End-to-End CI/CD Pipeline with Jenkins, ArgoCD, Kubernetes, Security Integration, Automated Deployments and Monitoring

Introduction

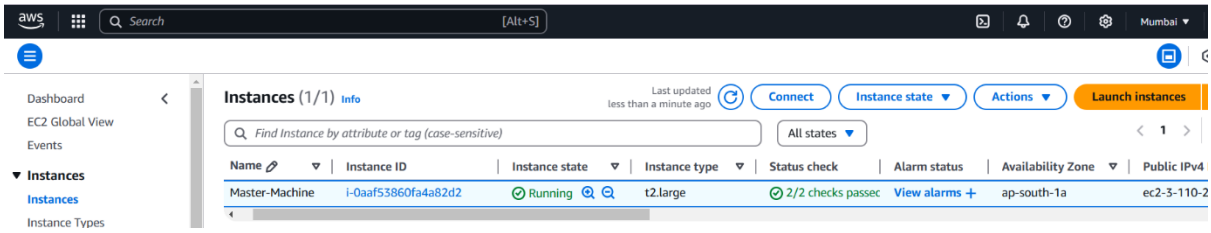
The project is a complete DevOps pipeline designed to automate and optimize the software development lifecycle, from code integration to deployment and monitoring. It incorporates cutting-edge tools and technologies to implement robust Continuous Integration (CI) and Continuous Deployment (CD) processes. GitHub serves as the source code repository, while Jenkins orchestrates the CI/CD workflows. Code quality and security are ensured using SonarQube and Trivy, respectively, while Docker manages containerization. ArgoCD handles deployment to Kubernetes, enabling efficient application delivery to production environments. Real-time system monitoring is achieved using Prometheus and Grafana, with email notifications providing timely updates on pipeline statuses. This pipeline ensures secure, scalable, and efficient application deployment with a focus on automation and reliability.

Prerequisites:

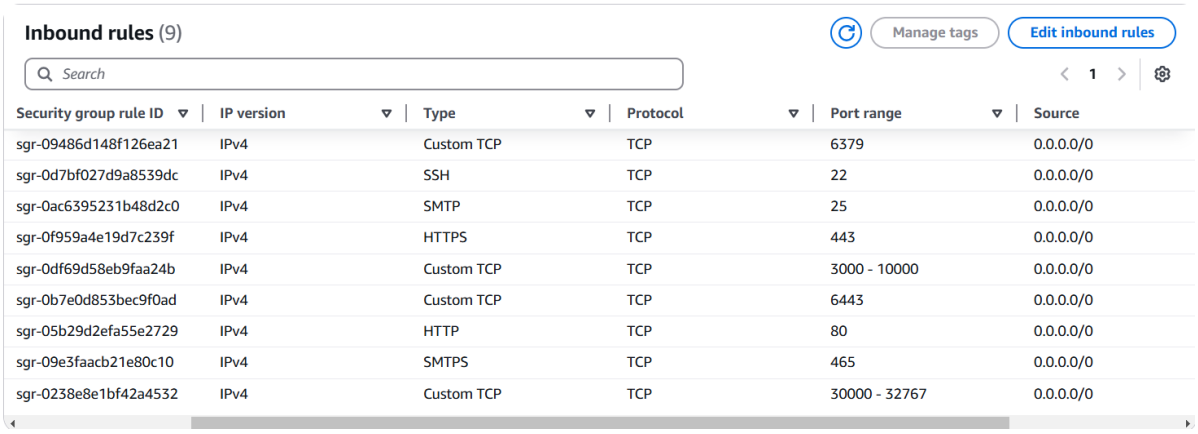
1. Dockerhub Account
2. Github Account
3. AWS Account

Step1 : Create a master machine using terraform script

- Create 1 Master machine on AWS with 2CPU, 8GB of RAM (t2.large) and 30 GB of storage and install Docker on it.



- Open the below ports in security group of master machine and also attach same security group to Jenkins worker node



- Create one IAM Role for Ec2 instance provide admin access and attach the role to this master machine

- Install & Configure Docker by using below command, "NewGrp docker" will refresh the group config hence no need to restart the EC2 machine

  sudo apt-get update

  sudo apt-get install docker.io -y

  sudo usermod -aG docker ubuntu && newgrp docker

  sudo chmod 777 /var/run/docker.sock

```
ubuntu@ip-172-31-46-111:~$ docker -v
Docker version 26.1.3, build 26.1.3-0ubuntu1~24.04.1
ubuntu@ip-172-31-46-111:~$
```

- **Install and configure Jenkins (Master machine)**

  sudo apt update -y

  sudo apt install fontconfig openjdk-17-jre -y

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

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

  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

  sudo apt-get update -y

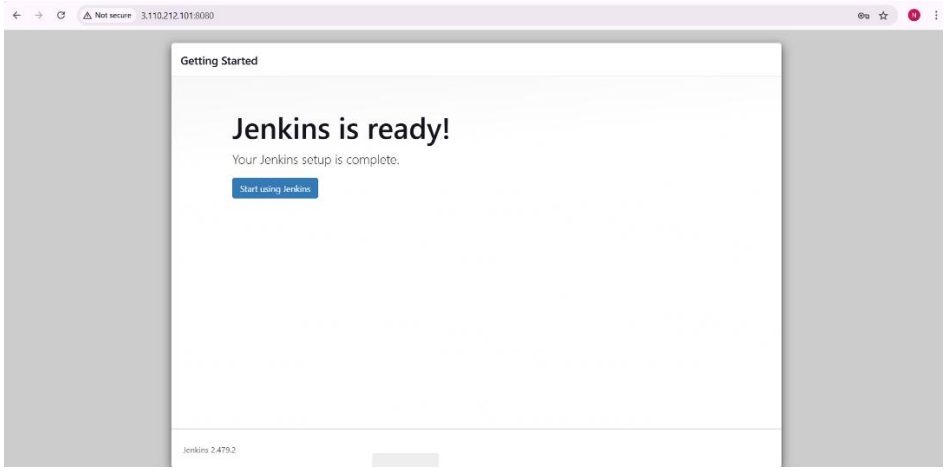
  sudo apt-get install jenkins -y

  sudo usermod -aG docker Jenkins

  sudo systemctl restart jenkins

Now, access Jenkins Master on the browser on port 8080 and configure it

http://3.110.212.101:8080/



Step 2 : Create EKS Cluster on AWS (Master machine)

- IAM user with **access keys and secret access keys**
- AWSCLI should be configured(master machine)

  curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

  sudo apt install unzip

  unzip awscliv2.zip

  sudo ./aws/install

  aws configure

```
ubuntu@ip-172-31-46-111:~$ aws configure
AWS Access Key ID [****************LHEH]:
AWS Secret Access Key [****************q0OI]:
Default region name [None]:
Default output format [None]:
```

- **Install kubectl (Master machine**)

curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin

kubectl version --short --client

- **Install eksctl (Master machine)**

  curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp

  sudo mv /tmp/eksctl /usr/local/bin

eksctl version

```
ubuntu@ip-172-31-46-111:~$ curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.19.6/2021-01-05/bin/linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin
kubectl version --short --client
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 57.4M  100 57.4M    0     0  2434k      0  0:00:24  0:00:24 --:--:-- 3304k
Client Version: v1.19.6-eks-49a6c0
ubuntu@ip-172-31-46-111:~$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz
 -C /tmp
sudo mv /tmp/eksctl /usr/local/bin
eksctl version
0.199.0
ubuntu@ip-172-31-46-111:~$
```

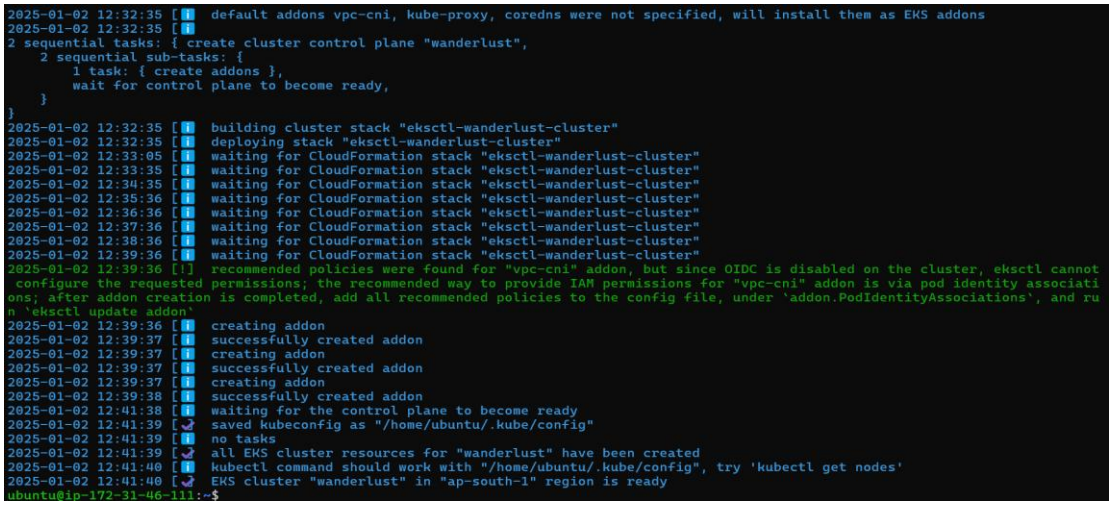
- **Create EKS Cluster (Master machine)**

  eksctl create cluster --name=wanderlust \

  --region=ap-south-1 \

  --version=1.30 \

  --without-nodegroup



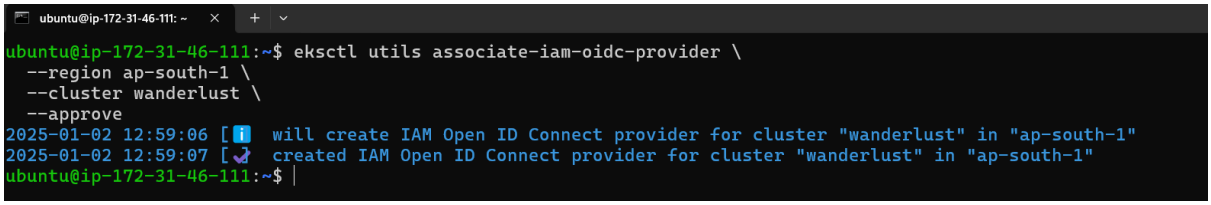
- **Associate IAM OIDC Provider (Master machine)**

  eksctl utils associate-iam-oidc-provider \

  --region ap-south-1 \

  --cluster wanderlust \

  --approve



- **Create Nodegroup (Master machine)**

  eksctl create nodegroup --cluster=wanderlust \

  --region=ap-south-1 \

  --name=wanderlust \

  --node-type=t2.large \

  --nodes=2 \

  --nodes-min=2 \

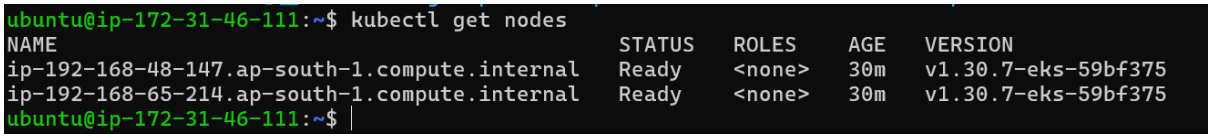
  --nodes-max=2 \

  --node-volume-size=30 \

  --ssh-access \

  --ssh-public-key=eks-nodegroup-key

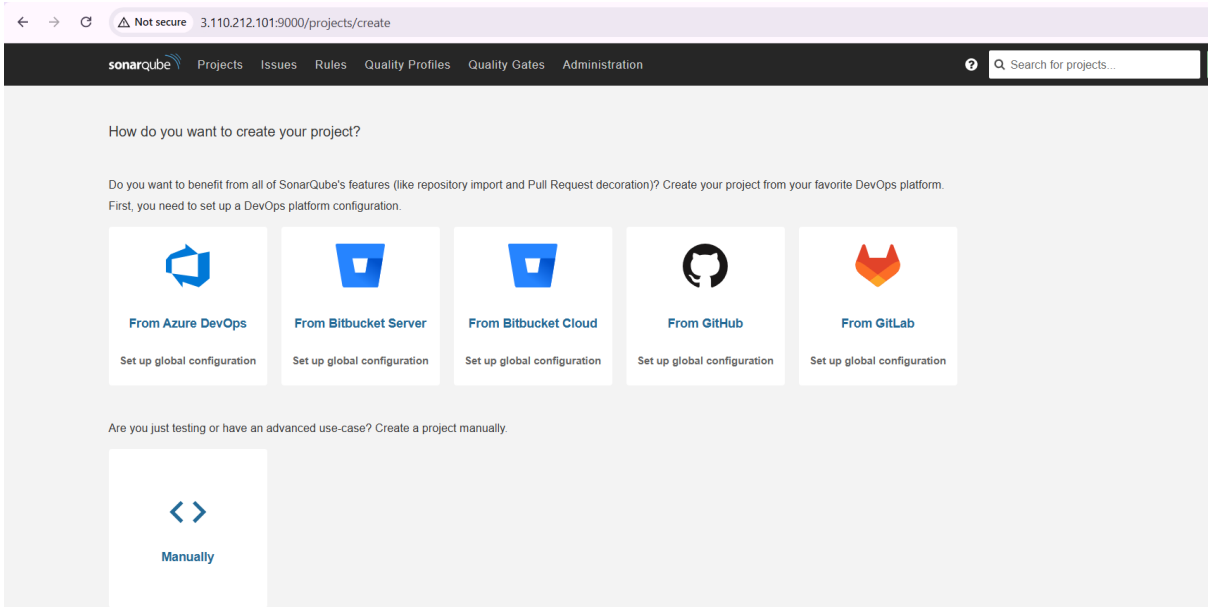
Make sure the ssh-public-key "eks-nodegroup-key" is available in your AWS account



