

Node Application CI/CD Pipeline with Jenkins

What is CI/CD?

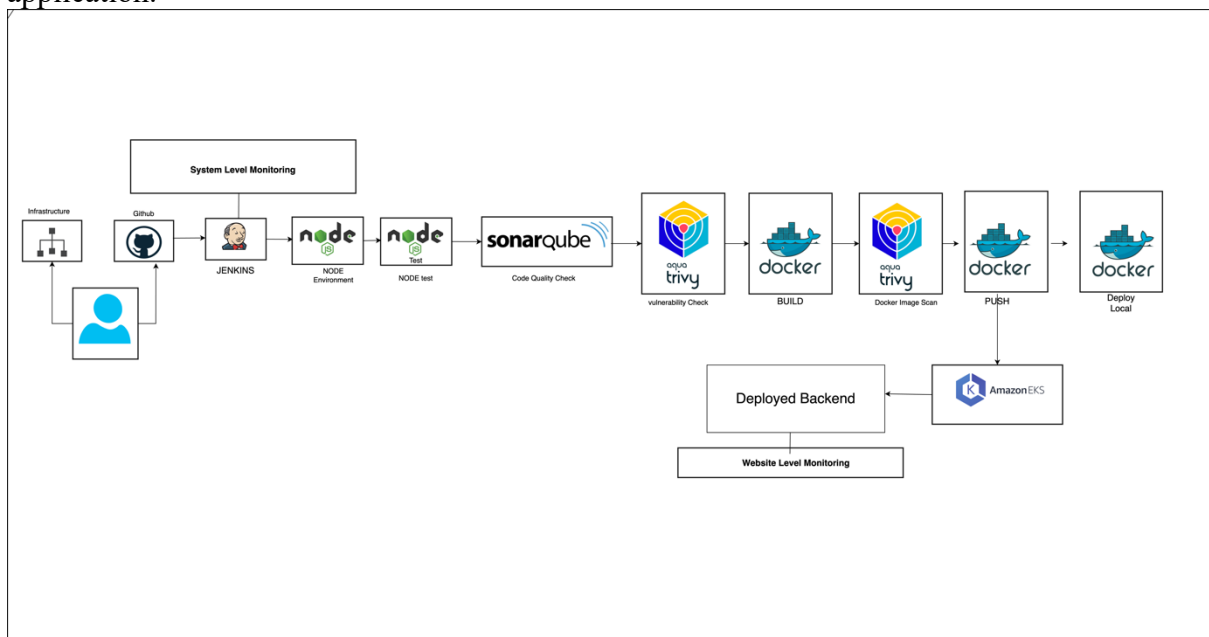
CI/CD stands for Continuous Integration and Continuous Deployment (or Continuous Delivery). It is a set of practices and tools used to automate the processes of software integration, testing, and deployment. This helps in ensuring that software can be reliably released at any time.

Continuous integration (CI) refers to the practice of automatically and frequently integrating code changes into a shared source code repository.

Continuous delivery and/or deployment (CD) is a 2-part process that refers to the integration, testing, and delivery of code changes. Continuous delivery stops short of automatic production deployment, while continuous deployment automatically releases the updates into the production environment.

Architecture Diagram

This is the full architectural diagram for the CI/CD pipeline for the deployment of a Node application.



