

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.

A screenshot of the AWS EC2 Instances page. The top navigation bar shows the AWS logo, a search bar, and a 'Mumbai' region dropdown. Below the header, there's a 'Launch instances' button. The main area displays a table titled 'Instances (1/1) Info'. The table has columns for Name, Instance ID, Instance state, Instance type, Status check, Alarm status, Availability Zone, and Public IPv4. One row is shown: 'Master-Machine' (Instance ID: i-0aaaf53860fa4a82d2, State: Running, Type: t2.large, Status: 2/2 checks passed, Zone: ap-south-1a, IP: ec2-3-110-2).

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

A screenshot of the AWS Security Groups Inbound rules table. The top navigation bar shows the AWS logo, a search bar, and a 'Manage tags' button. Below the header, there's an 'Edit inbound rules' button. The main area displays a table with columns: Security group rule ID, IP version, Type, Protocol, Port range, and Source. There are 9 rows listed, each defining a specific port range and protocol for the security group.

Security group rule ID	IP version	Type	Protocol	Port range	Source
sgr-09486d148f126ea21	IPv4	Custom TCP	TCP	6379	0.0.0.0/0
sgr-0d7bf027d9a8539dc	IPv4	SSH	TCP	22	0.0.0.0/0
sgr-0ac6395231b48d2c0	IPv4	SMTP	TCP	25	0.0.0.0/0
sgr-0f959a4e19d7c239f	IPv4	HTTPS	TCP	443	0.0.0.0/0
sgr-0df69d58eb9faa24b	IPv4	Custom TCP	TCP	3000 - 10000	0.0.0.0/0
sgr-0b7e0d853bec9f0ad	IPv4	Custom TCP	TCP	6443	0.0.0.0/0
sgr-05b29d2efa55e2729	IPv4	HTTP	TCP	80	0.0.0.0/0
sgr-09e3faacb21e80c10	IPv4	SMTPS	TCP	465	0.0.0.0/0
sgr-0238e8e1bf42a4532	IPv4	Custom TCP	TCP	30000 - 32767	0.0.0.0/0

- 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
```

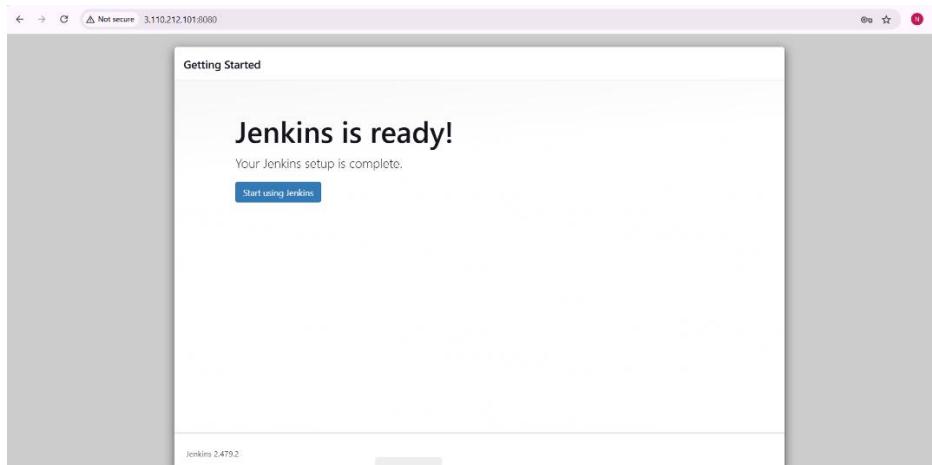
```
ubuntu@ip-172-31-46-111:~$ docker -v  
Docker version 26.1.3, build 26.1.3-0ubuntu1~24.04.1  
ubuntu@ip-172-31-46-111:~$
```

- **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 [*****q00I]:  
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      0:00:24  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
```

```

2025-01-02 12:32:35 [+] default addons vpc-cni, kube-proxy, coredns were not specified, will install them as EKS addons
2025-01-02 12:32:35 [+] 2 sequential tasks: { create cluster control plane "wanderlust",
  2 sequential sub-tasks: {
    1 task: { create addons },
    wait for control plane to become ready,
  }
}
2025-01-02 12:32:35 [+] building cluster stack "eksctl-wanderlust-cluster"
2025-01-02 12:32:35 [+] deploying stack "eksctl-wanderlust-cluster"
2025-01-02 12:33:05 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:33:35 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:34:35 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:35:36 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:36:36 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:37:36 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:38:36 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:39:36 [+] waiting for CloudFormation stack "eksctl-wanderlust-cluster"
2025-01-02 12:39:36 [!] recommended policies were found for "vpc-cni" addon, but since OIDC is disabled on the cluster, eksctl cannot
  configure the requested permissions; the recommended way to provide IAM permissions for "vpc-cni" addon is via pod identity associations, after addon creation is completed, add all recommended policies to the config file, under 'addon.PodIdentityAssociations', and run
  'eksctl update addon'
2025-01-02 12:39:36 [+] creating addon
2025-01-02 12:39:37 [+] successfully created addon
2025-01-02 12:39:37 [+] creating addon
2025-01-02 12:39:37 [+] successfully created addon
2025-01-02 12:39:37 [+] creating addon
2025-01-02 12:39:38 [+] successfully created addon
2025-01-02 12:41:38 [+] waiting for the control plane to become ready
2025-01-02 12:41:39 [+] saved kubeconfig as "/home/ubuntu/.kube/config"
2025-01-02 12:41:39 [+] tasks
2025-01-02 12:41:39 [+] all EKS cluster resources for "wanderlust" have been created
2025-01-02 12:41:40 [+] Hubectl command should work with "/home/ubuntu/.kube/config", try 'kubectl get nodes'
2025-01-02 12:41:40 [+] EKS cluster "wanderlust" in "ap-south-1" region is ready
ubuntu@ip-172-31-46-111:~$
```