Step 3: Install SonarQube, Trivy & OWASP on master machine

- **Install and configure SonarQube (Master machine)**

  docker run -itd --name SonarQube-Server -p 9000:9000 sonarqube:lts-community



- **Install Trivy**

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

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

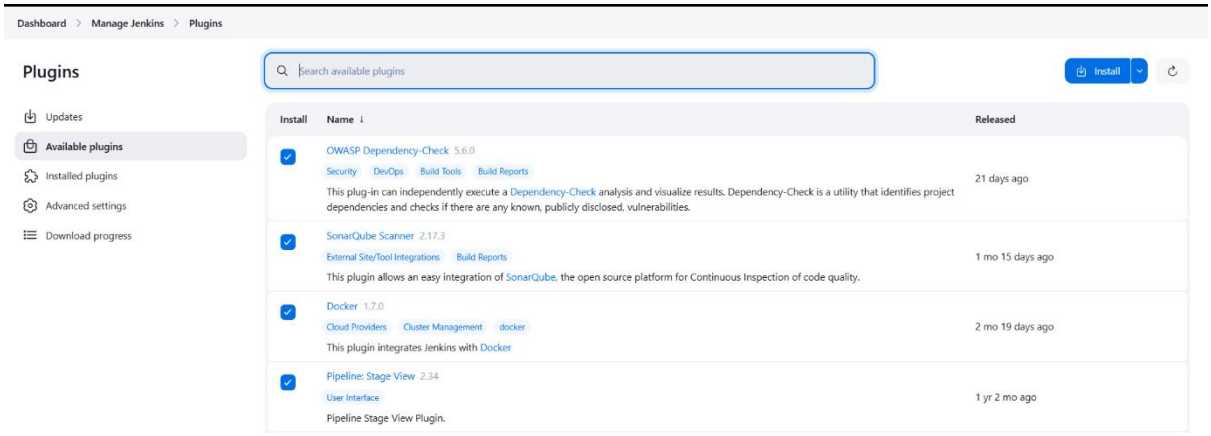
  echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main | sudo tee -a /etc/apt/sources.list.d/trivy.list

  sudo apt-get update -y

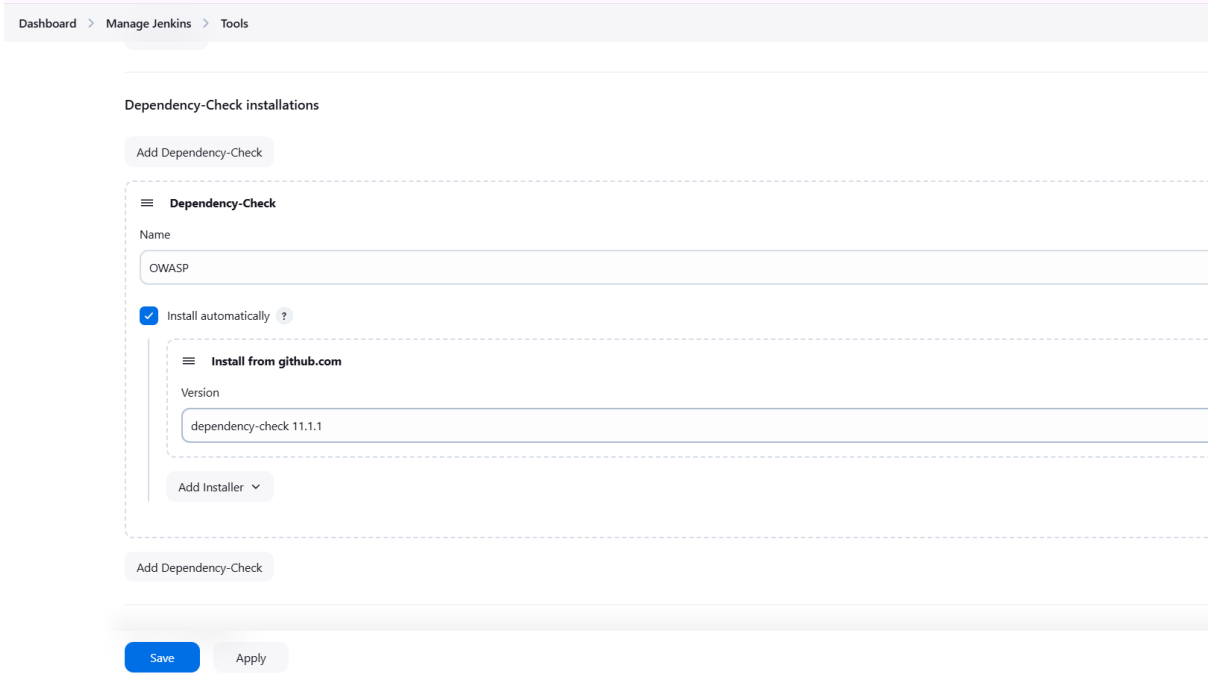
  sudo apt-get install trivy -y

```
ubuntu@ip-172-31-46-111:~$ trivy -v
Version: 0.58.1
ubuntu@ip-172-31-46-111:~$
```

- **Go to Jenkins Master and click on Manage Jenkins --> Plugins --> Available plugins install the below plugins:**
  - **OWASP**
  - **SonarQube Scanner**
  - **Docker**
  - **Pipeline: Stage View**

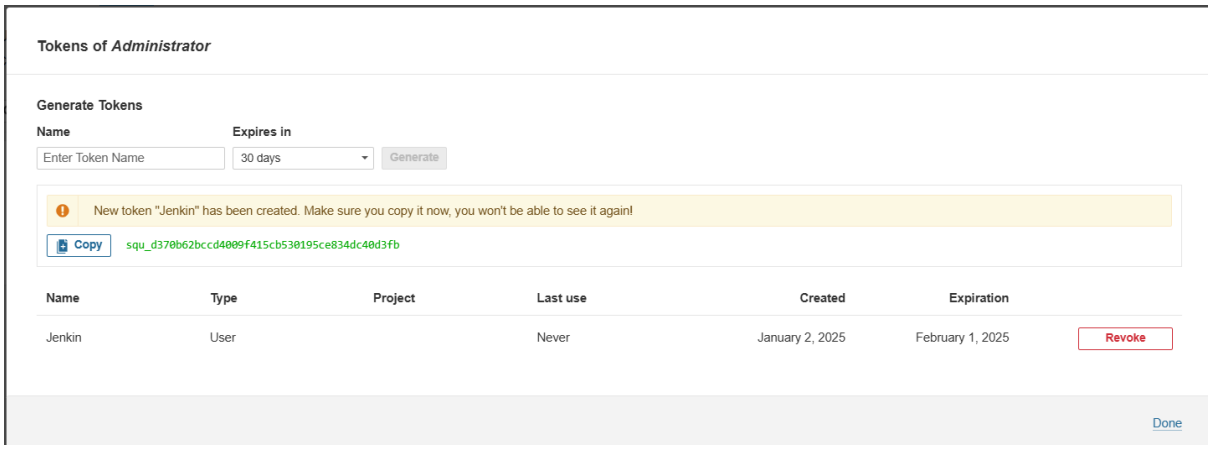


- **After OWASP plugin is installed, Now move to Manage jenkins --> Tools**

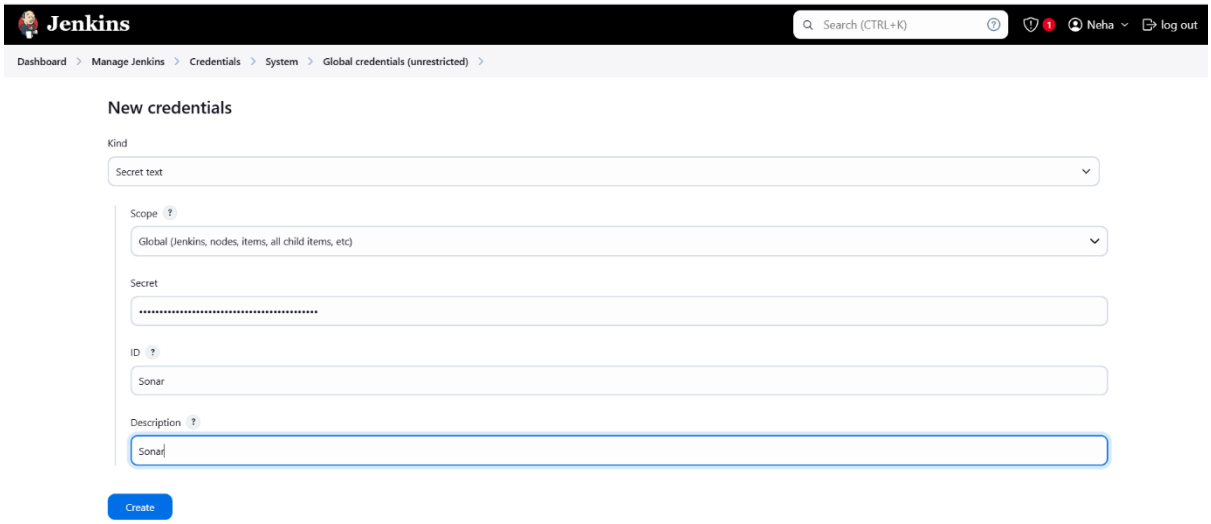


- **Login to SonarQube server and create the credentials for jenkins to integrate with SonarQube**

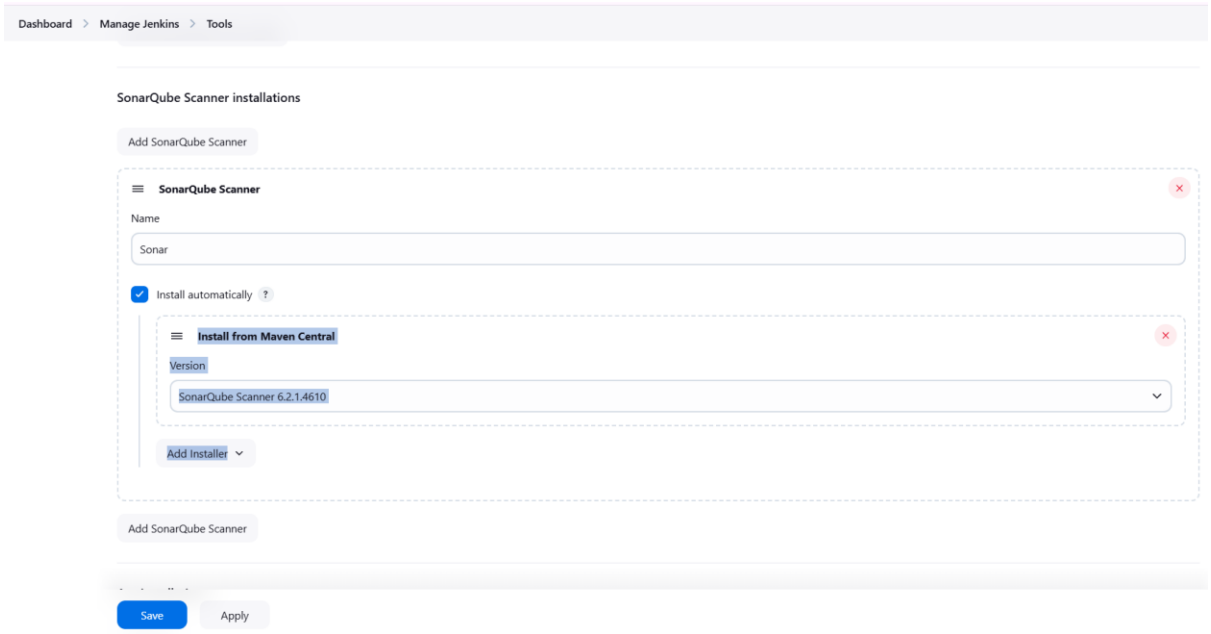
Navigate to Administration --> Security --> Users --> Token



- **Now, go to Manage Jenkins --> credentials and add Sonarqube credentials:**

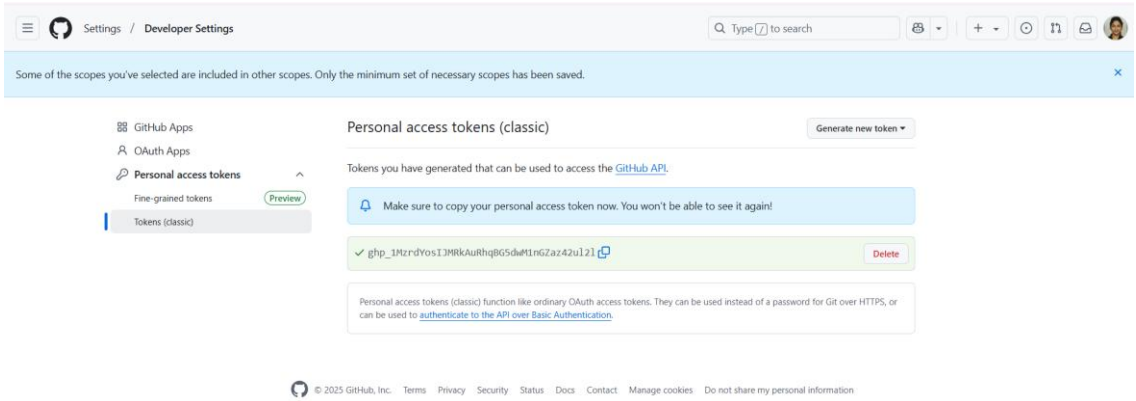


- **Go to Manage Jenkins --> Tools and search for SonarQube Scanner installations:**

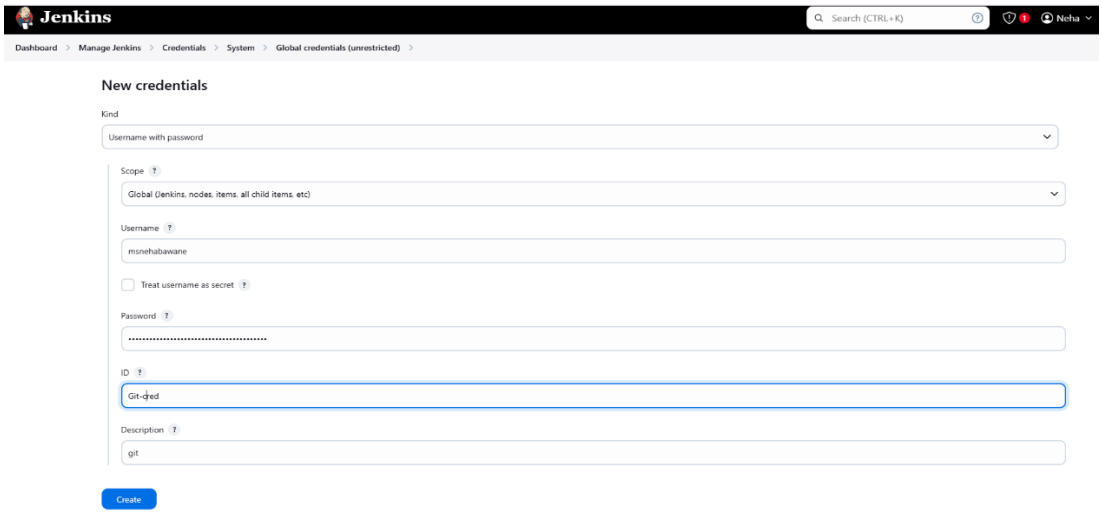


While adding github credentials add Personal Access Token in the password field.