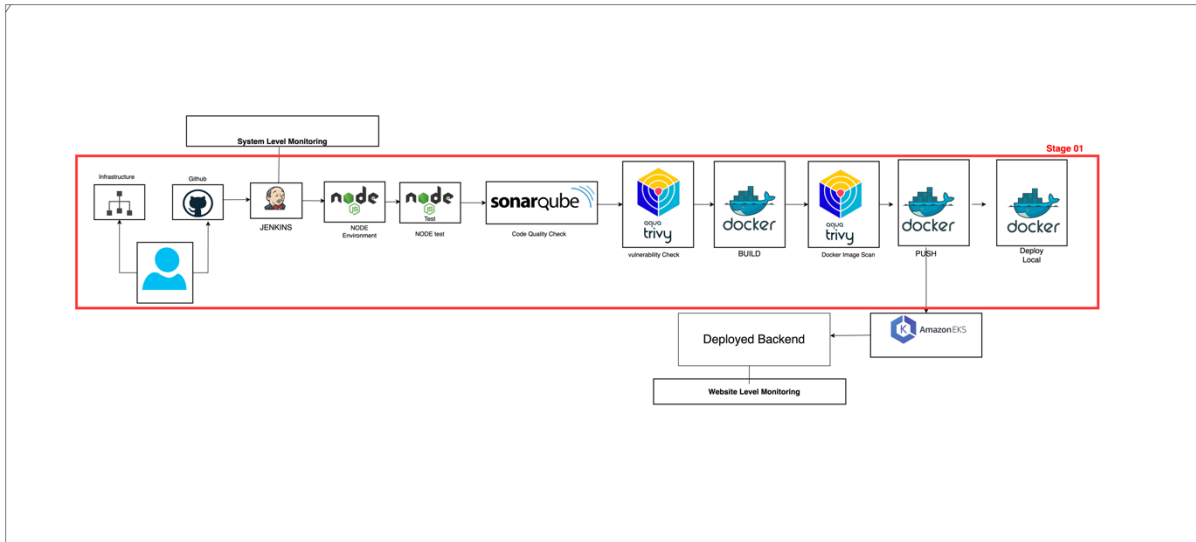
Explain Tool Used for pipeline

- Jenkins: Automation server to orchestrate the CI/CD pipeline.
- GitHub: Version control system for managing the source code.
- Node.js: JavaScript runtime environment for executing code.
- SonarQube: Tool for continuous inspection of code quality.
- Aqua Trivy: Security scanner for vulnerabilities in filesystem and Docker images.

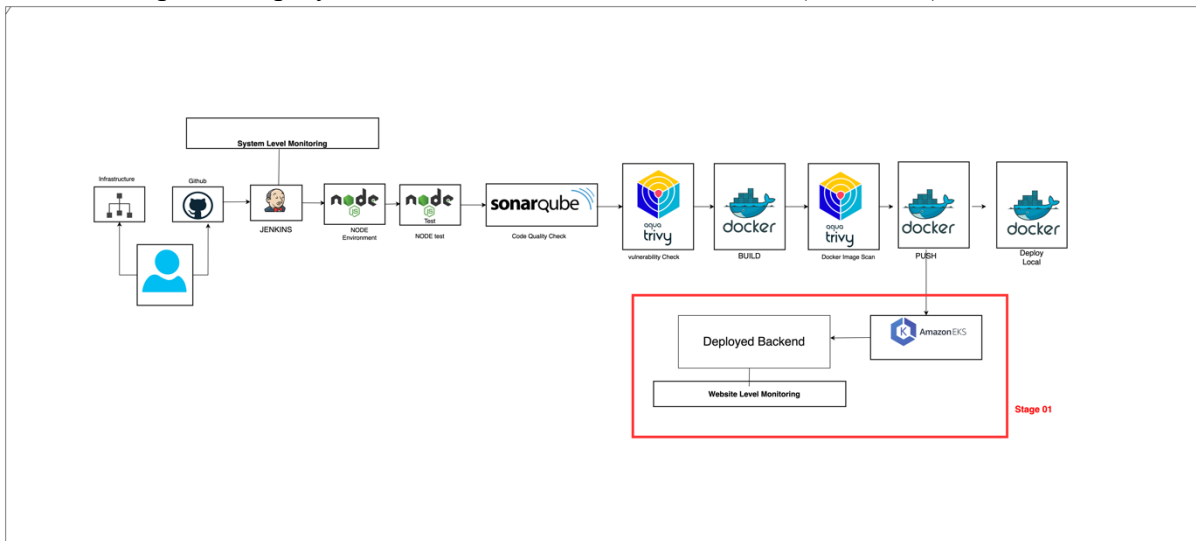
- Docker: Containerization platform for building and running applications.
- Amazon EKS: Managed Kubernetes service for deploying, managing, and scaling containerized applications.

