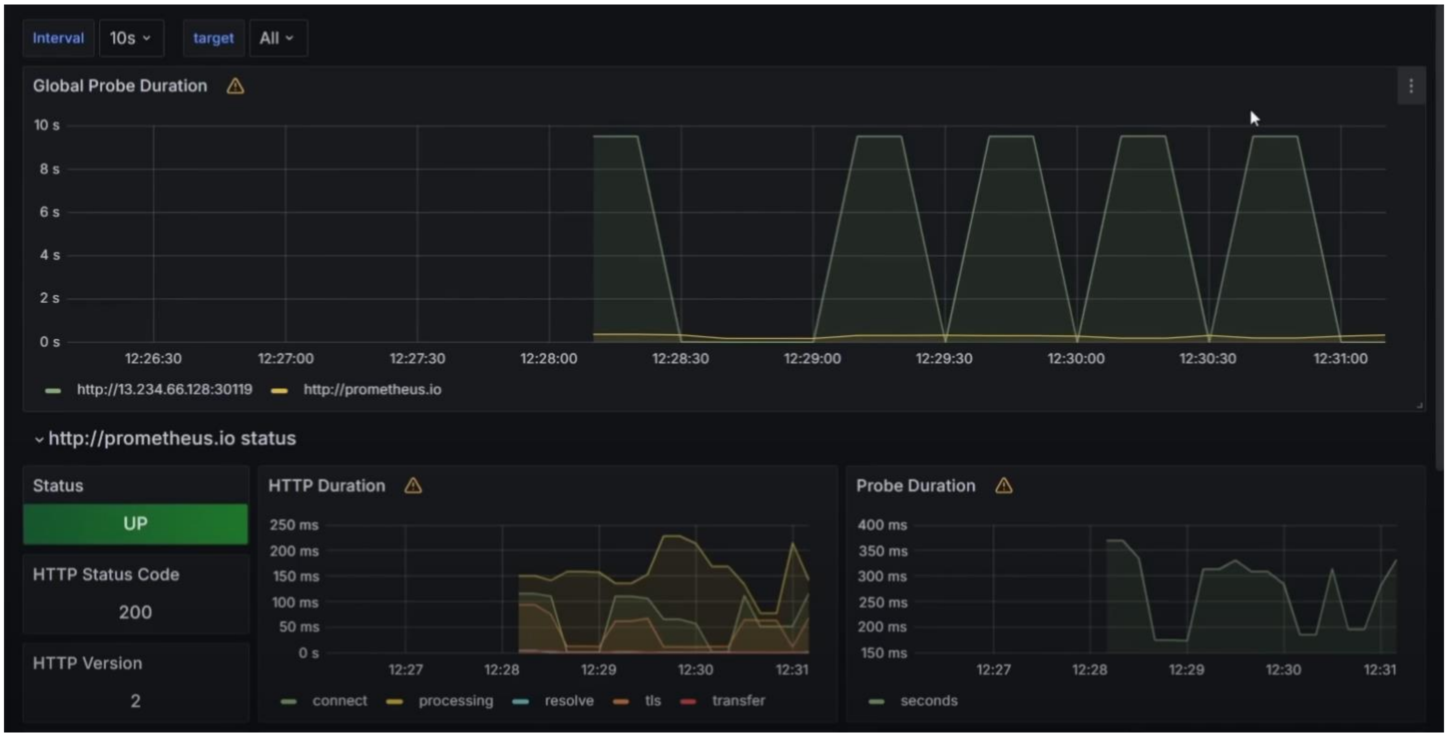
```
[Pipeline] withKubeConfig
[Pipeline] {
[Pipeline] sh
+ kubectl get pods -n webapps
NAME                                  READY   STATUS              RESTARTS   AGE
boardgame-deployment-8455d44765-rmbx6   0/1     ContainerCreating   0          0s
boardgame-deployment-8455d44765-xkt6h   0/1     ContainerCreating   0          0s
[Pipeline] sh
+ kubectl get svc -n webapps
NAME             TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
boardgame-ssvc   LoadBalancer   10.98.166.97   <pending>     8080:30119/TCP   1s
[Pipeline] }
[kubernetes-cli] kubectl configuration cleaned up
[Pipeline] // withKubeConfig
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
```