- Associate IAM OIDC Provider (Master machine)

```

eksctl utils associate-iam-oidc-provider \
--region ap-south-1 \
--cluster wanderlust \
--approve
```

```

ubuntu@ip-172-31-46-111:~$ eksctl utils associate-iam-oidc-provider \
--region ap-south-1 \
--cluster wanderlust \
--approve
2025-01-02 12:59:06 [+] will create IAM Open ID Connect provider for cluster "Wanderlust" in "ap-south-1"
2025-01-02 12:59:07 [+] created IAM Open ID Connect provider for cluster "wanderlust" in "ap-south-1"
ubuntu@ip-172-31-46-111:~$ |
```

- 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

NAME	STATUS	ROLES	AGE	VERSION
ip-192-168-48-147.ap-south-1.compute.internal	Ready	<none>	30m	v1.30.7-eks-59bf375
ip-192-168-65-214.ap-south-1.compute.internal	Ready	<none>	30m	v1.30.7-eks-59bf375

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:its-community
```

The screenshot shows the SonarQube interface for creating a new project. At the top, there's a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, and Administration. A search bar is also present. Below the navigation, a message asks how to create the project. It then provides options for creating it from popular CI/CD platforms: Azure DevOps, Bitbucket Server, Bitbucket Cloud, GitHub, and GitLab. Each option has a 'From [Platform]' icon and a 'Set up global configuration' link. At the bottom, there's a manual creation option with a 'Manually' link and a code editor icon.

- **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**

Dashboard > Manage Jenkins > Plugins

Plugins

- Updates
- Available plugins**
- Installed plugins
- Advanced settings
- Download progress

Search available plugins

Install	Name	Released
<input checked="" type="checkbox"/>	OWASP Dependency-Check 5.6.0 Security DevOps Build Tools Build Reports This plug-in can independently execute a Dependency-Check analysis and visualize results. Dependency-Check is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities.	21 days ago
<input checked="" type="checkbox"/>	SonarQube Scanner 2.17.3 External Site/Tool Integrations Build Reports This plugin allows an easy integration of SonarQube, the open source platform for Continuous Inspection of code quality.	1 mo 15 days ago
<input checked="" type="checkbox"/>	Docker 1.7.0 Cloud Providers Cluster Management docker This plugin integrates Jenkins with Docker	2 mo 19 days ago
<input checked="" type="checkbox"/>	Pipeline Stage View 2.34 User Interface Pipeline Stage View Plugin.	1 yr 2 mo ago

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

Dashboard > Manage Jenkins > Tools

Dependency-Check installations

Add Dependency-Check

Dependency-Check

Name: OWASP

Install automatically ?

Install from github.com

Version: dependency-check 11.1.1

Add Installer ▾

Add Dependency-Check

Save **Apply**

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

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

Tokens of Administrator

Generate Tokens

Name	Expires in
Enter Token Name	30 days

Info New token "Jenkin" has been created. Make sure you copy it now, you won't be able to see it again!

Copy squ_d37eb62bccd4009f415cb530195ce834dc40d3fb

Name	Type	Project	Last use	Created	Expiration
Jenkin	User		Never	January 2, 2025	February 1, 2025

Revoke

Done

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

New credentials

Kind