We plan to setup the CI/CD pipeline in two phases, the entire pipeline

1. Completed up to docker deployment. (Deployment to development environment)



2. Completed deployment on Elastic Kubernetes Service (Aws EKS).



Explain Pipeline Stages

- **Git Checkout:** Jenkins pulls the latest code from the GitHub repository.
- **Node.js Environment Setup:** Jenkins sets up the Node.js environment.
- **Testing:** Runs Node.js tests to ensure code functionality.
- **Code Quality Check:** SonarQube performs a code quality analysis.
- **Filesystem Scan:** Aqua Trivy scans the filesystem for vulnerabilities.
- **Docker Build:** Builds the Docker image for the Node.js application.
- **Docker Image Scan:** Aqua Trivy scans the Docker image for vulnerabilities.
- **Push Docker Image:** The Docker image is pushed to the Docker registry(Hub).
- **Deploy to Local Environment:** The Docker image is deployed to the Amazon EKS cluster.
- **Deploy to Amazon EKS:** The Docker image is deployed to the Amazon EKS cluster.

Implement Stage 01

Setup Security Group

First, you have to setup Security Group in the AWS.

Go EC2 Dashboard->Security Group->Give Name and Description (CICDSecurityGrp)->Go inbound

Then setup inbound security policies as below

Custom	TCP	30000-32767	VM's for K8's for deployment
Custom	TCP	6443	K8's API Server Traffic
SSH	TCP	22	For SSH
HTTPS	TCP	443	For incoming web traffic(secure)
HTTP	TCP	80	For incoming web traffic
Custom	TCP	85/443	Application Deployment
Custom	TCP	8080	Jenkins

Then Goto Outbound Traffic Security policies as this,

All Traffic	ALL	ALL	
-------------	-----	-----	--

IMDSv2
Optional
⚠️ EC2 recommends setting IMDSv2 to required |
[Learn more](#)

Instance ARN
arn:aws:ec2:us-east-1:851725423168:instance/i-028155b3e6eda22df

Details | Status and alarms | Monitoring | **Security** | Networking | Storage | Tags

▼ Instance details Info

Platform
Amazon Linux (Inferred)

Platform details
Linux/UNIX

AMI ID
ami-0a1179631ec8933d7

AMI name
amzn2-ami-kernel-5.10-hvm-2.0.20240412.0-x86_64-gp2

IMDSv2
Optional
⚠️ EC2 recommends setting IMDSv2 to required |
[Learn more](#)

subnet-0f0d4e0b255c39610

Instance ARN
arn:aws:ec2:us-east-1:851725423168:instance/i-028155b3e6eda22df

Details | Status and alarms | Monitoring | **Security** | Networking | Storage | Tags

▼ Security details

IAM Role
-

Security groups
sg-0d40a42d12f580680 (launch-wizard-2)

Owner ID
851725423168

Launch time
Wed Jul 17 2024 13:54:31 GMT+0530 (India Standard Time)

Inbound rules | Outbound rules | Tags

Inbound rules (3)

Q Search

Manage tags Edit inbound rules

	Name	Security group rule...	IP version	Type	Protocol	Port range
<input type="checkbox"/>	-	sgr-08b0342db2c8401bf	IPv4	HTTP	TCP	80
<input type="checkbox"/>	-	sgr-008b58b45fef07002	IPv4	Custom TCP	TCP	8080
<input type="checkbox"/>	-	sgr-04df39925a2d025...	IPv4	SSH	TCP	22

Setup Virtual Machines (VM's)

AWS EC2 instances used for VM's. Create one VM and its name as Jenkins.

Specification of the VM

- **Instance type:** t2 large (We require additional CPU and Memory because we are running Jenkins, SonarQube, Docker, and Aqua Trivy on the same machine.)
- **AMI:** Ubuntu (22.04 LTS)
- **Storage:** 20GB
- **Security Group:** Select Before Created Security Group (CICDSecurityGrp)