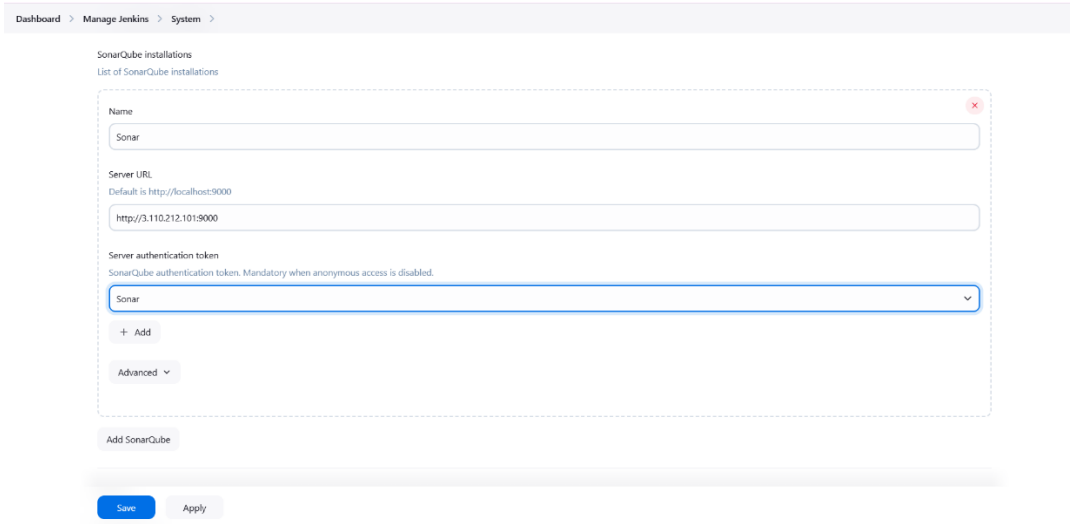
Go to Github account profile --> settings --> developer settings



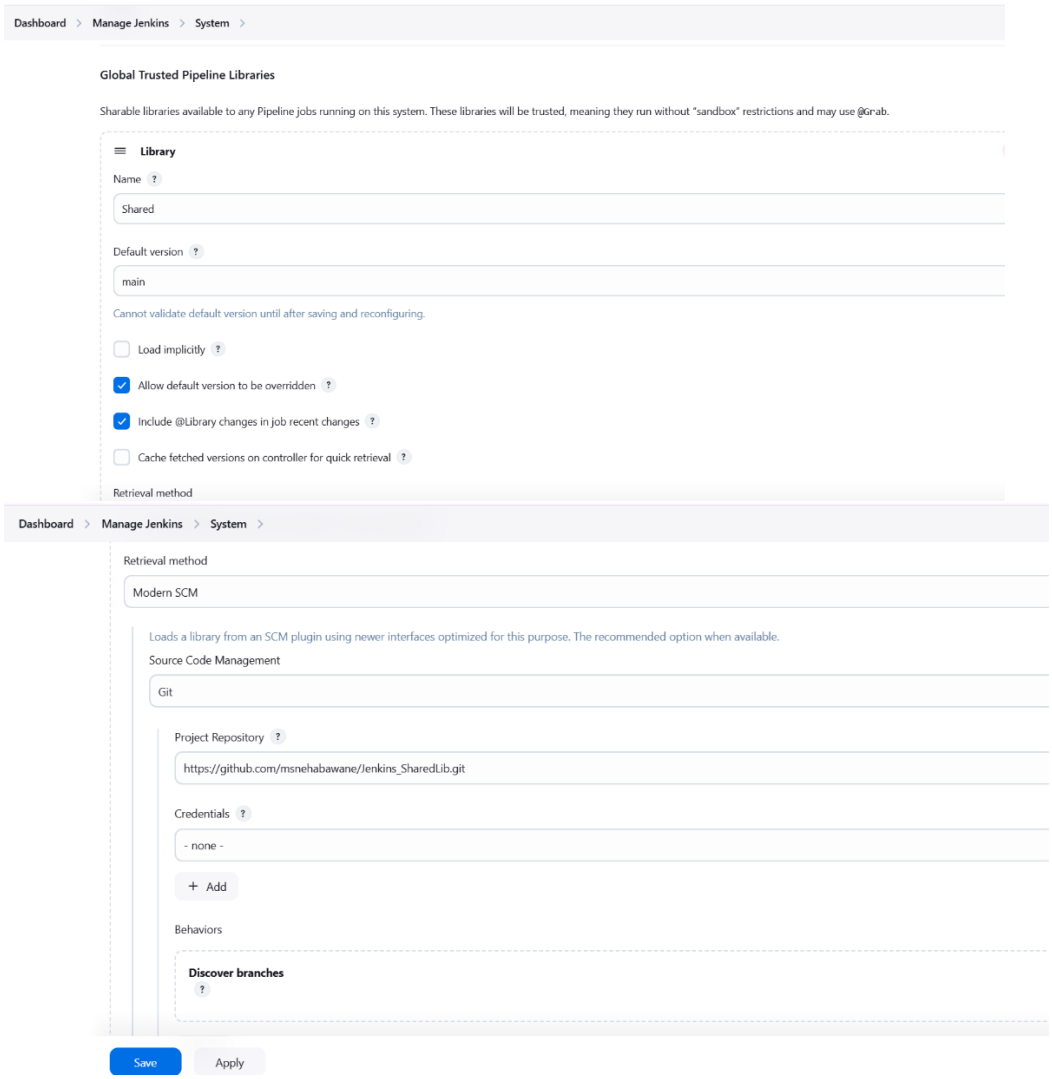
Go to Manage Jenkins --> credentials and add Github credentials to push updated code from the pipeline:



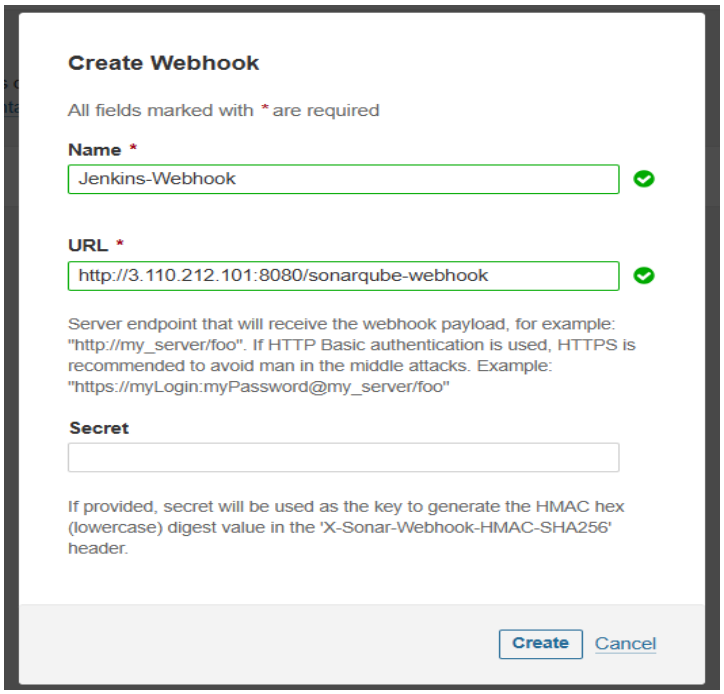
- **Go to Manage Jenkins --> System and search for SonarQube installations:**



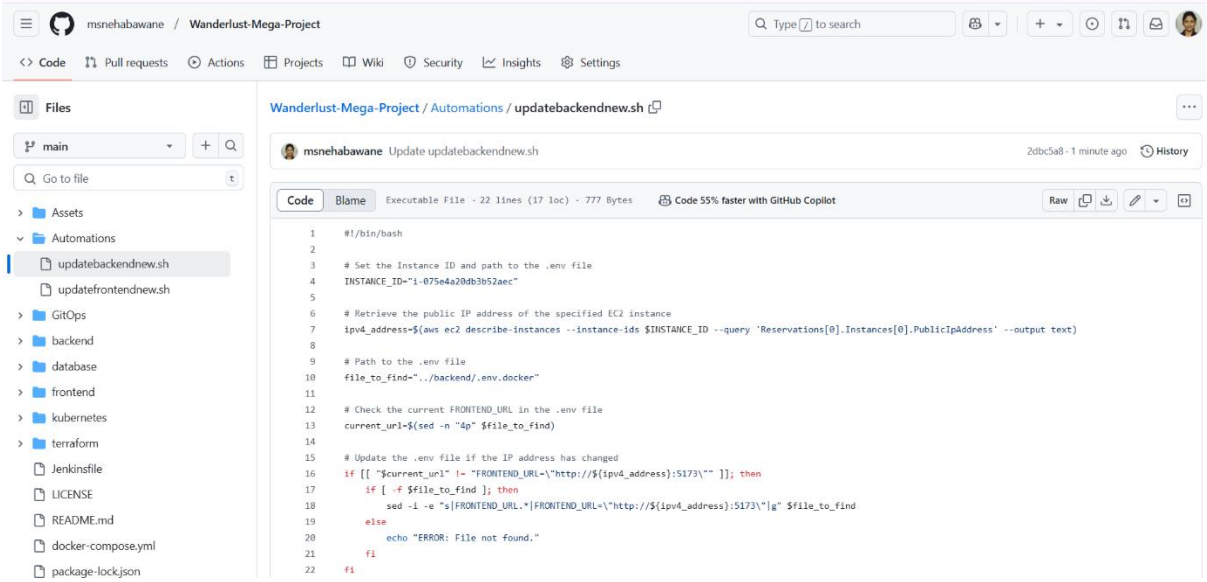
- **Now again, Go to Manage Jenkins --> System and search for Global Trusted Pipeline Libraries:</b**



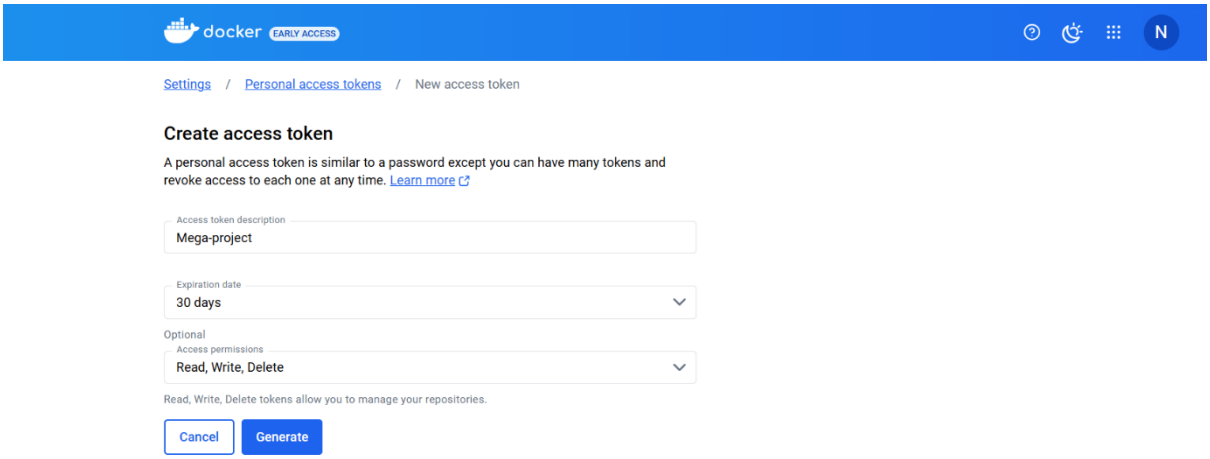
- **Login to SonarQube server, go to Administration --> Configuration —> Webhook and click on create**



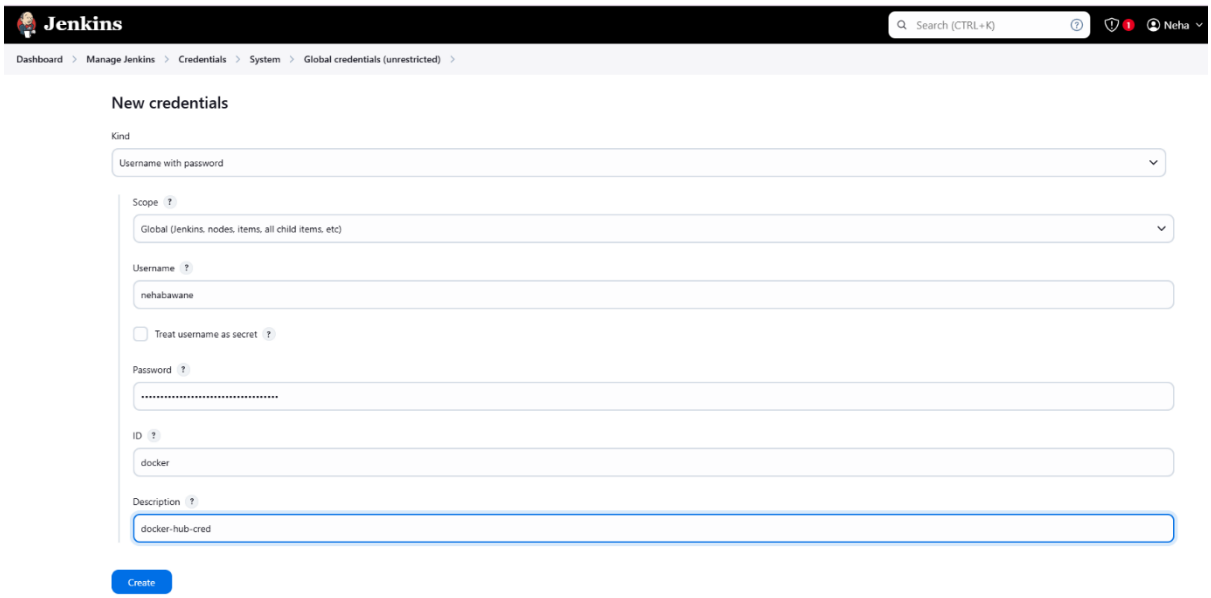
- **Now, go to github repository and under Automations directory update the instance-id field on both the updatefrontendnew.sh updatebackendnew.sh with the k8s worker node's instance id**



- Navigate to the DockerHub account **Settings --> Personal Access Token create access token**

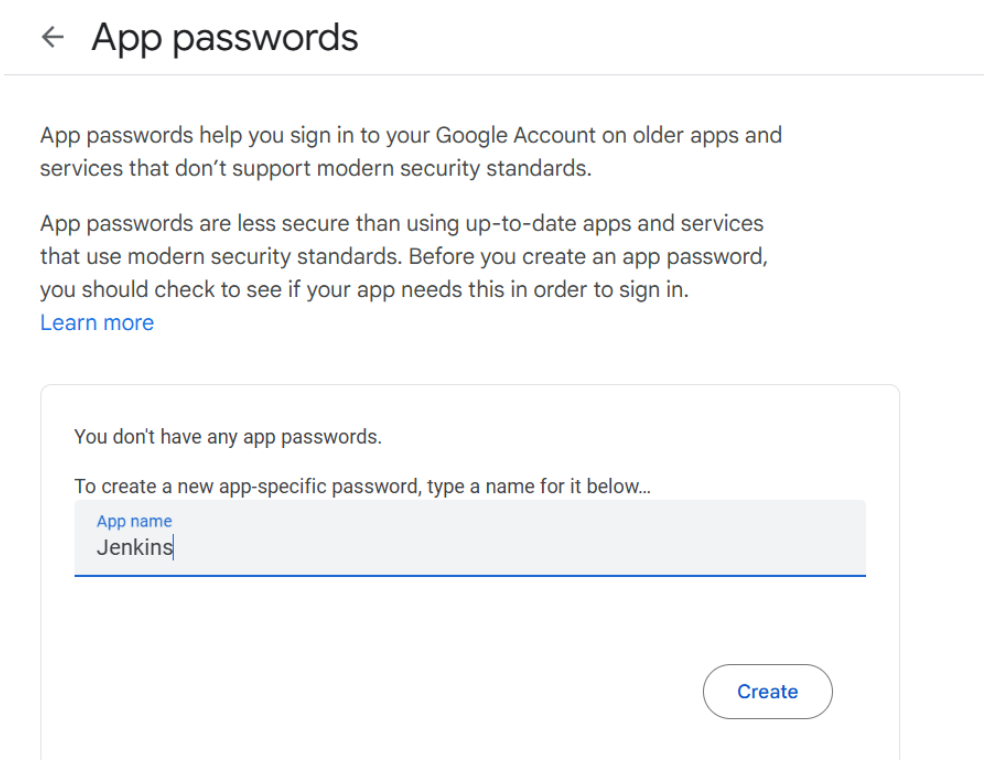


- **Navigate to Manage Jenkins --> credentials and add credentials for docker login to push docker image:**