Secret text

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

Secret

ID ? Sonar

Description ? Sonar

Create

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

SonarQube Scanner installations

Add SonarQube Scanner

Sonar

Name

Sonar

Install automatically

Version

SonarQube Scanner 6.2.1.4610

Add Installer

Add SonarQube Scanner

Save Apply

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

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

Settings / Developer Settings

Some of the scopes you've selected are included in other scopes. Only the minimum set of necessary scopes has been saved.

Personal access tokens (classic)	
<input type="button" value="Generate new token"/>	<input type="button" value="Delete"/>
Tokens you have generated that can be used to access the GitHub API .	
ghp_1HzrdYosTJMRkAuRhqB65dwM1nGaz4zu123	
Make sure to copy your personal access token now. You won't be able to see it again!	
<input type="button" value="Delete"/>	
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.	

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

New credentials

Kind: Username with password

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

Username: msnehabawane

Treat username as secret:

Password:

ID: Git-dred

Description: git

Create

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

SonarQube installations

List of SonarQube installations

Name: Sonar

Server URL: Default is http://localhost:9000
http://3.110.212.101:9000

Server authentication token: SonarQube authentication token. Mandatory when anonymous access is disabled.

Save Apply Advanced

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

Global Trusted Pipeline Libraries

Sharable libraries available to any Pipeline jobs running on this system. These libraries will be trusted, meaning they run without "sandbox" restrictions and may use @Grab.

Library

Name ? Shared

Default version ? main

Cannot validate default version until after saving and reconfiguring.

Load implicitly ?

Allow default version to be overridden ?

Include @Library changes in job recent changes ?

Cache fetched versions on controller for quick retrieval ?

Retrieval method

Dashboard > Manage Jenkins > System >

Retrieval method

Modern SCM

Source Code Management

Git

Project Repository ? https://github.com/msnehabawane/Jenkins_SharedLib.git

Credentials ? - none -

+ Add

Behaviors

Discover branches ?

Save Apply

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

Create Webhook

All fields marked with * are required

Name * Jenkins-Webhook

URL * http://3.110.212.101:8080/sonarqube-webhook

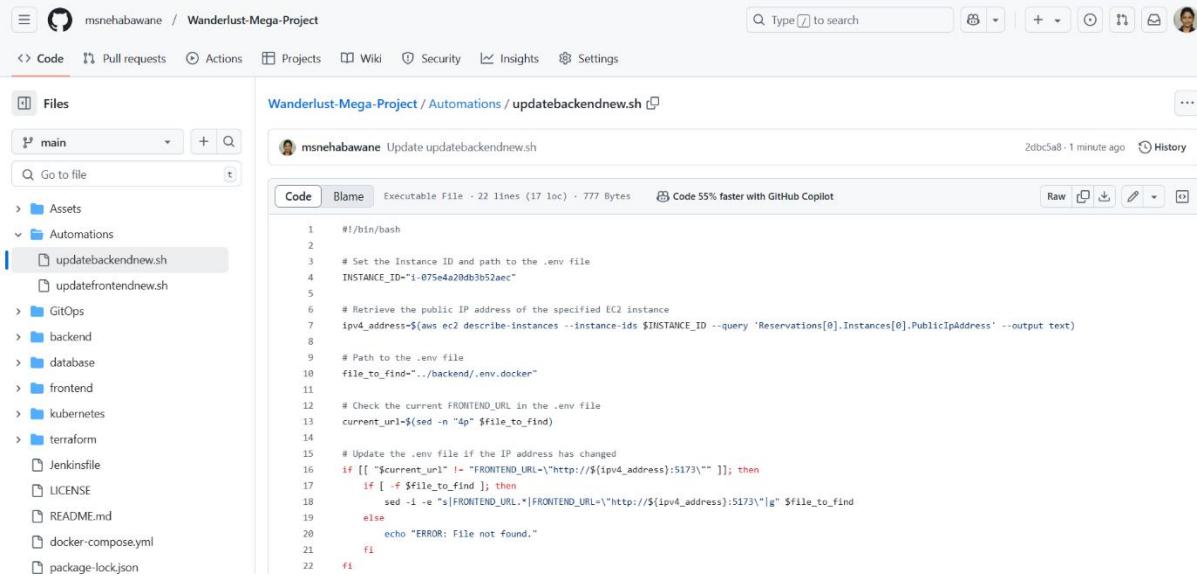
Server endpoint that will receive the webhook payload, for example: "http://my_server/foo". If HTTP Basic authentication is used, HTTPS is recommended to avoid man in the middle attacks. Example: "https://myLogin:myPassword@my_server/foo"

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

- 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



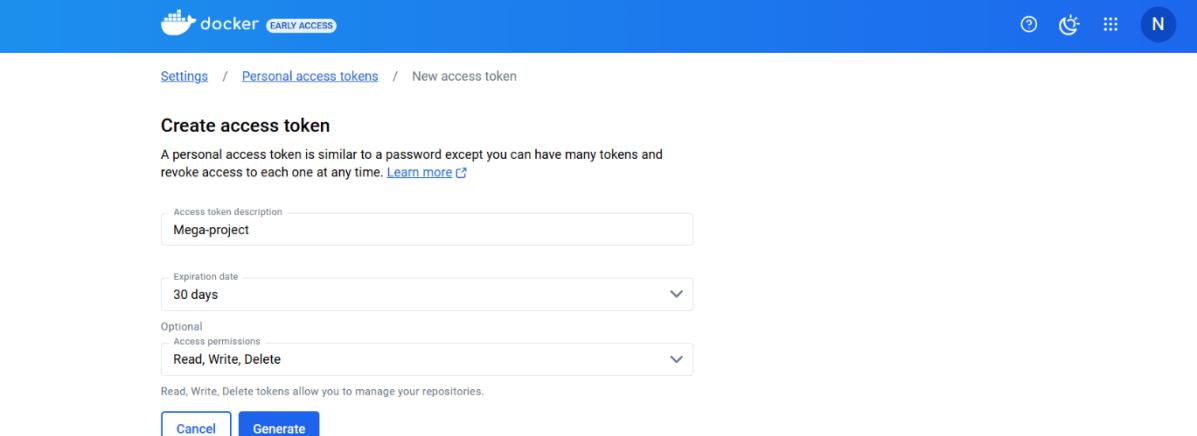
The screenshot shows a GitHub repository named "Wanderlust-Mega-Project". The "Automations" directory contains two files: "updatebackendnew.sh" and "updatefrontendnew.sh". The "updatebackendnew.sh" file is currently open in the editor. The code in the file is as follows:

```

1  #!/bin/bash
2
3  # Set the Instance ID and path to the .env file
4  INSTANCE_ID="i-075e4a20db3b52ae"
5
6  # Retrieve the public IP address of the specified EC2 instance
7  ipv4_address=$(aws ec2 describe-instances --instance-ids $INSTANCE_ID --query 'Reservations[0].Instances[0].PublicIpAddress' --output text)
8
9  # Path to the .env file
10 file_to_find="../backend/.env.docker"
11
12 # Check the current FRONTEND_URL in the .env file
13 current_url=$(sed -n "4p" $file_to_find)
14
15 # Update the .env file if the IP address has changed
16 if [[ "$current_url" != "FRONTEND_URL=http://$(ipv4_address):5173" ]]; then
17     if [ -f $file_to_find ]; then
18         sed -i -e "s|FRONTEND_URL.*|FRONTEND_URL=\"http://$(ipv4_address):5173\"|g" $file_to_find
19     else
20         echo "ERROR: File not found."
21     fi
22 fi

```

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



The screenshot shows the DockerHub settings page for creating a new personal access token. The token is named "Mega-project", has an expiration date of "30 days", and the "Access permissions" are set to "Read, Write, Delete".

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

New credentials

Kind

Username with password

Scope

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

Username

nehabawane

Treat username as secret

Password

ID

docker

Description

docker-hub-cred

Create

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

← App passwords

App passwords help you sign in to your Google Account on older apps and services that don't support modern security standards.

App passwords are less secure than using up-to-date apps and services that use modern security standards. Before you create an app password, you should check to see if your app needs this in order to sign in.

[Learn more](#)

You don't have any app passwords.

To create a new app-specific password, type a name for it below...

App name
Jenkins

Create

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

New credentials

Kind: Username with password

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

Username: neha.sb1302@gmail.com

Password: *****

ID: Gmail

Description: ?

Create

- 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

Extended E-mail Notification

SMTP server: smtp.gmail.com

SMTP Port: 465

Advanced ^ Edited

Credentials: neha.sb1302@gmail.com/***** (Gmail Password)

+ Add

Use SSL
 Use TLS

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>



Your connection is not private

Attackers might be trying to steal your information from 13.201.34.33 (for example, passwords, messages, or credit cards). [Learn more about this warning](#)

NET::ERR_CERT_AUTHORITY_INVALID

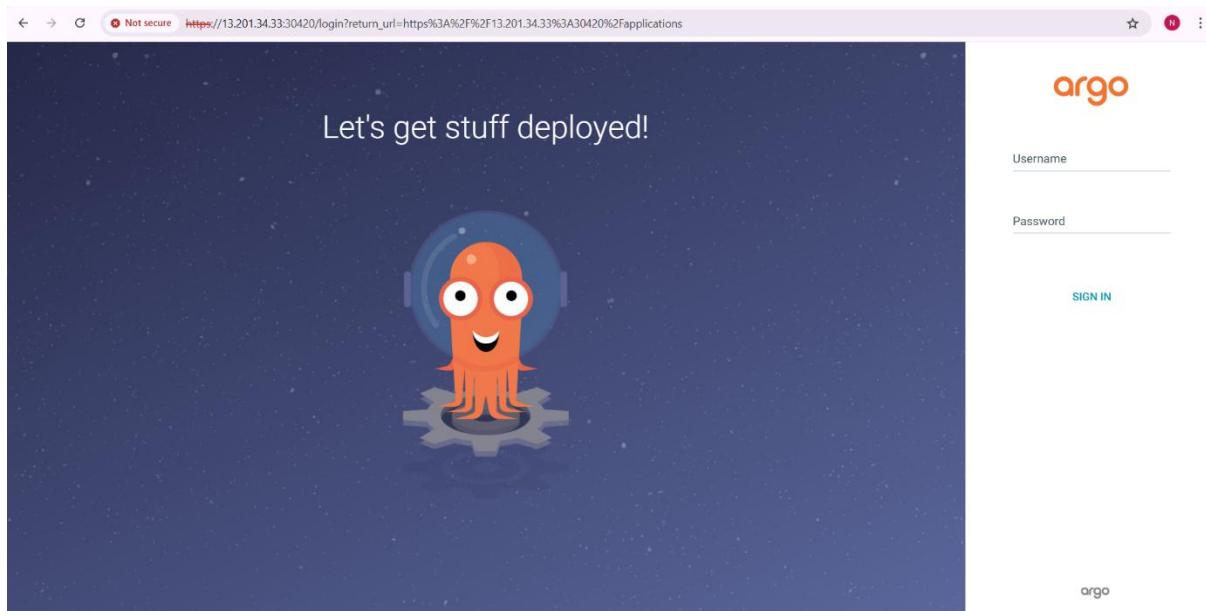
[Turn on enhanced protection](#) to get Chrome's highest level of security

[Hide advanced](#)

[Back to safety](#)

This server could not prove that it is 13.201.34.33; its security certificate is not trusted by your computer's operating system. This may be caused by a misconfiguration or an attacker intercepting your connection.

[Proceed to 13.201.34.33 \(unsafe\)](#)



- 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
```

```
ubuntu@ip-172-31-46-111:~$ kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath=".data.password" | base64 -d; echo
ICqchFifFF8Luqt
ubuntu@ip-172-31-46-111:~$ |
```

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



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

The top screenshot shows the ArgoCD Settings page with various configuration options like Repositories, Clusters, and Projects.

The middle screenshot shows a modal dialog for connecting a repository. It asks for a connection method (VIA HTTPS), sets the type to git, specifies a project (default), and provides a Repository URL (https://github.com/msnehabawane/Wanderlust-Mega-Project.git). A Username field is also present.

The bottom screenshot shows the repositories list after connection. It displays a single entry: a git repository named 'git' under the 'default' project, connected to the specified GitHub URL, with a successful connection status.

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. Proceed 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.
```

- **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:~$
```

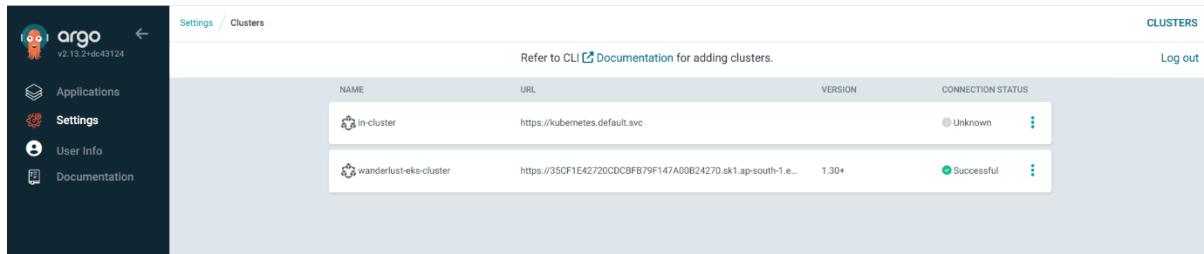
- **Add your cluster to argocd**

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.eksctl.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://35CF1E42720CDCFB79F147A00B24270.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