Setup Jenkins

- Connect to the VM using its public IP address with PuTTY or any other method.
- Next update the package system in ubuntu system Use Command
sudo apt update
- Next Install java
sudo apt install openjdk-17-jre-headless
- After installed java next install Jenkins
**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
sudo apt-get install Jenkins
- Start Jenkins (Follow below steps to start Jenkins)
 - You can enable the Jenkins service to start at boot with the command:
sudo systemctl enable jenkins
 - You can start the Jenkins service with the command:
sudo systemctl start jenkins
 - You can check the status of the Jenkins service using the command:
sudo systemctl status jenkins
 - After access Jenkins URL (Vm's public url (IP)with 8080 port)
Grab the password from bellow location
"/var/lib/jenkins/secrets/initialAdminPassword"
Or use '**sudo systemctl status Jenkins**' password

```

Last login: Fri Apr 21 07:23:27 2023 from 157.45.215.185
ubuntu@ip-172-31-34-193:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
   Active: activating (start) since Fri 2023-04-21 09:07:35 UTC; 39s ago
     Main PID: 445 (java)
       Tasks: 40 (limit: 1141)
      Memory: 274.4M
         CPU: 22.102s
    CGroup: /system.slice/jenkins.service
            └─445 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: Joadac7b06c6f4fe28b314bbe2af251ff
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: *****
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: *****
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: *****
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: WARNING: An illegal reflective access operation has occurred
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: WARNING: Illegal reflective access by org.codehaus.groovy.vmplugin.v7.Java7$1 (file:/var/cache/jenkins
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: WARNING: Please consider reporting this to the maintainers of org.codehaus.groovy.vmplugin.v7.Java7$1
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
Apr 21 09:08:01 ip-172-31-34-193 jenkins[445]: WARNING: All illegal access operations will be denied in a future release
lines 1-20/20 (END)

```

Install suggested plugins.

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

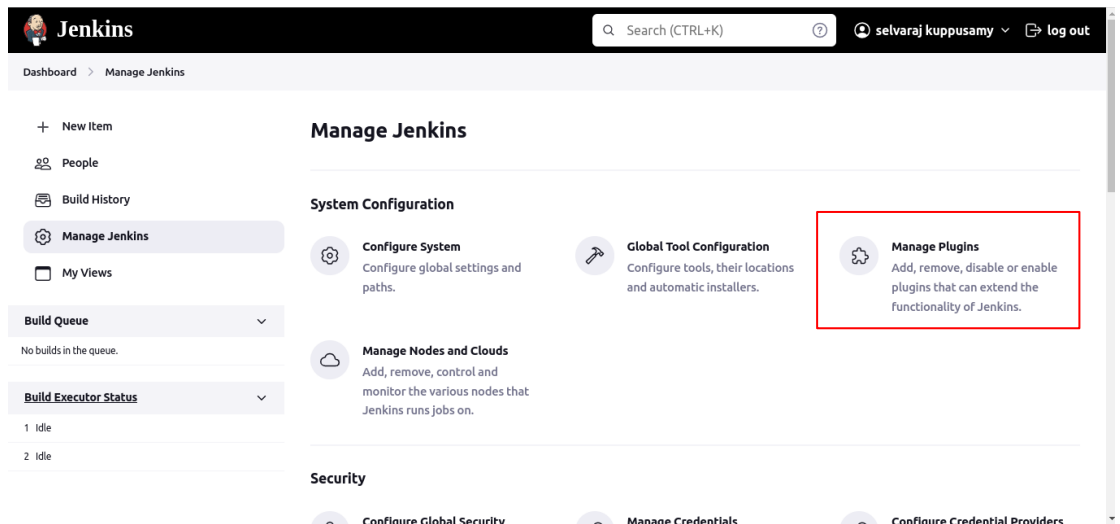
Select plugins to install

Select and install plugins most suitable for your needs.