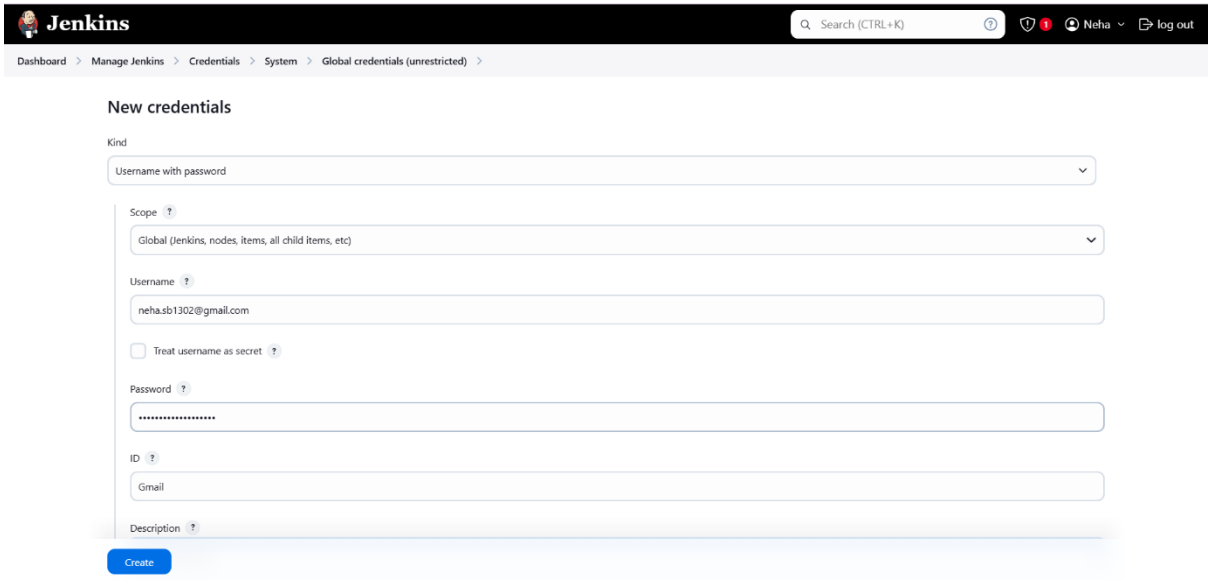


Step 4 : Steps to add Email notification

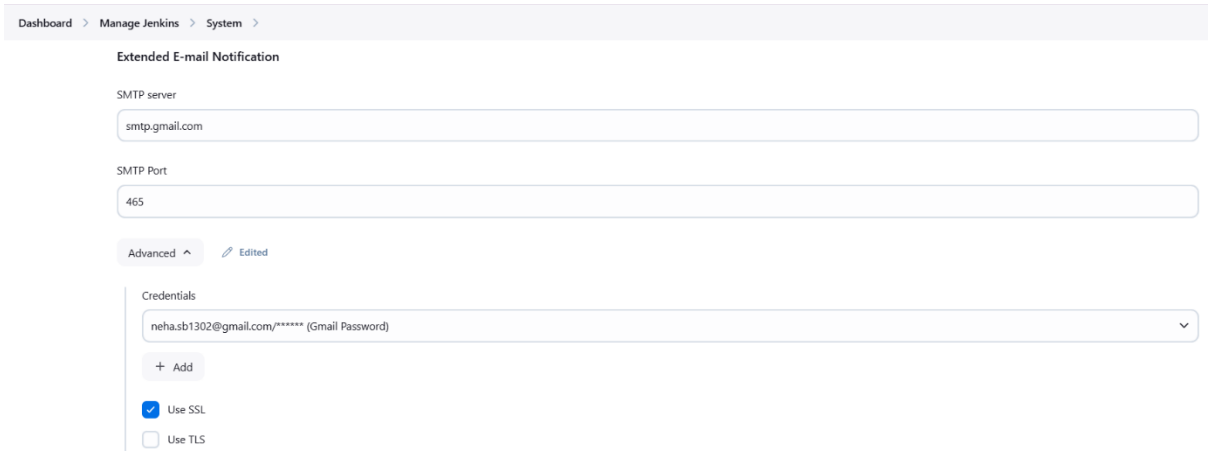
- **Now, we need to generate an application password from our gmail account to authenticate with jenkins**
    - Open gmail and go to Manage your Google Account --> Security
    - Make sure 2 step verification must be on
    - Search for App password and create a app password for jenkins



- **Once, app password is create and go back to jenkins Manage Jenkins --> Credentials to add username and password for email notification**



- **Go back to Manage Jenkins --> System and search for Extended E-mail Notification**
- **Scroll down and search for E-mail Notification and setup email notification**
- **Enter your gmail password which we copied recently in password field E-mail Notification --> Advance**



Step 5 : Install and Configure ArgoCD (Master Machine)

- **Create Argocd namespace**

  kubectl create namespace argocd

- **Apply Argocd manifest**

  kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

- **Make sure all pods are running in Argocd namespace**

watch kubectl get pods -n argocd

- **Install Argocd CLI**

  sudo curl --silent --location -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/download/v2.4.7/argocd-linux-amd64

- **Provide executable permission**

  sudo chmod +x /usr/local/bin/argocd

- **Check Argocd services**

  kubectl get svc -n argocd

- **Change ArgoCD server's service from ClusterIP to NodePort**

  kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'

- **Confirm service is patched or not**

  kubectl get svc -n argocd

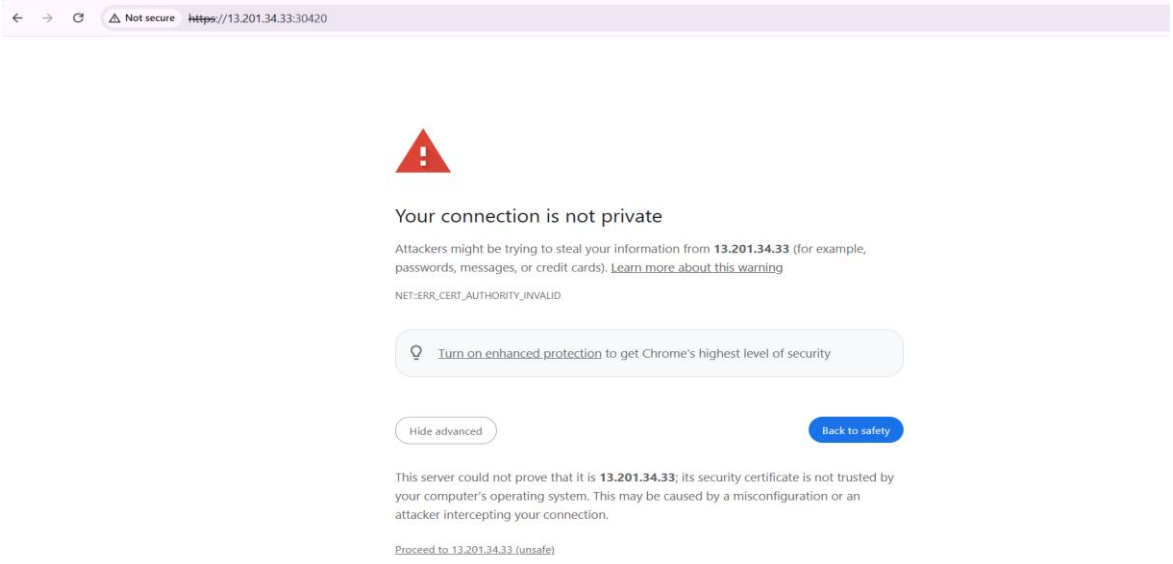
- **Check the port where Argo CD server is running and expose it on security groups of a worker node**

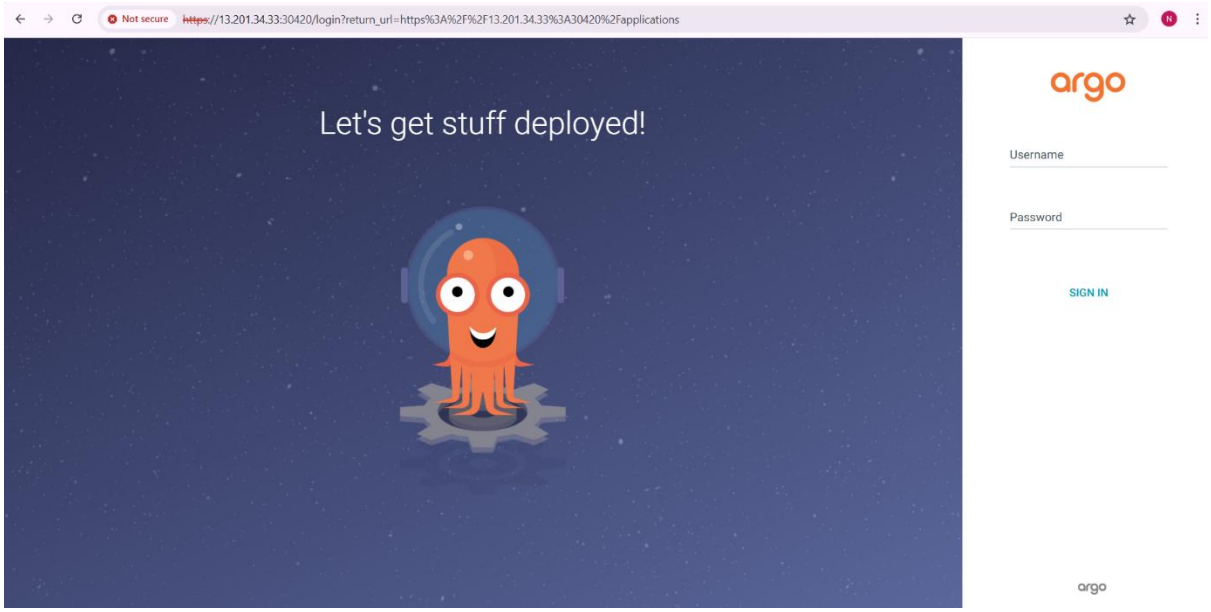
```
ubuntu@ip-172-31-46-111:~$ kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'
service/argocd-server patched
ubuntu@ip-172-31-46-111:~$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.100.42.215    <none>        7000/TCP,8080/TCP            4m16s
argocd-dex-server                         ClusterIP   10.100.27.203    <none>        5556/TCP,5557/TCP,5558/TCP   4m16s
argocd-metrics                            ClusterIP   10.100.47.115    <none>        8082/TCP                     4m16s
argocd-notifications-controller-metrics   ClusterIP   10.100.38.2      <none>        9001/TCP                     4m16s
argocd-redis                              ClusterIP   10.100.74.125    <none>        6379/TCP                     4m16s
argocd-repo-server                        ClusterIP   10.100.179.149   <none>        8081/TCP,8084/TCP            4m16s
argocd-server                             NodePort    10.100.236.100   <none>        80:30420/TCP,443:31287/TCP   4m16s
argocd-server-metrics                     ClusterIP   10.100.22.92     <none>        8083/TCP                     4m16s
ubuntu@ip-172-31-46-111:~$
```

- **Access it on browser, click on advance and proceed with**

  <public-ip-worker>:<port>

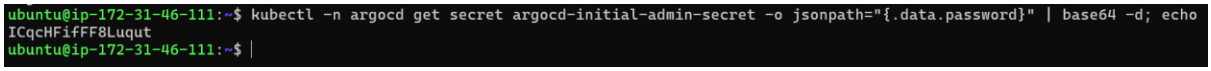
  https://13.201.34.33:30420



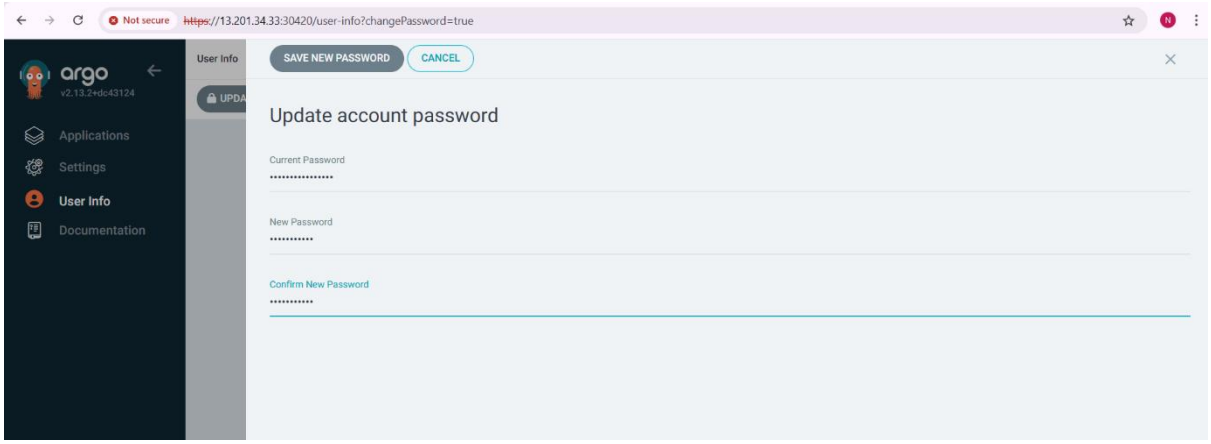


- **Fetch the initial password of Argocd server(master machine)**

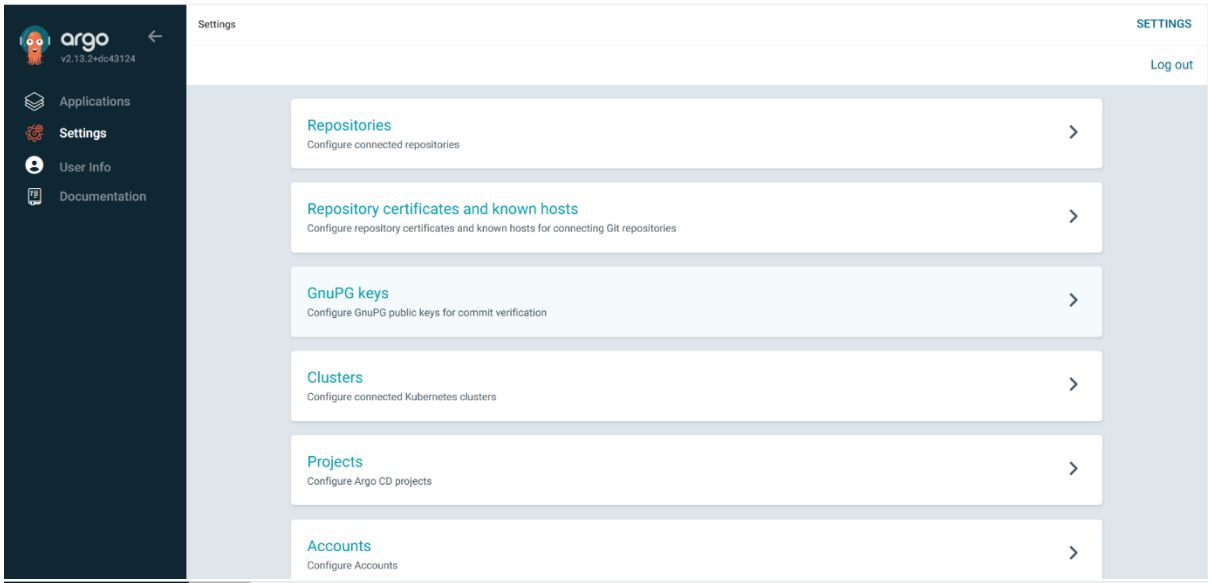
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo

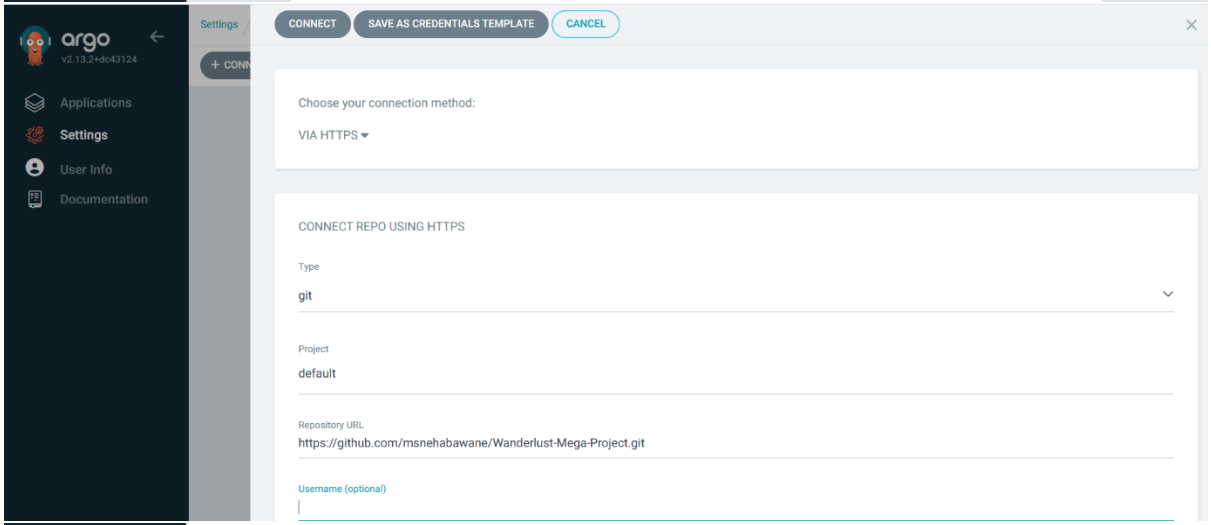


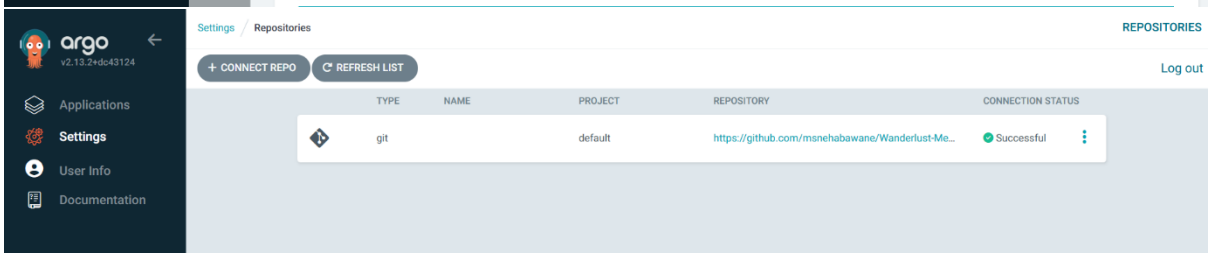
- **Username: admin**
- **Now, go to User Info and update your Argocd password**



Go to Settings --> Repositories and click on Connect repo







Connection should be successful

- **Go to Master Machine and add our own eks cluster to argocd for application deployment using cli**
    - **Login to ArgoCD from CLI**

      argocd login 13.201.34.33:30420 --username admin

```
ubuntu@ip-172-31-46-111:~$ argocd login 13.201.34.33:30420 --username admin
WARNING: server certificate had error: x509: cannot validate certificate for 13.201.34.33 because it doesn't contain any IP SANs. Proc
eed insecurely (y/n)? yes
Password:
'admin:login' logged in successfully
Context '13.201.34.33:30420' updated
ubuntu@ip-172-31-46-111:~$
```

      13.201.34.33:30420**--> This should be your argocd url**

- **Check how many clusters are available in argocd**

argocd cluster list

```
ubuntu@ip-172-31-46-111:~$ argocd cluster list
SERVER                          NAME        VERSION  STATUS   MESSAGE                                                  PROJECT
https://kubernetes.default.svc  in-cluster           Unknown  Cluster has no applications and is not being monitored.
ubuntu@ip-172-31-46-111:~$
```

- **Get your cluster name**

  kubectl config get-contexts

```
ubuntu@ip-172-31-46-111:~$ kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                                       AUTHINFO
NAMESPACE
*         Plyaground@wanderlust.ap-south-1.eksctl.io    wanderlust.ap-south-1.eksctl.io    Plyaground@wanderlust.ap-south-1.eksctl.io
          Plyaground@wanderlust.us-east-2.eksctl.io     wanderlust.us-east-2.eksctl.io     Plyaground@wanderlust.us-east-2.eksctl.io
ubuntu@ip-172-31-46-111:~$
```

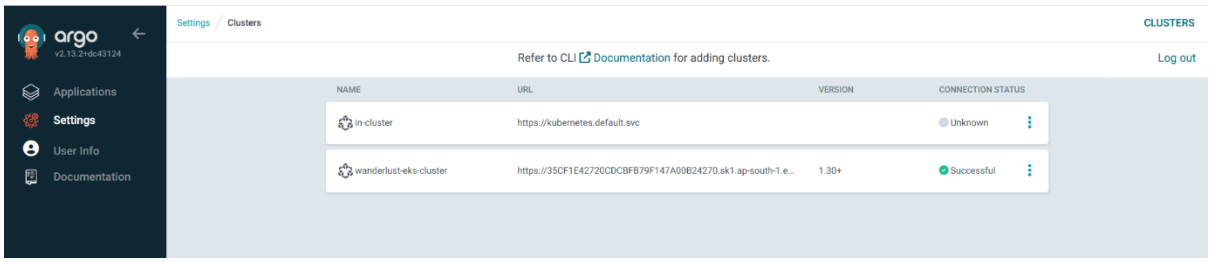
- **Add your cluster to argocd**

  argocd cluster add Plyaground@wanderlust.ap-south-1.eksctl.io --name wanderlust-eks-cluster

```
ubuntu@ip-172-31-46-111:~$ argocd cluster add Plyaground@wanderlust.ap-south-1.eksctl.io --name wanderlust-eks-cluster
WARNING: This will create a service account `argocd-manager` on the cluster referenced by context `Plyaground@wanderlust.ap-south-1.ek
sctl.io` with full cluster level privileges. Do you want to continue [y/N]? y
INFO[0004] ServiceAccount "argocd-manager" created in namespace "kube-system"
INFO[0004] ClusterRole "argocd-manager-role" created
INFO[0004] ClusterRoleBinding "argocd-manager-role-binding" created
INFO[0009] Created bearer token secret for ServiceAccount "argocd-manager"
Cluster 'https://35CF1E42720CDCBFB79F147A00B24270.sk1.ap-south-1.eks.amazonaws.com' added
ubuntu@ip-172-31-46-111:~$
```

Plyaground@wanderlust.ap-south-1.eksctl.io--> This should be your EKS Cluster Name.

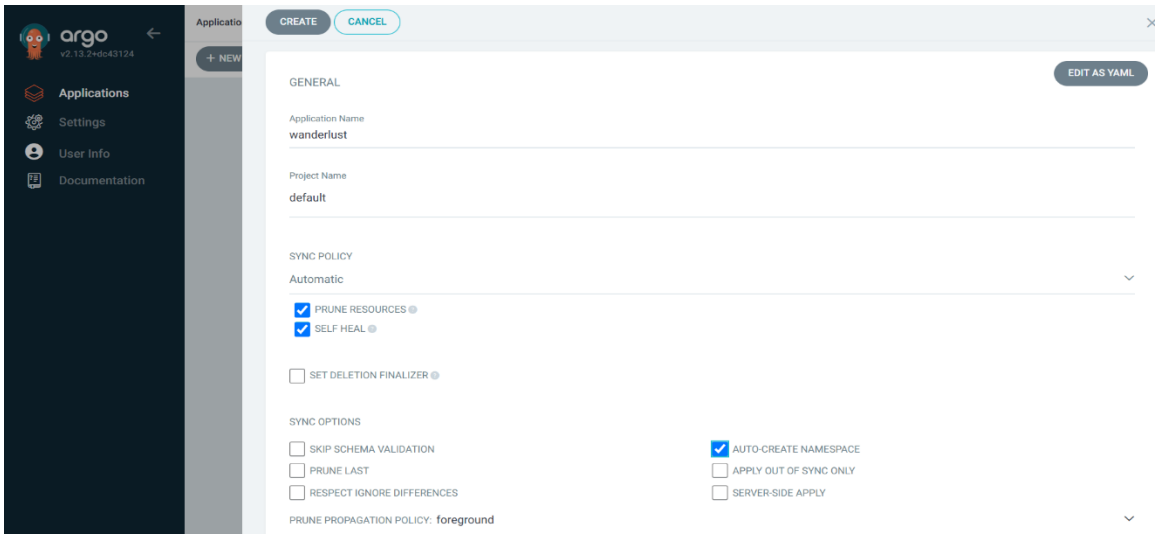
Once your cluster is added to argocd, go to argocd console Settings --> Clusters and verify it

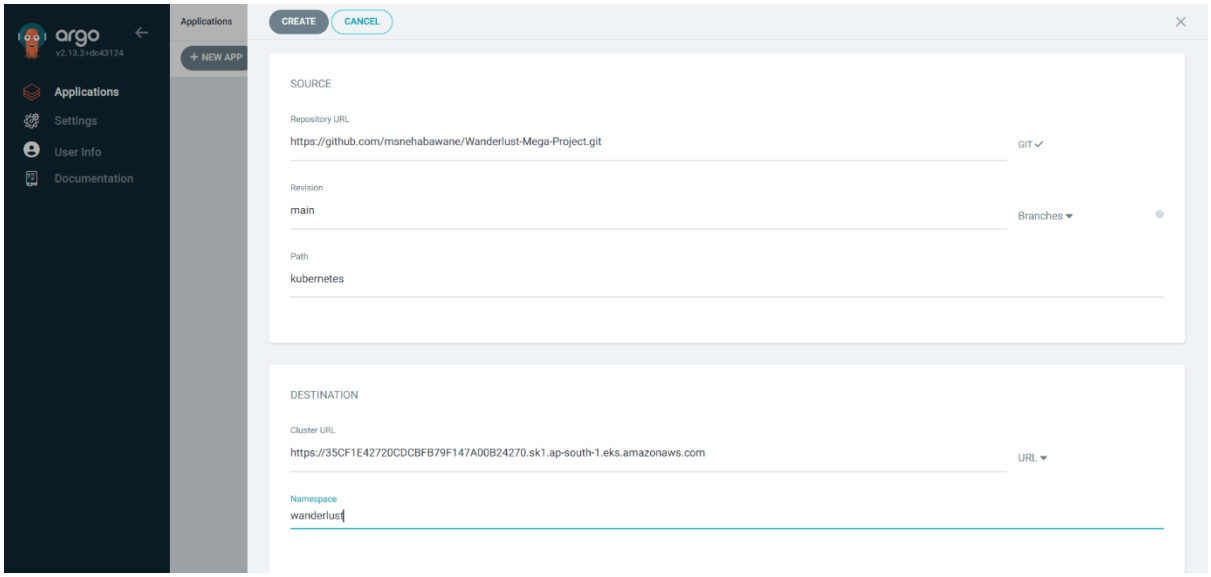


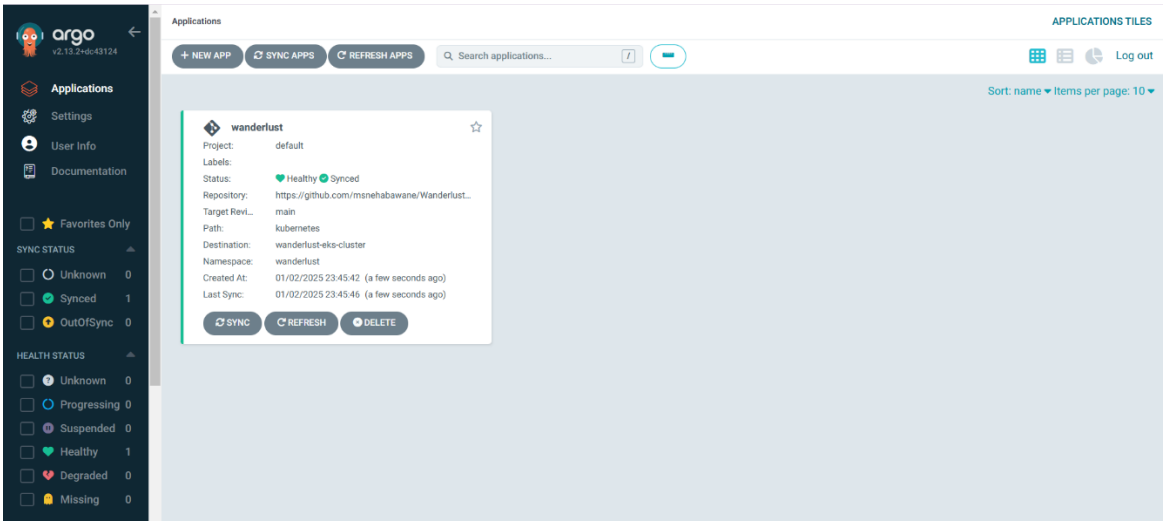
Connection should be successful

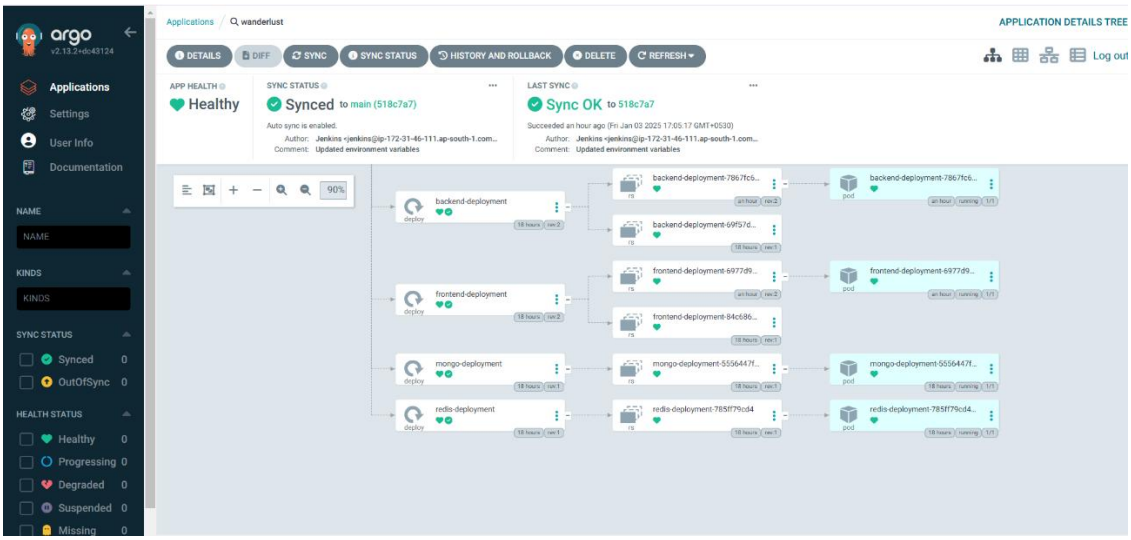
- **Now, go to Applications and click on New App**