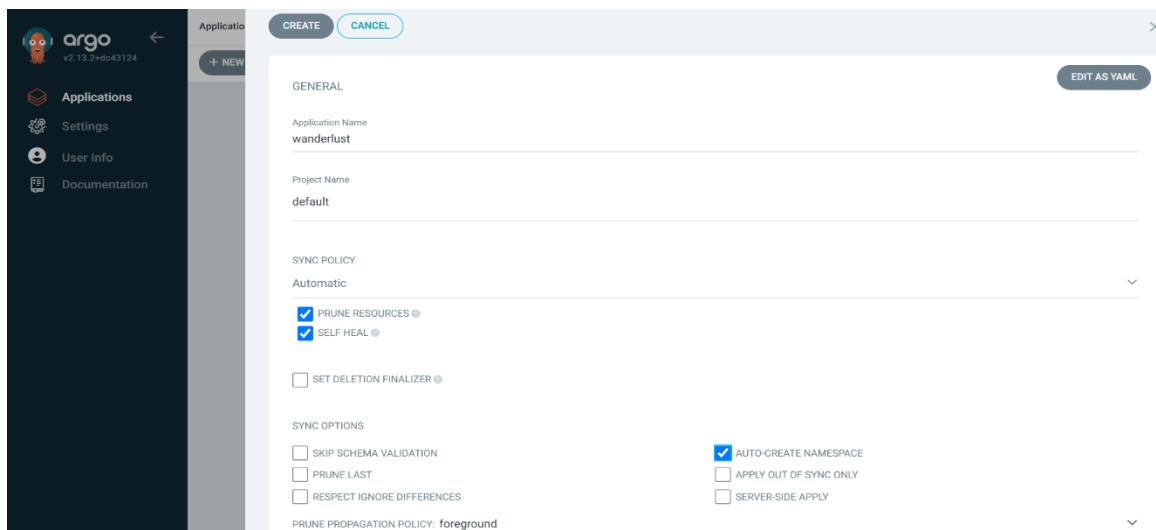


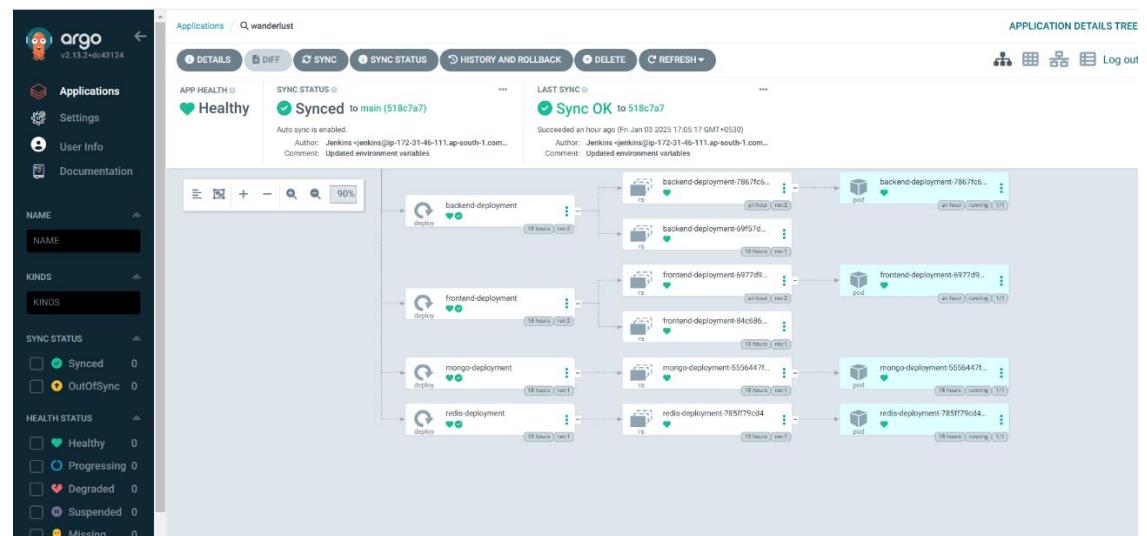
NAME	URL	VERSION	CONNECTION STATUS
in-cluster	https://kubernetes.default.svc	Unknown	⋮
wanderlust-eks-cluster	https://35CF1E42720CDCFB79F147A00B24270.sk1.ap-south-1.eks.amazonaws.com	1.30+	Successful ⋮

Connection should be successful

- Now, go to Applications and click on New App

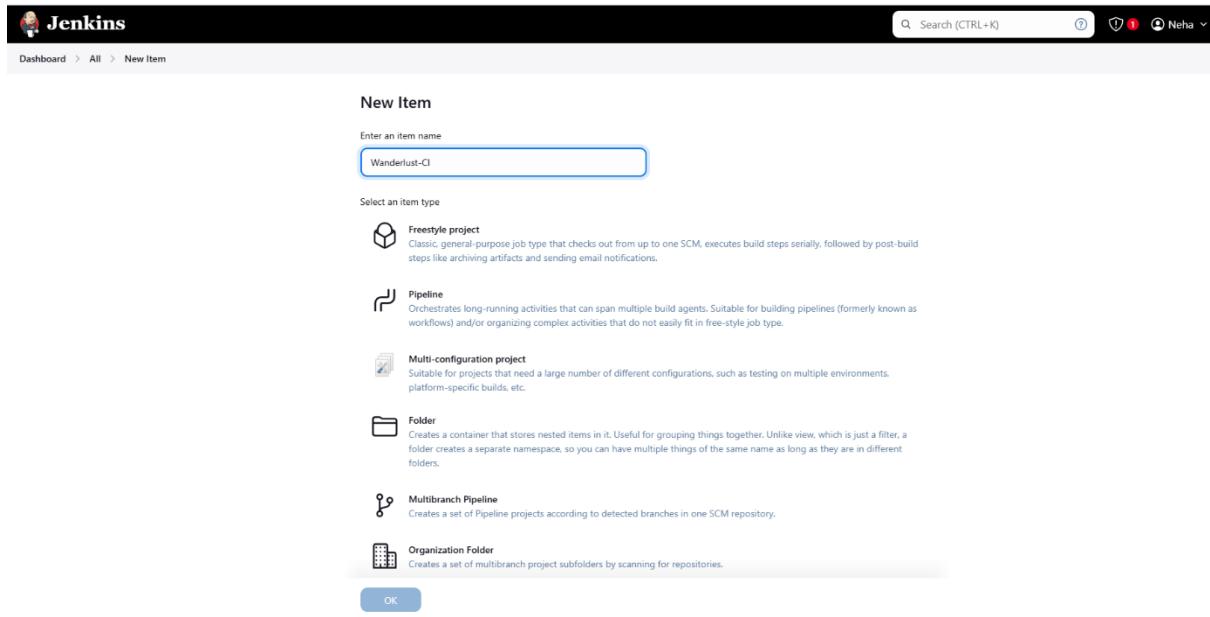
Make sure to click on the Auto-Create Namespace option while creating argocd application