Jenkins 2.129-SNAPSHOT

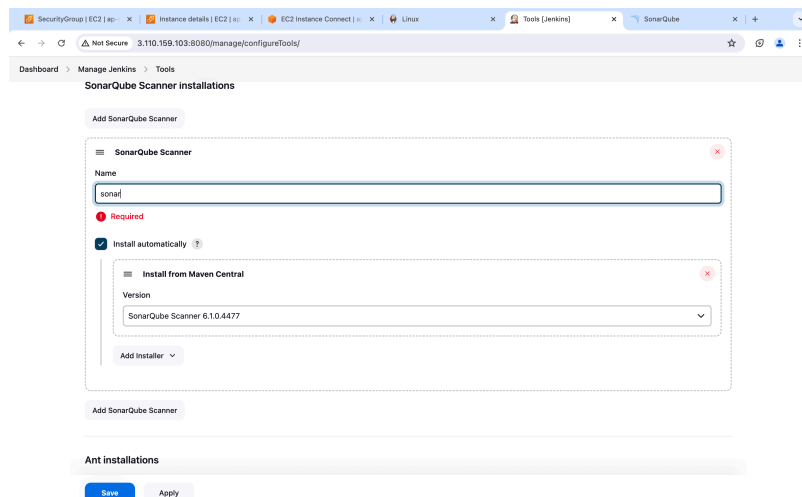
Setup Credential and start.

- Download and install required plugins (Dashboard->Manage Jenkins->Manage Plugins->Available plugins)

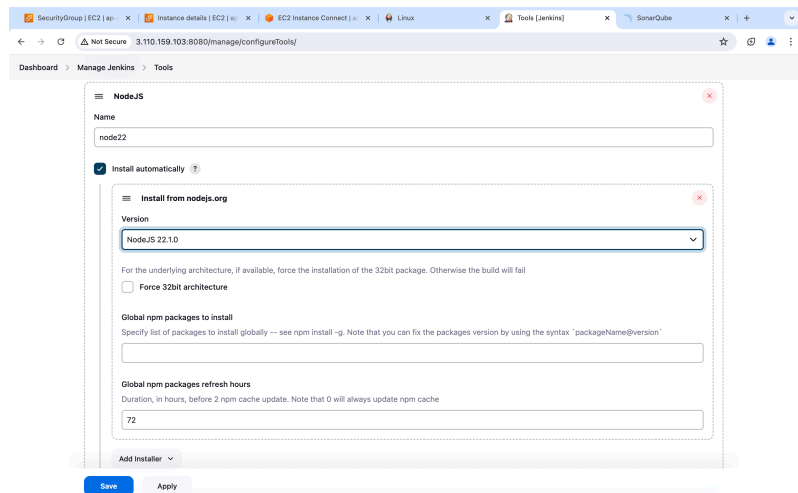


- Select below plugins and download
 - NodeJS
 - SonarQube Scanner
 - Docker
 - Docker Pipeline
 - Kubernetes
 - Kubernetes CLI

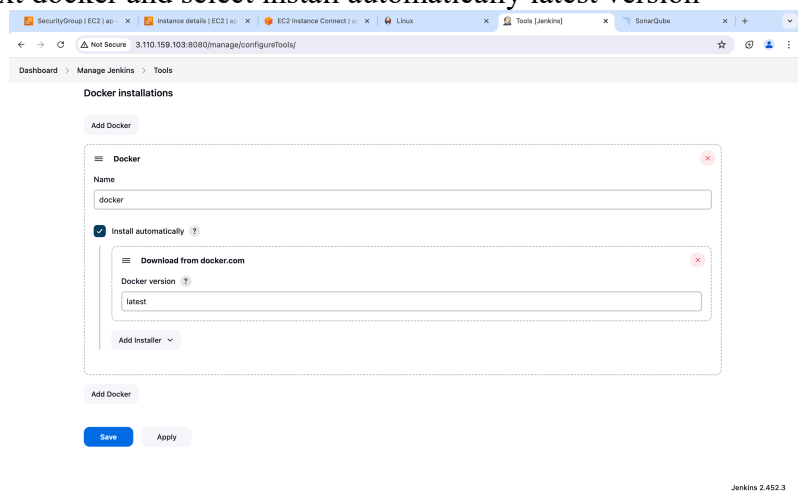
- Configure Tools (Dashboard->Manage Jenkins->Tools)
 - Select SonarQube Scanner and set name and select latest version.