Make sure to click on the Auto-Create Namespace option while creating argocd application



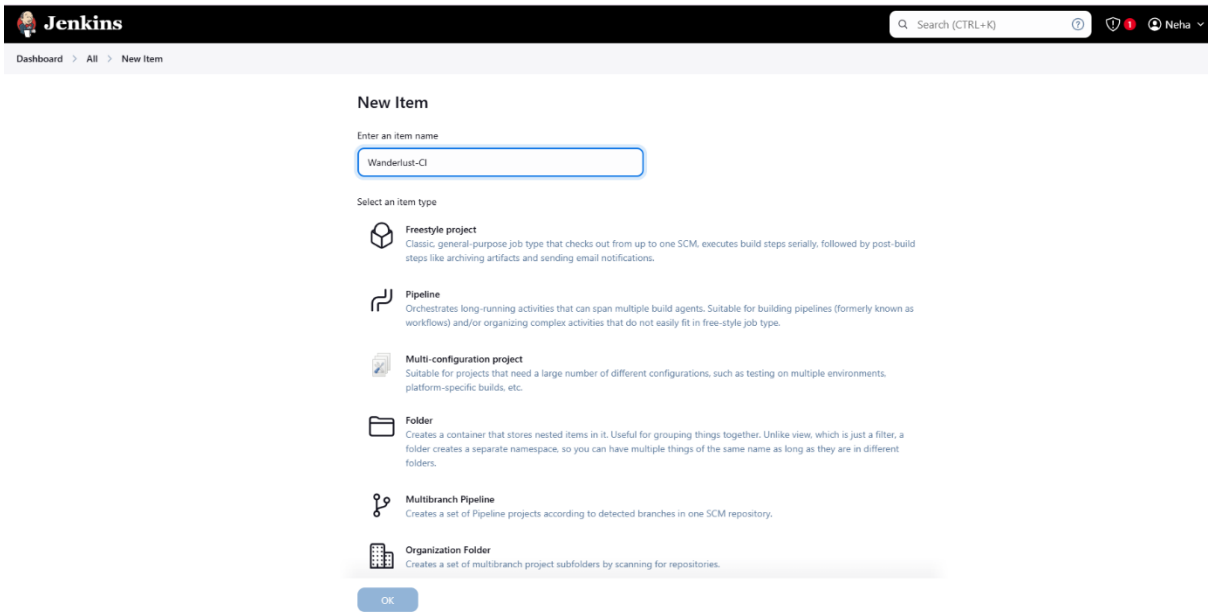


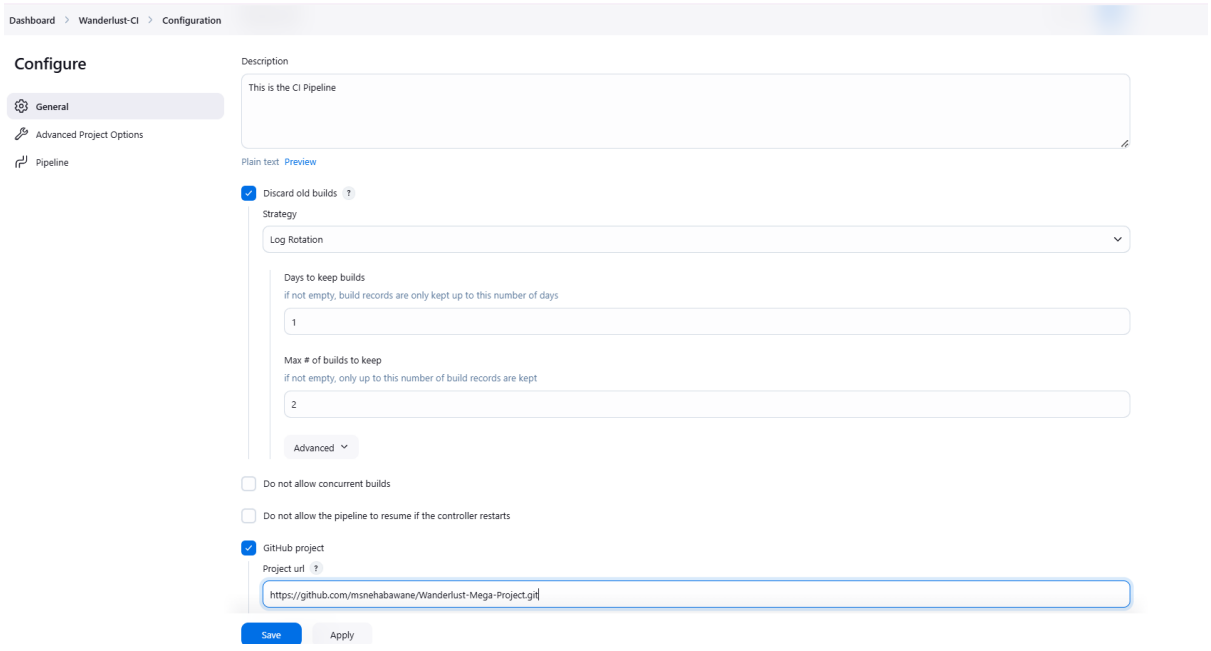


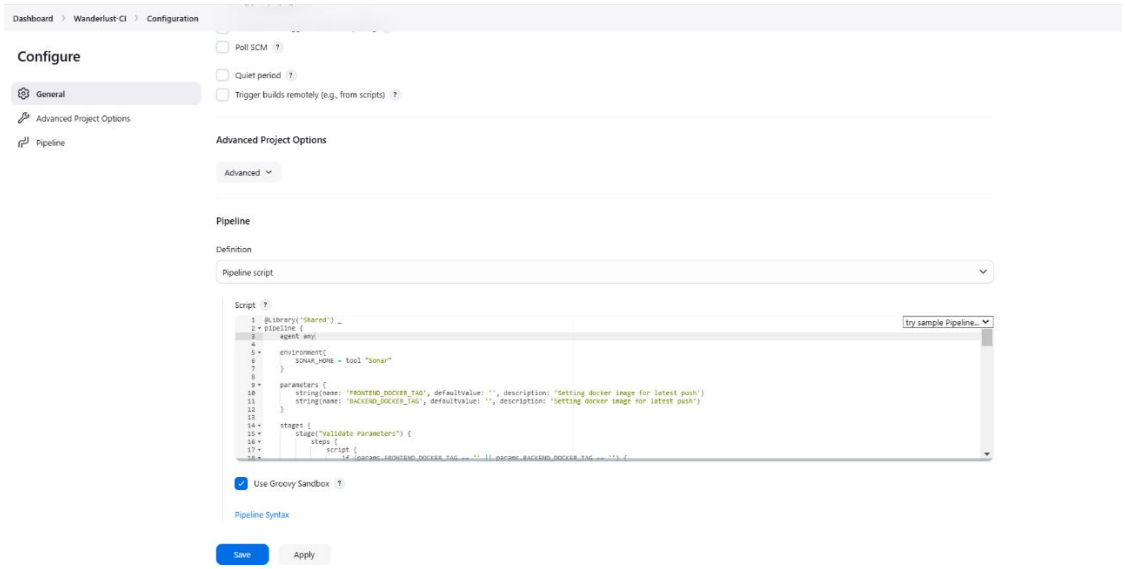


Open port 31000 and 31100 on worker node and Access it on browser

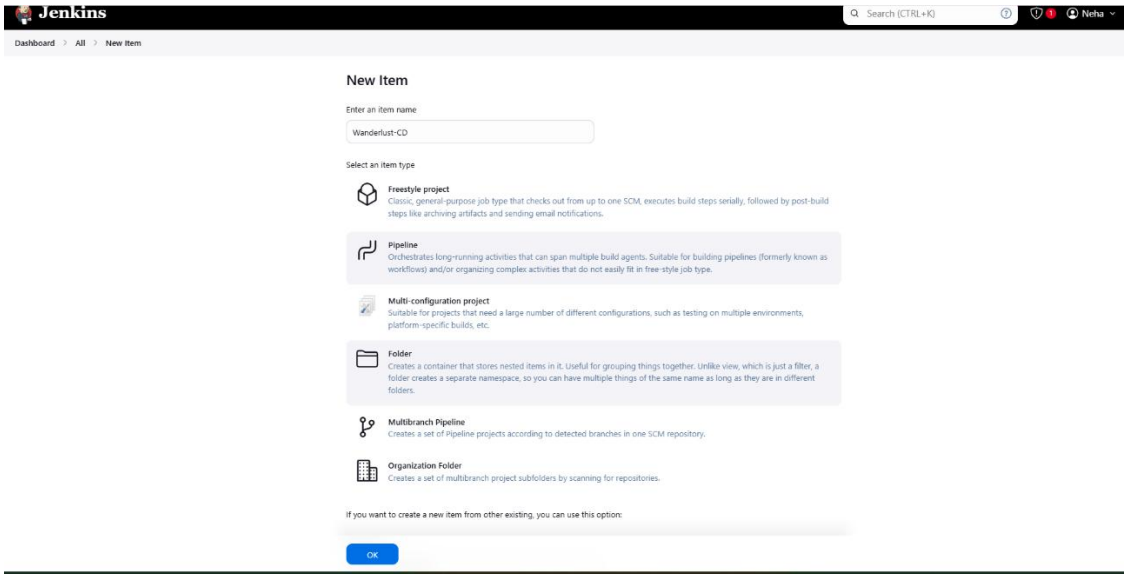
Step 6: Create a Wanderlust CI And CD Pipeline

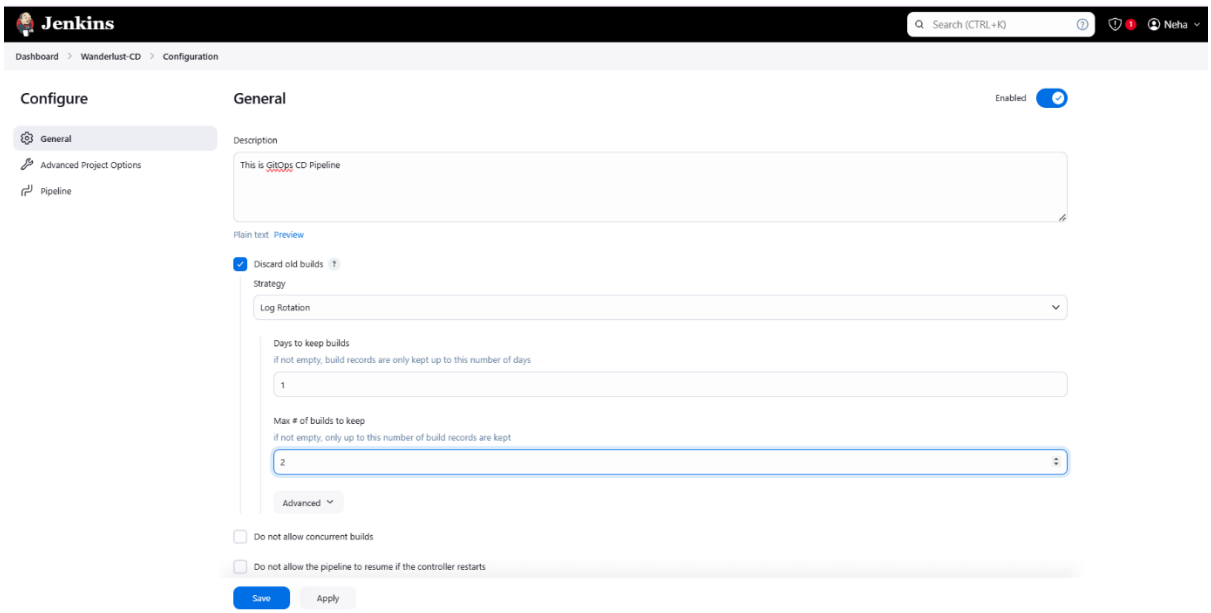


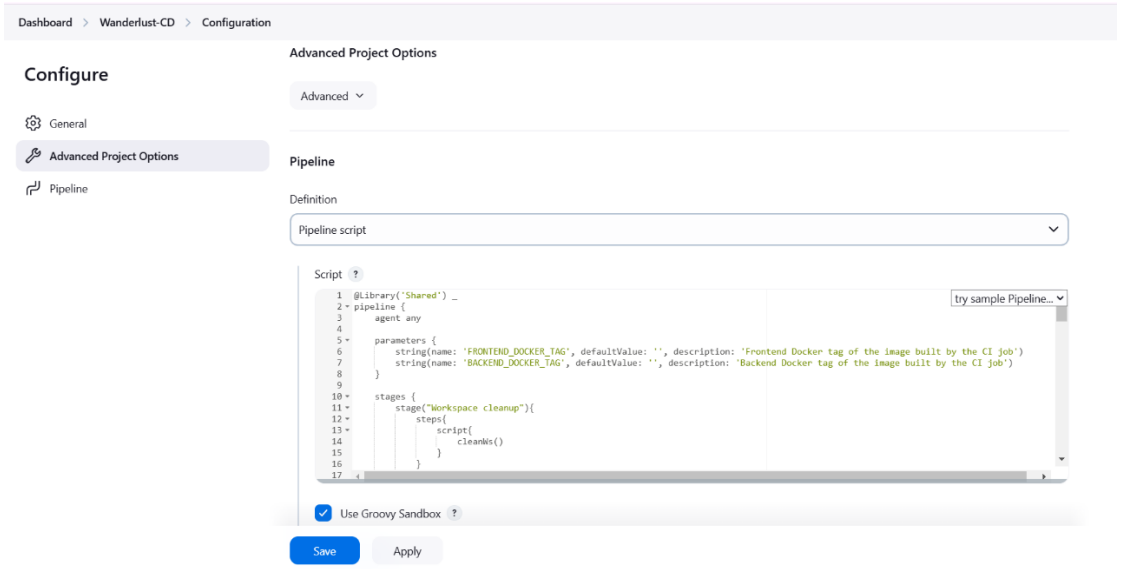


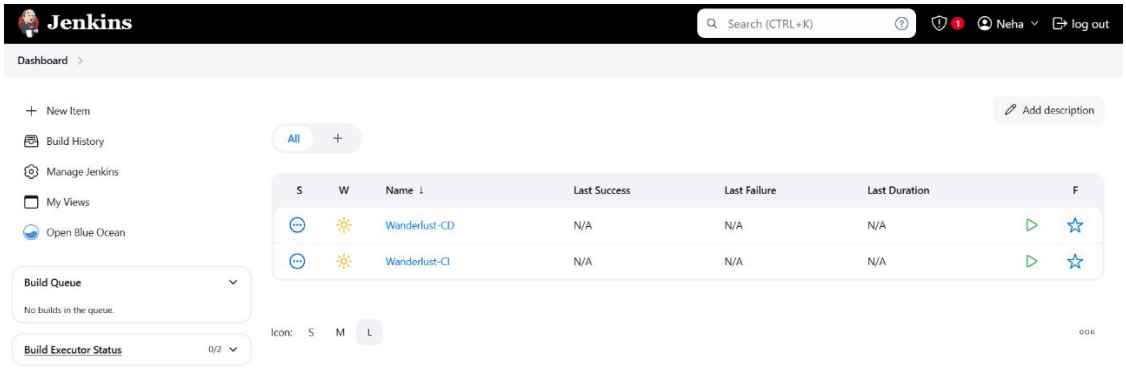

Dashboard › Wanderlust-CI › Configuration
Configure
General
Advanced Project Options
Pipeline
Poll SCM
Quiet period
Trigger builds remotely (e.g., from scripts)
Advanced Project Options
Advanced
Pipeline
Definition
Pipeline script
Script
try sample Pipeline...
Use Groovy Sandbox
Pipeline Syntax
Save
Apply