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

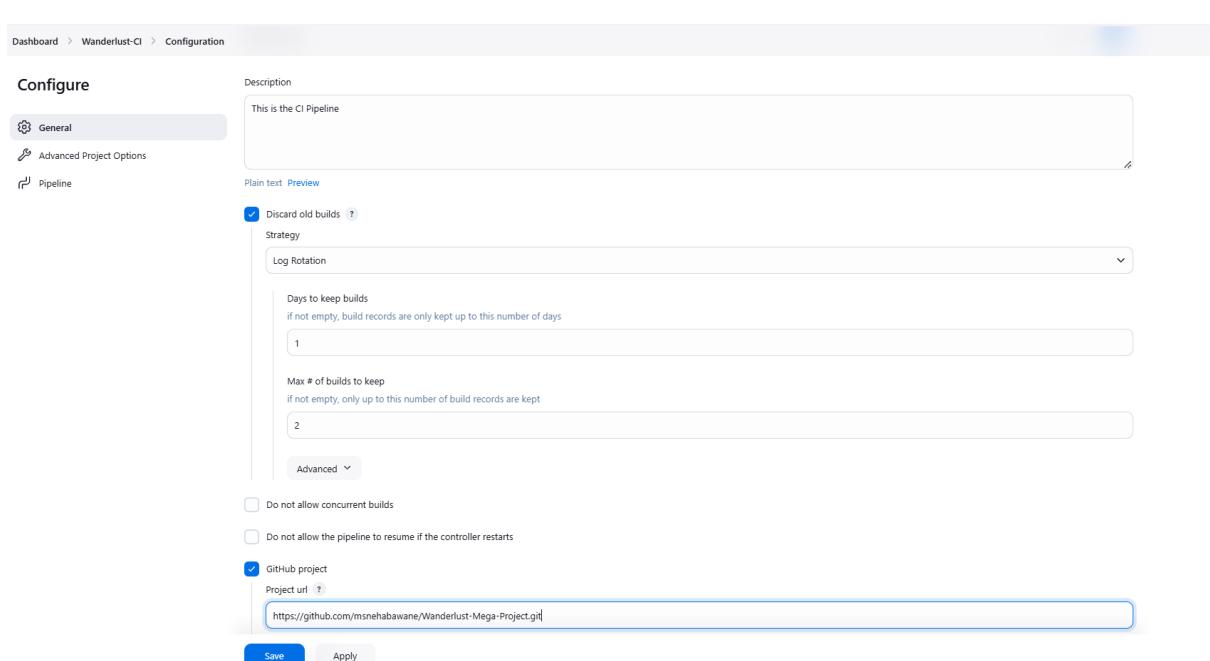
Step 6: Create a Wanderlust CI And CD Pipeline



The screenshot shows the Jenkins 'New Item' creation interface. In the 'Enter an item name' field, 'Wanderlust-CI' is typed. Below it, under 'Select an item type', several options are listed:

- 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.

An 'OK' button is visible at the bottom of the list.



The screenshot shows the 'Configure' screen for the 'Wanderlust-CI' pipeline project. The 'General' tab is selected. The 'Description' field contains the text 'This is the CI Pipeline'. Under the 'Log Rotation' section, the 'Days to keep builds' is set to 1 and the 'Max # of builds to keep' is set to 2. The 'Advanced' tab is partially visible, showing options like 'Do not allow concurrent builds' and 'GitHub project' with the URL 'https://github.com/msnehabawane/Wanderlust-Mega-Project.git' entered.

Dashboard > Wanderlust-CD > Configuration

Configure

- Poll SCM
- Quiet period
- Trigger builds remotely (e.g., from scripts)

General

Advanced Project Options

Pipeline

Advanced

Pipeline

Definition

Pipeline script

```
Script ?  
@library('Shared') ->  
pipeline {  
    agent any;  
    environment{  
        SONAR_HOME = tool "Sonar"  
    }  
    parameters {  
        string(name: 'FRONTEND_DOCKER_TAG', defaultValue: '', description: 'Setting docker image for latest push')  
        string(name: 'BACKEND_DOCKER_TAG', defaultValue: '', description: 'Setting docker image for latest push')  
    }  
    stages {  
        stage("Validate Parameters") {  
            steps {  
                script {  
                    if (params.FRONTEND_DOCKER_TAG == "" || params.BACKEND_DOCKER_TAG == "") {  
                        error "Docker tags are required"  
                    }  
                }  
            }  
        }  
    }  
}
```

Use Groovy Sandbox

Pipeline Syntax

Save Apply

CD Pipeline

Jenkins

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

Dashboard > Wanderlust-CD > Configuration

Configure

General

Description: This is a CD Pipeline

Enabled:

Discard old builds

Strategy: Log Rotation

Days to keep builds: 1

Max # of builds to keep: 2

Advanced

Do not allow concurrent builds

Do not allow the pipeline to resume if the controller restarts

Save Apply

Dashboard > Wanderlust-CD > Configuration

Configure

Advanced Project Options

Advanced ▾

General

Advanced Project Options Selected

Pipeline

Pipeline

Definition

Pipeline script

try sample Pipeline...

```
Script ?  
1  @Library('Shared') -  
2  pipeline {  
3      agent any  
4  
5      parameters {  
6          string(name: 'FRONTEND_DOCKER_TAG', defaultValue: '', description: 'Frontend Docker tag of the image built by the CI job')  
7          string(name: 'BACKEND_DOCKER_TAG', defaultValue: '', description: 'Backend Docker tag of the image built by the CI job')  
8      }  
9  
10     stages {  
11         stage("Workspace cleanup"){  
12             steps{  
13                 script{  
14                     cleanWs()  
15                 }  
16             }  
17         }  
18     }  
19 }
```

Use Groovy Sandbox ?

Save Apply



Jenkins

Dashboard >

New Item Build History Manage Jenkins My Views Open Blue Ocean

All +

S	W	Name ↴	Last Success	Last Failure	Last Duration	F
...	...	Wanderlust-CD	N/A	N/A	N/A	▶ ⭐
...	...	Wanderlust-CI	N/A	N/A	N/A	▶ ⭐

Build Queue: No builds in the queue.