Phase -4

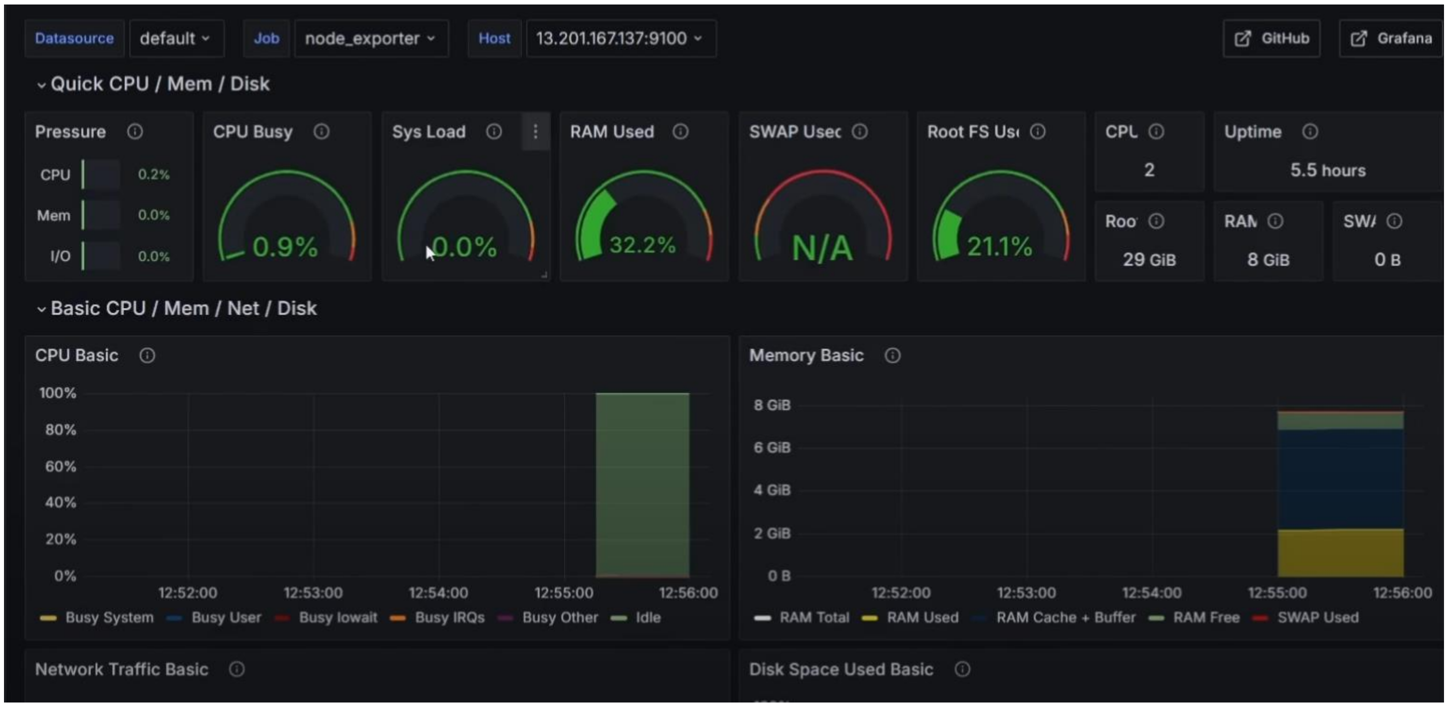
Links to download Prometheus, Node_Exporter & black Box exporter https://prometheus.io/download/

Links to download Grafana https://grafana.com/grafana/download





To monitor Jenkins intall node expoter in jenkins
It is used to monitor the system metrics.


Datasource
default
Job
node_exporter
Host
13.201.167.137:9100
GitHub
Grafana
Quick CPU / Mem / Disk
Pressure
CPU 0.2%
Mem 0.0%
I/O 0.0%
CPU Busy 0.9%
Sys Load 0.0%
RAM Used 32.2%
SWAP Usec N/A
Root FS Us 21.1%
CPL 2
Uptime 5.5 hours
Roo 29 GiB
RAM 8 GiB
SW/ 0 B
Basic CPU / Mem / Net / Disk
CPU Basic
Busy System
Busy User
Busy Iowait
Busy IRQs
Busy Other
Idle
Memory Basic
RAM Total
RAM Used
RAM Cache + Buffer
RAM Free
SWAP Used
Network Traffic Basic
Disk Space Used Basic