CD Pipeline

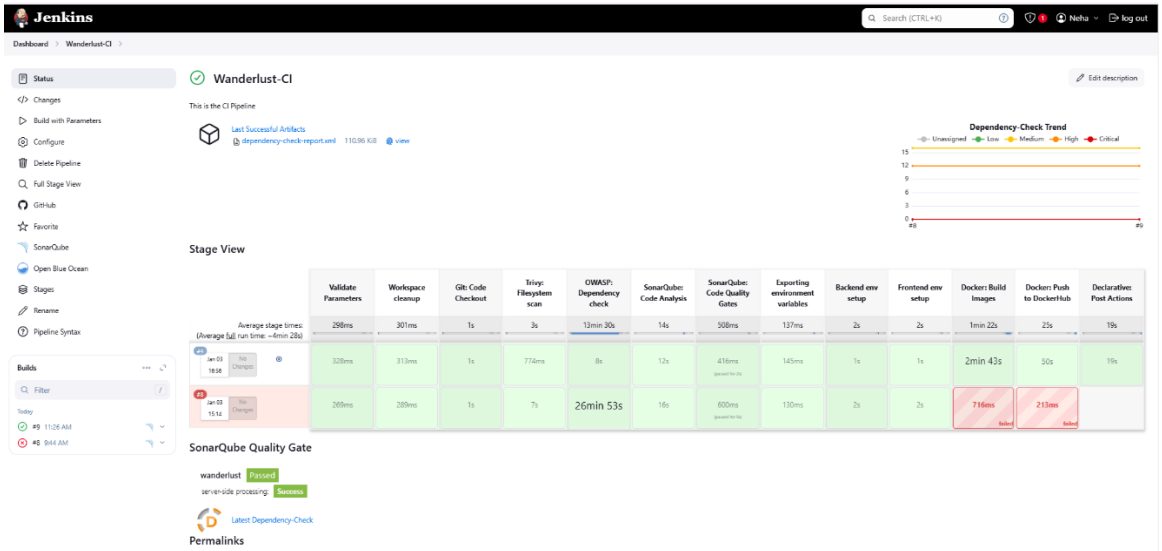

Jenkins
Search (CTRL+K)
Neha
Dashboard › All › New Item
New Item
Enter an item name
Wanderlust-CD
Select an item type
Freestyle project
Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.
Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.
Organization Folder
Creates a set of multibranch project subfolders by scanning for repositories.
If you want to create a new item from other existing, you can use this option:
OK


Jenkins
Search (CTRL+K)
Neha
Dashboard › Wanderlust-CD › Configuration
Configure
General
Advanced Project Options
Pipeline
General
Enabled
Description
This is GitOps CD Pipeline
Plain text Preview
Discard old builds
Strategy
Log Rotation
Days to keep builds
if not empty, build records are only kept up to this number of days
1
Max # of builds to keep
if not empty, only up to this number of build records are kept
2
Advanced
Do not allow concurrent builds
Do not allow the pipeline to resume if the controller restarts
Save
Apply

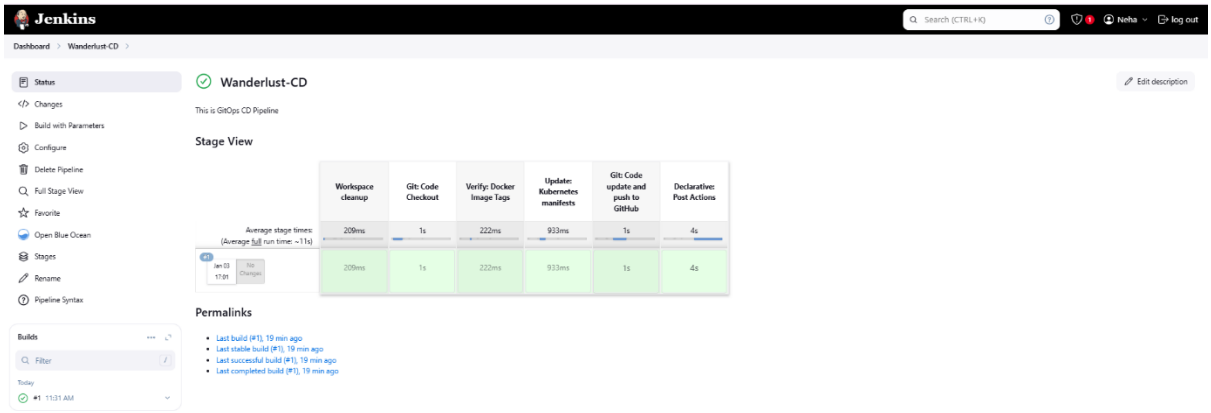




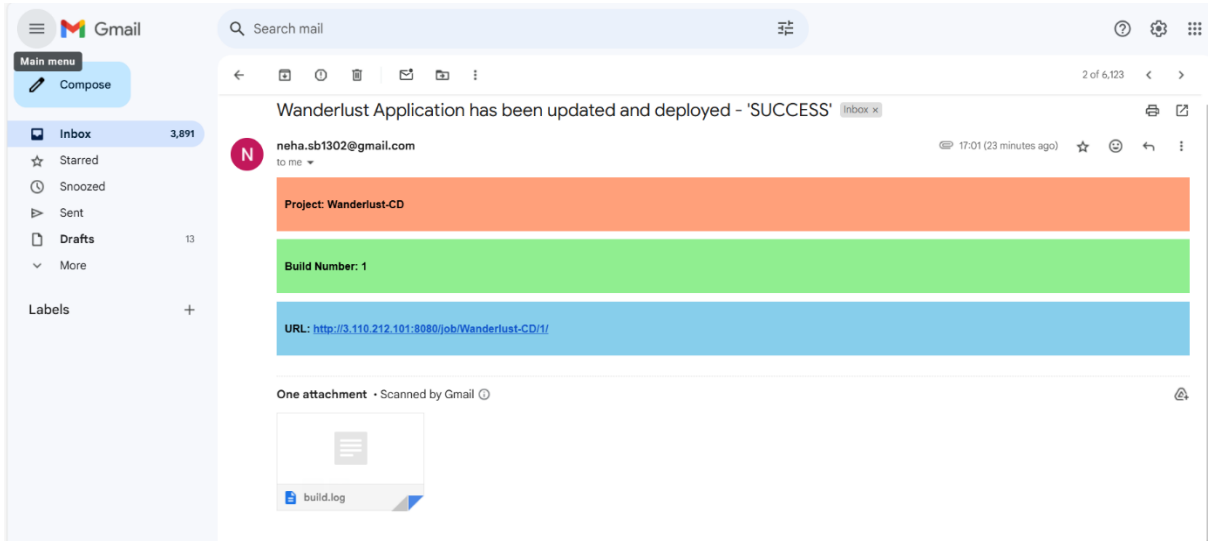
Wanderlust-CI Pipeline Output



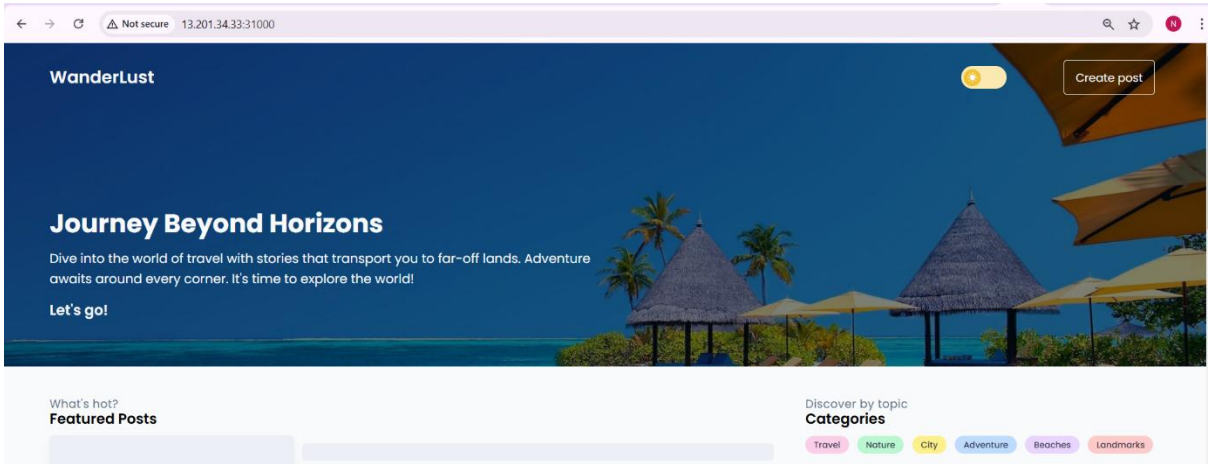
Wanderlust-CD Pipeline Output



Email Notification After Application is Deployed Successfully



Access application on cluster node public ip on port 31000



Step 7 : Monitor EKS cluster, Kubernetes components and workloads using Prometheus and Grafana via HELM (On Master machine)

- **Install Helm Chart**

  curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

  chmod 700 get_helm.sh

  ./get_helm.sh

- **Add Helm Stable Charts for Your Local Client**

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

- **Add Prometheus Helm Repository**

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

- **Create Prometheus Namespace**

  kubectl create namespace prometheus

  kubectl get ns

- **Install Prometheus using Helm**

  helm install stable prometheus-community/kube-prometheus-stack -n prometheus

- **Verify Prometheus installation**

  kubectl get pods -n prometheus

- **Check the services file (svc) of the Prometheus**

  kubectl get svc -n prometheus

- **Expose Prometheus and Grafana to the external world through Node Port**

Important

change it from Cluster IP to NodePort after changing make sure you save the file and open the assigned nodeport to the service.

kubectl edit svc stable-kube-prometheus-sta-prometheus -n prometheus

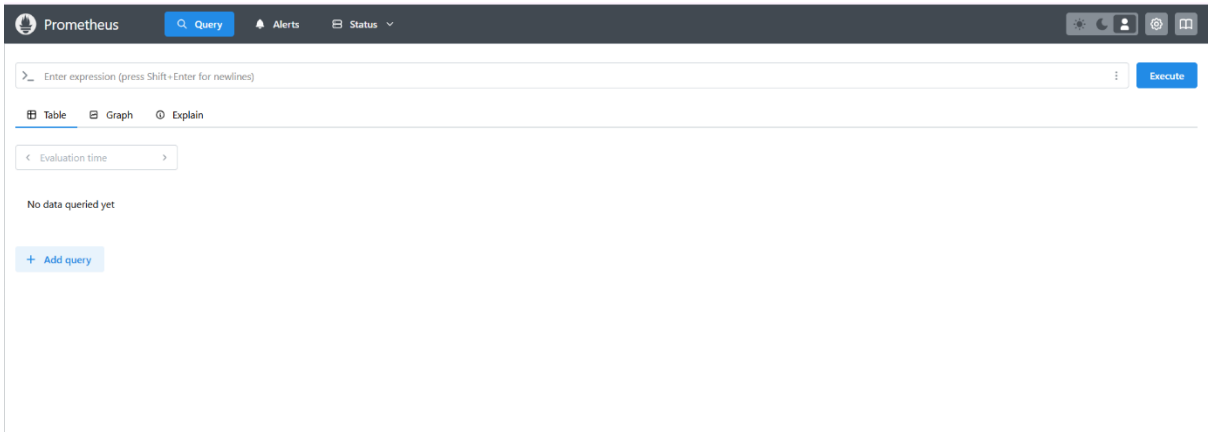
```
    app.kubernetes.io/version: 67.6.0
    chart: kube-prometheus-stack-67.6.0
    heritage: Helm
    release: stable
    self-monitor: "true"
  name: stable-kube-prometheus-sta-prometheus
  namespace: prometheus
  resourceVersion: "212510"
  uid: be076738-96f3-4cb2-a864-db6fbb3114d1
spec:
  clusterIP: 10.100.89.140
  clusterIPs:
  - 10.100.89.140
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    nodePort: 31177
    port: 9090
    protocol: TCP
    targetPort: 9090
  - appProtocol: http
    name: reloader-web
    nodePort: 31854
    port: 8080
    protocol: TCP
    targetPort: reloader-web
  selector:
    app.kubernetes.io/name: prometheus
    operator.prometheus.io/name: stable-kube-prometheus-sta-prometheus
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
-- INSERT --
```

- **Verify service**

  kubectl get svc -n prometheus

```
ubuntu@ip-172-31-46-111:~$ kubectl get svc -n prometheus
NAME                                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                         AGE
alertmanager-operated                    ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP      4m5s
prometheus-operated                      ClusterIP   None             <none>        9090/TCP                        4m5s
stable-grafana                           ClusterIP   10.100.26.143    <none>        80/TCP                          4m11s
stable-kube-prometheus-sta-alertmanager  ClusterIP   10.100.210.111   <none>        9093/TCP,8080/TCP               4m11s
stable-kube-prometheus-sta-operator      ClusterIP   10.100.246.41    <none>        443/TCP                         4m11s
stable-kube-prometheus-sta-prometheus    NodePort    10.100.89.140    <none>        9090:31177/TCP,8080:31854/TCP   4m11s
stable-kube-state-metrics                ClusterIP   10.100.154.163   <none>        8080/TCP                        4m11s
stable-prometheus-node-exporter          ClusterIP   10.100.254.124   <none>        9100/TCP                        4m11s
ubuntu@ip-172-31-46-111:~$
```

Access your Prometheus here : http://13.201.34.33:31177/query



- **Now,let's change the SVC file of the Grafana and expose it to the outer world**

  kubectl edit svc stable-grafana -n prometheus

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    meta.helm.sh/release-name: stable
    meta.helm.sh/release-namespace: prometheus
  creationTimestamp: "2025-01-03T10:23:29Z"
  labels:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: grafana
    app.kubernetes.io/version: 11.4.0
    helm.sh/chart: grafana-8.8.2
  name: stable-grafana
  namespace: prometheus
  resourceVersion: "211774"
  uid: af7bbccd-8445-402a-93a4-79d72a118d2a
spec:
  clusterIP: 10.100.26.143
  clusterIPs:
  - 10.100.26.143
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http-web
    port: 80
    protocol: TCP
    targetPort: 3000
  selector:
    app.kubernetes.io/instance: stable
    app.kubernetes.io/name: grafana
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