Build Executor Status: 0/2 ▾

Icon: S M L

Search (CTRL+K)    Neha 

00:00

Wanderlust-CI Pipeline Output

Wanderlust-Cl

This is the CI Pipeline

Last Successful Artifacts

dependency-check-report.html 110.95 KB

Dependency-Check Trend

Stage View

Validate Parameters	Workspace cleanup	Git Code Checkout	Trivy: filesystem scan	OWASP: Dependency check	SonarQube: Code Analysis	SonarQube: Code Quality Gates	Exporting environment variables	Backend env setup	Frontend env setup	Docker: Build Images	Docker: Push to DockerHub	Declarative Post Actions
258ms	301ms	1s	3s	13min 30s	14s	508ms	137ms	2s	2s	1min 22s	25s	19s
Average stage times: (Average full run time: ~4min 28s)												
Jan 19 10:28	10 changes											
329ms	313ms	1s	774ms	8s	12s	416ms (passed for 20)	145ms	1s	1s	2min 43s	50s	19s
269ms	289ms	1s	7s	26min 53s	16s	600ms (passed for 10)	130ms	2s	2s	716ms (failed)	213ms (failed)	

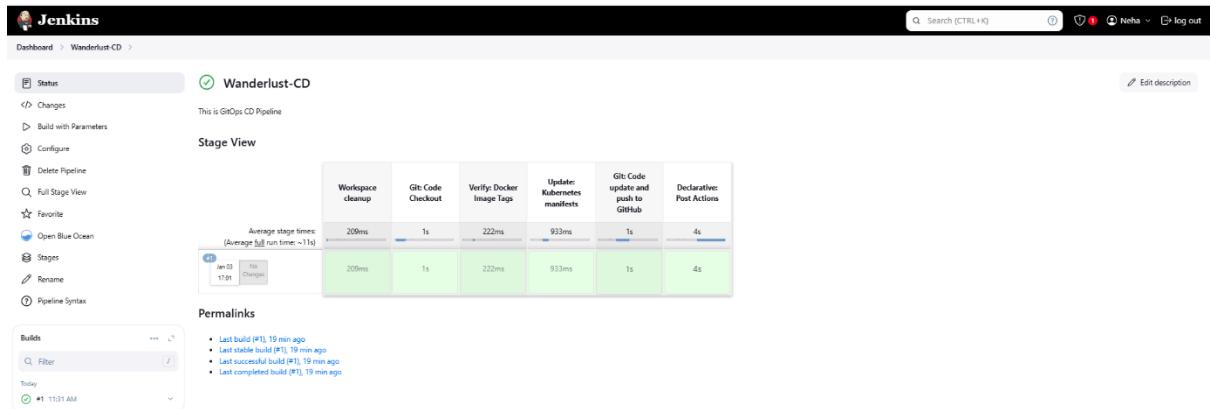
SonarQube Quality Gate

wanderlust: Passed

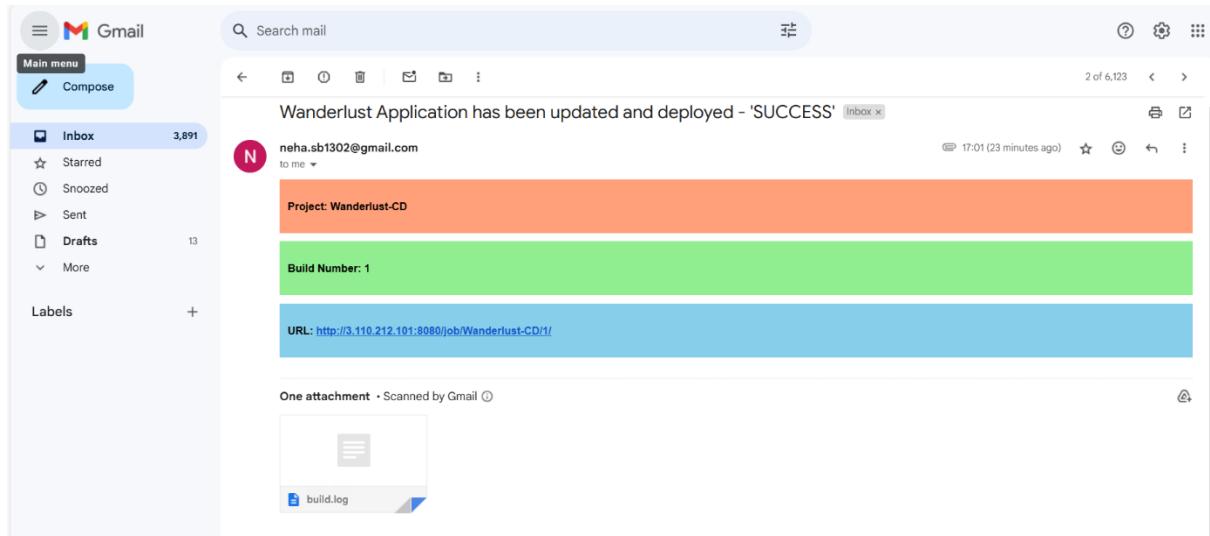
server-side processing: Success

Latest Dependency-Check

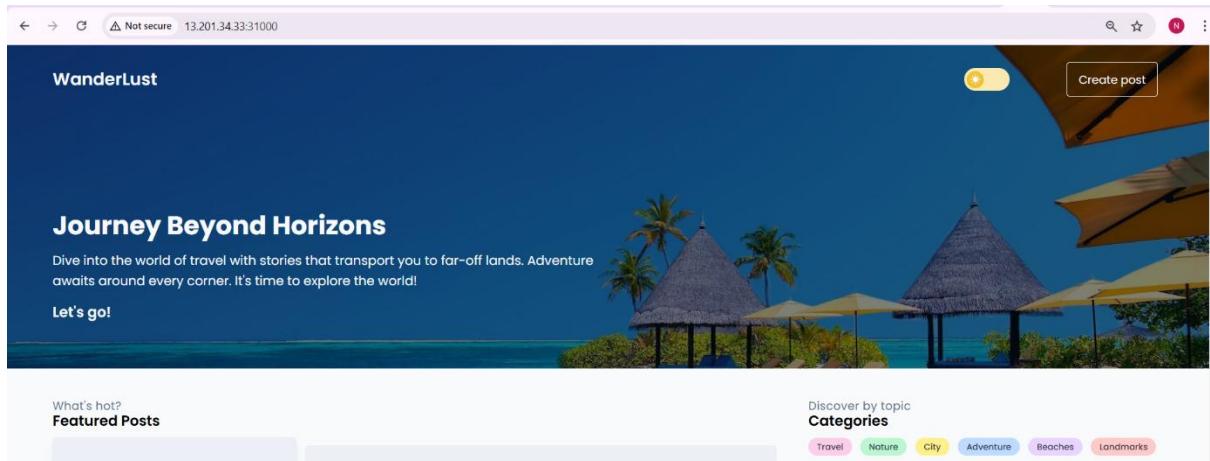
Wanderlust-CD Pipeline Output



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

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

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

The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with links for 'Prometheus', 'Query' (which is active), 'Alerts', and 'Status'. Below the navigation is a search bar with placeholder text 'Enter expression (press Shift+Enter for newlines)'. Underneath the search bar are three tabs: 'Table' (selected), 'Graph', and 'Explain'. A small note below the tabs says 'No data queried yet'. At the bottom left is a button labeled '+ Add 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: af7bbcccd-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  10.100.206.41  <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>

The image consists of three vertically stacked screenshots of the Grafana web application.