- **Check Grafana service**

  kubectl get svc -n prometheus

```
ubuntu@ip-172-31-46-111:~$ kubectl get svc -n prometheus
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                       AGE
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP    9m37s
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                      9m37s
stable-grafana                            NodePort    10.100.26.143    <none>        80:31032/TCP                  9m43s
stable-kube-prometheus-sta-alertmanager   ClusterIP   10.100.210.111   <none>        9093/TCP,8080/TCP             9m43s
stable-kube-prometheus-sta-operator       ClusterIP   10.100.246.41    <none>        443/TCP                       9m43s
stable-kube-prometheus-sta-prometheus     NodePort    10.100.89.140    <none>        9090:31177/TCP,8080:31854/TCP 9m43s
stable-kube-state-metrics                 ClusterIP   10.100.154.163   <none>        8080/TCP                      9m43s
stable-prometheus-node-exporter           ClusterIP   10.100.254.124   <none>        9100/TCP                      9m43s
ubuntu@ip-172-31-46-111:~$
```

- **Open the ports in cluster node's security group for both Prometheus and Grafana to access Externally**
- **Get a password for Grafana**

  kubectl get secret --namespace prometheus stable-grafana -o jsonpath="{.data.admin-password}" |

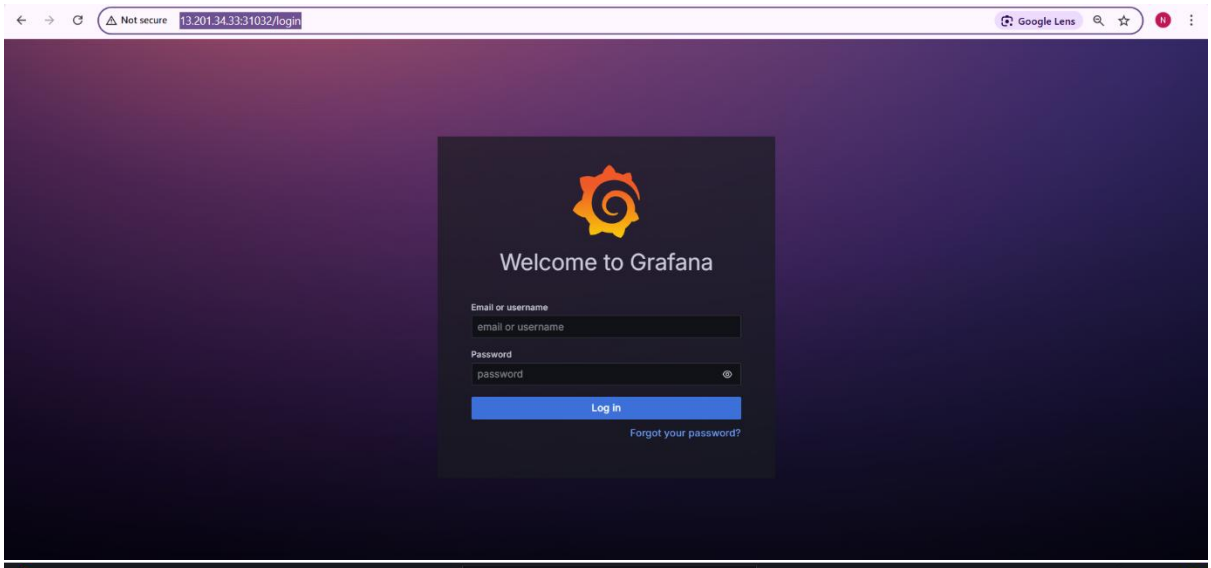
  base64 --decode ; echo

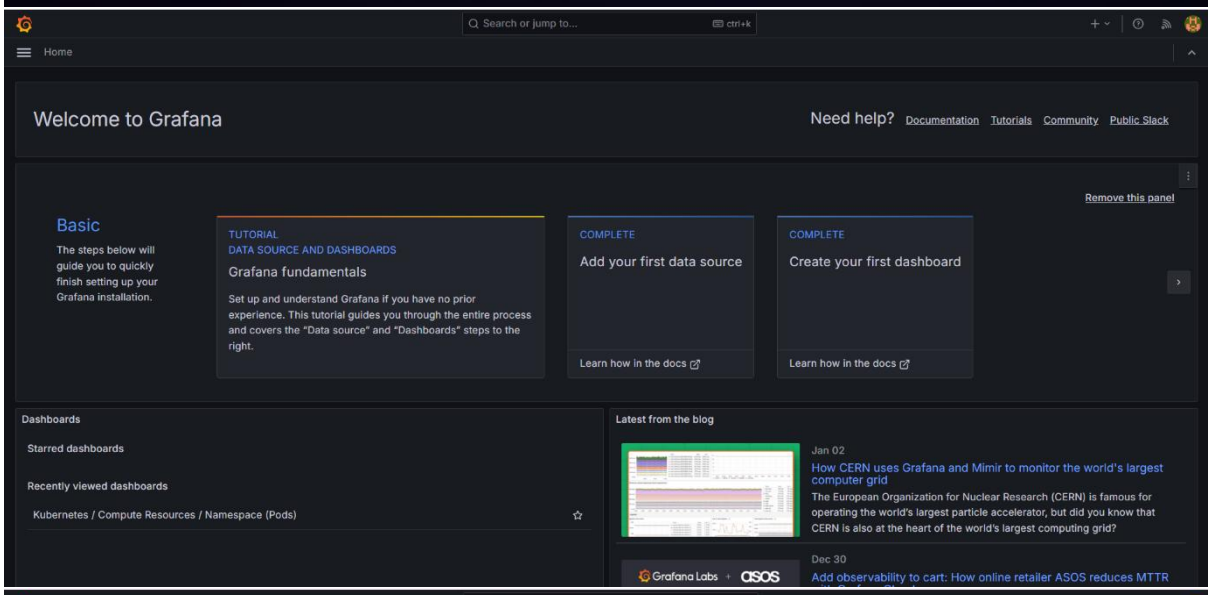
```
ubuntu@ip-172-31-46-111:~$ kubectl get secret --namespace prometheus stable-grafana -o jsonpath="{.data.admin-password}" | base64 --de
code ; echo
prom-operator
ubuntu@ip-172-31-46-111:~$
```

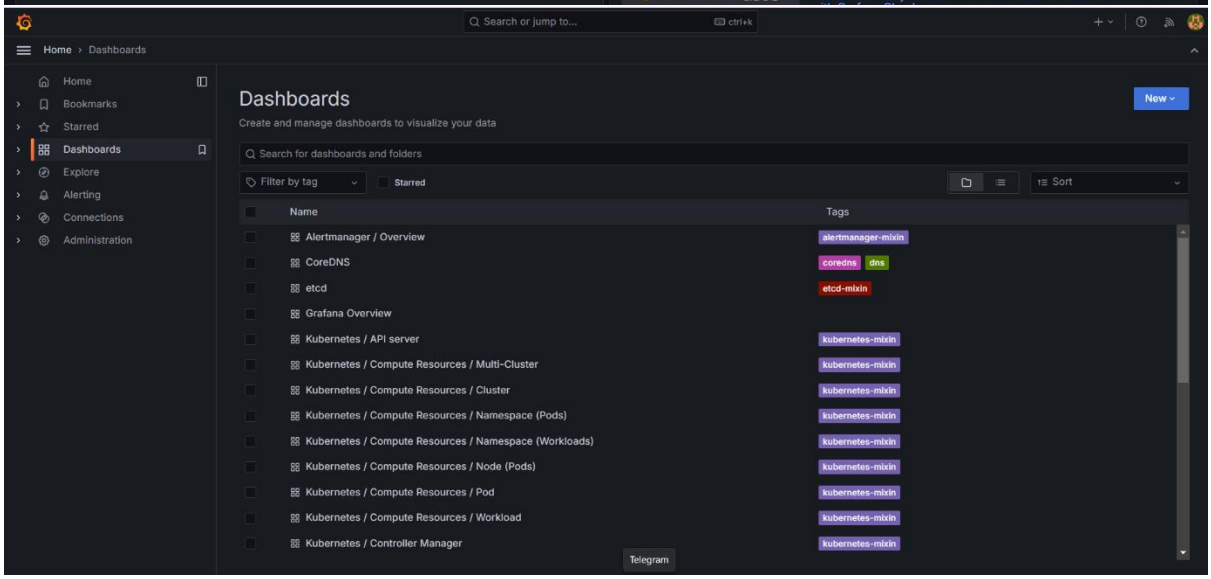
Username: admin

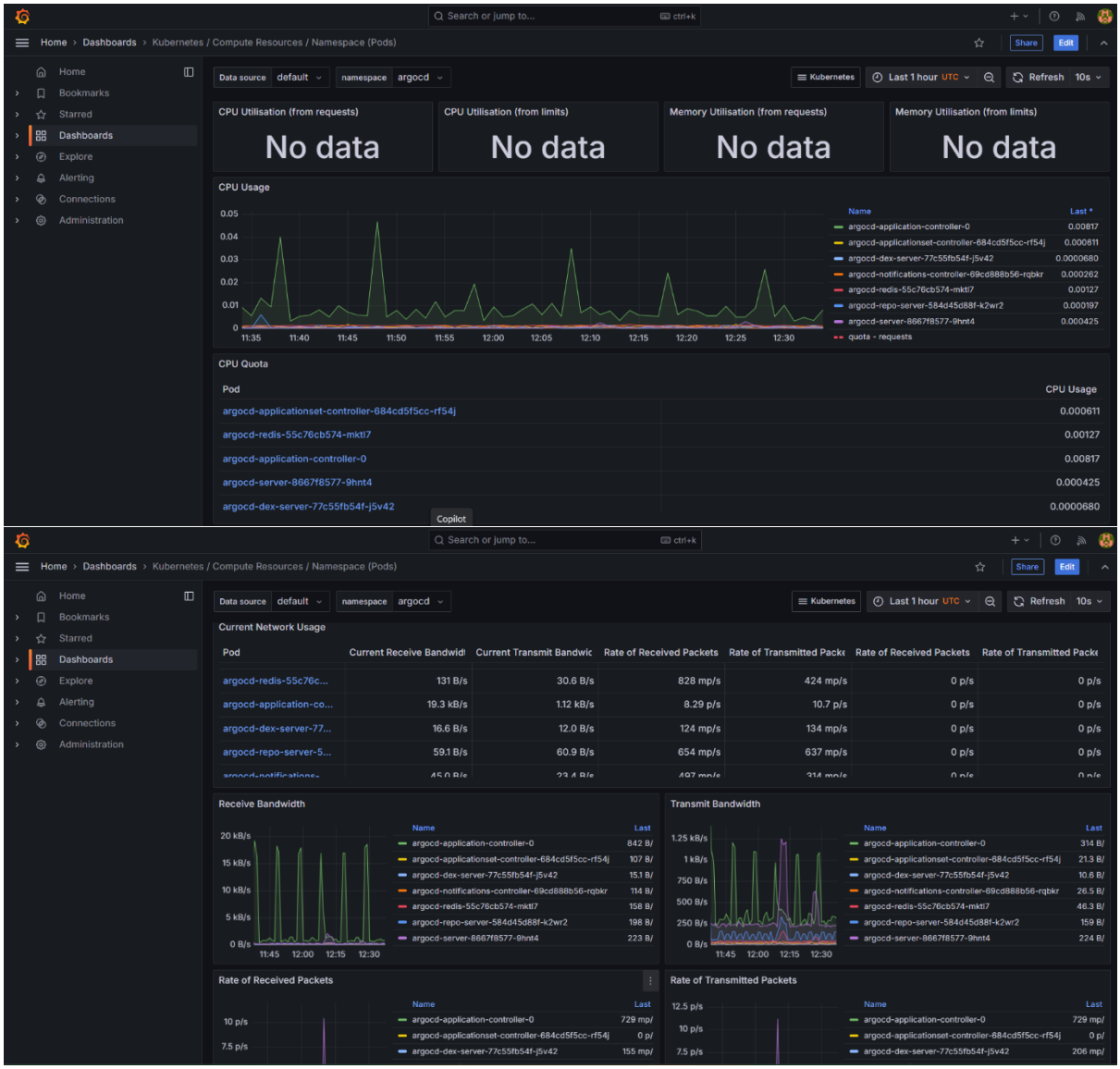
Now, view the Dashboard in Grafana

http://13.201.34.33:31032/login


Not secure
13.201.34.33:31032/login
Google Lens
Welcome to Grafana
Email or username
email or username
Password
password
Log in
Forgot your password?


Search or jump to...
ctrl+k
Home
Welcome to Grafana
Need help?
Documentation
Tutorials
Community
Public Slack
Remove this panel
Basic
The steps below will guide you to quickly finish setting up your Grafana installation.
TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals
Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.
COMPLETE
Add your first data source
Learn how in the docs
COMPLETE
Create your first dashboard
Learn how in the docs
Dashboards
Starred dashboards
Recently viewed dashboards
Kubernetes / Compute Resources / Namespace (Pods)
Latest from the blog
Jan 02
How CERN uses Grafana and Mimir to monitor the world's largest computer grid
The European Organization for Nuclear Research (CERN) is famous for operating the world's largest particle accelerator, but did you know that CERN is also at the heart of the world's largest computing grid?
Dec 30
Grafana Labs
ASOS
Add observability to cart: How online retailer ASOS reduces MTTR


Search or jump to...
ctrl+k
Home
Dashboards
Home
Bookmarks
Starred
Dashboards
Explore
Alerting
Connections
Administration
Dashboards
Create and manage dashboards to visualize your data
New
Search for dashboards and folders
Filter by tag
Starred
Sort
Name
Tags
Alertmanager / Overview
alertmanager-mixin
CoreDNS
coredns
dns
etcd
etcd-mixin
Grafana Overview
Kubernetes / API server
kubernetes-mixin
Kubernetes / Compute Resources / Multi-Cluster
kubernetes-mixin
Kubernetes / Compute Resources / Cluster
kubernetes-mixin
Kubernetes / Compute Resources / Namespace (Pods)
kubernetes-mixin
Kubernetes / Compute Resources / Namespace (Workloads)
kubernetes-mixin
Kubernetes / Compute Resources / Node (Pods)
kubernetes-mixin
Kubernetes / Compute Resources / Pod
kubernetes-mixin
Kubernetes / Compute Resources / Workload
kubernetes-mixin
Kubernetes / Controller Manager
kubernetes-mixin
Telegram



Summary

This DevOps pipeline integrates key tools and processes to automate the software development lifecycle efficiently. The pipeline begins with developers pushing code to **GitHub**, triggering a **Jenkins CI job** that performs dependency checks (OWASP), code quality analysis (SonarQube), and Docker image security scans (Trivy). The CI process builds Docker containers for deployment.

In the **CD pipeline**, Jenkins updates the Docker image version and pushes it to GitHub, where **ArgoCD** retrieves it for deployment to a **Kubernetes** cluster. Real-time monitoring is handled by **Prometheus** and **Grafana**, ensuring system health and performance. Email notifications inform stakeholders of pipeline events and issues. This workflow ensures secure, scalable, and automated application delivery, adhering to modern DevOps best practices.

Done!!

A Heartfelt Thanks to My Readers, Connections, and Supporters!

Let’s keep the momentum going – stay connected, stay inspired! 💡 🚀

Stay tuned for my next blog. I will keep sharing my learnings and knowledge here with you.

Let's learn together! I appreciate any comments or suggestions you may have to improve my blog content.

Happy Learning !!!

Thank you,

Neha Bawane