Top Screenshot (Login Screen):

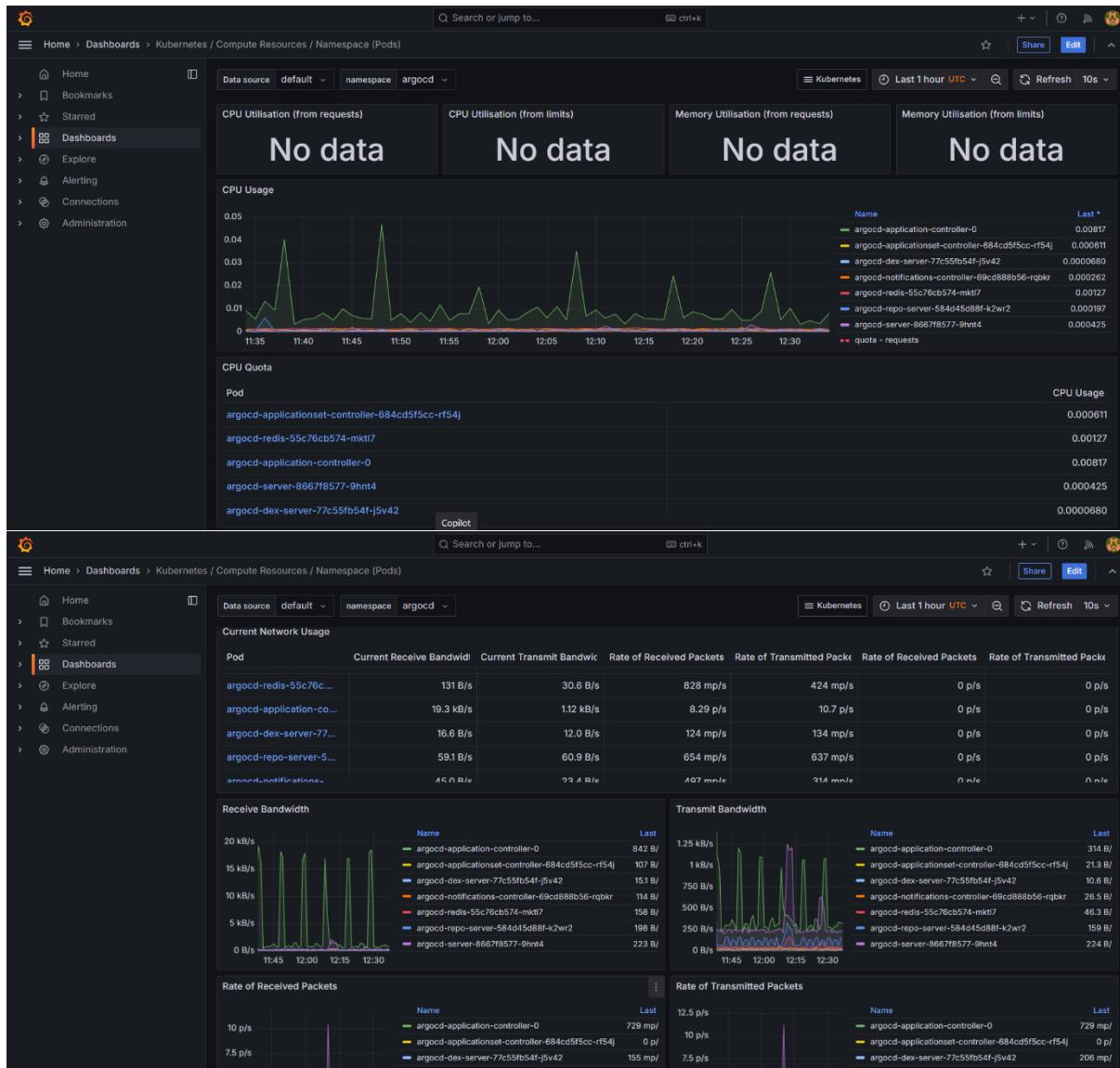
A dark-themed login page with a central logo. The text "Welcome to Grafana" is displayed above a form with fields for "Email or username" and "Password". A blue "Log in" button is at the bottom, and a link "Forgot your password?" is below it. The address bar at the top shows the URL `13.201.34.33:31032/login`.

Middle Screenshot (Home Dashboard):

A dark-themed dashboard titled "Welcome to Grafana". It features a "Basic" section with instructions for setting up Grafana. To the right, there are three cards: "TUTORIAL DATA SOURCE AND DASHBOARDS" (Grafana fundamentals), "COMPLETE Add your first data source", and "COMPLETE Create your first dashboard". Below these are sections for "Dashboards" (Starred dashboards, Recently viewed dashboards) and "Latest from the blog" (an article about CERN using Grafana and Mimir). The address bar shows `13.201.34.33:31032`.

Bottom Screenshot (Dashboards List):

A dark-themed list of dashboards. The left sidebar shows navigation links: Home, Bookmarks, Starred, Dashboards (which is selected and highlighted in orange), Explore, Alerting, Connections, and Administration. The main area lists 19 dashboards under the heading "Dashboards". Each entry includes a thumbnail, the dashboard name, and its associated tags. For example, "Alertmanager / Overview" has the tag "alertmanager-mixin", and "CoreDNS" has the tags "coredns" and "dns". The address bar shows `13.201.34.33:31032/dashboards`.



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