- Next NodeJS and give name and select required version



- Next docker and select install automatically latest version



Setup Docker

- In Jenkins VM install docker
sudo apt install docker.io
- Next check the docker working without any error pull hello-world
docker pull hello-world
- If founded any permission error run this command
sudo chmod 666 /var/run/docker.sock
- Now check again with “**docker pull hello-world**” and now working
- Then go to Jenkins and navigate to Credentials (Dashboard -> Manage Jenkins -> Manage Credentials).
- After, adding the credential, use 'Username with password' as the kind and provide the username and password of your Docker Hub account.

Setup SonarQube

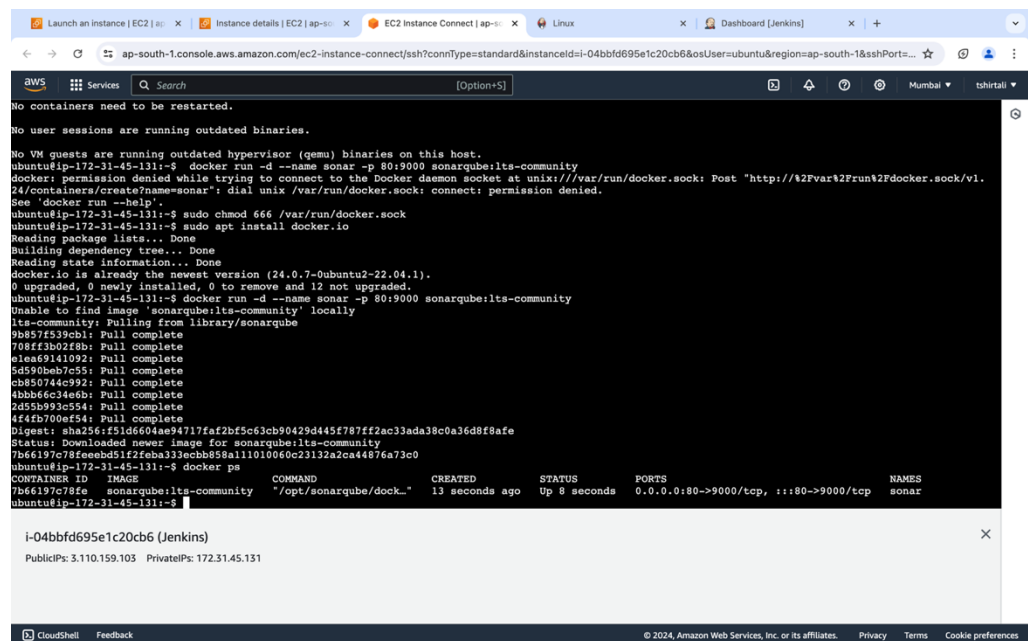
- Run SonarQube using docker image

docker run -d --name sonar -p 80:9000 sonarqube:lts-community

runs a SonarQube container in detached mode, names it "sonar", and maps port 9000 of the container to port 80 on the host machine, using the sonarqube:lts-community image.

- After Completed image pull check image is started using command

docker ps



```
Launch an instance | EC2 | ap-south-1 | Instance details | EC2 | ap-south-1 | EC2 Instance Connect | ap-south-1 | Linux | Dashboard | Jenkins | +
ap-south-1.console.aws.amazon.com/ec2-instance-connect/ssh?connType=standard&instanceId=i-04bbfd695e1c20cb6&osUser=ubuntu&region=ap-south-1&sshPort=22
No containers need to be restarted.
No user sessions are running outdated binaries.
No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-45-131:~$ docker run -d --name sonar -p 80:9000 sonarqube:lts-community
docker: permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post "http://%2Fvar%2Frun%2Fdocker.sock/v1.24/containers/create?name=sonar": dial unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
ubuntu@ip-172-31-45-131:~$ sudo chmod 666 /var/run/docker.sock
ubuntu@ip-172-31-45-131:~$ sudo apt install docker.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker.io is already the newest version (24.0.7-0ubuntu2-22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 12 not upgraded.
ubuntu@ip-172-31-45-131:~$ docker run -d --name sonar -p 80:9000 sonarqube:lts-community
Unable to find image 'sonarqube:lts-community' locally
lts-community: Pulling from library/sonarqube
9b857f539cb1: Pull complete
708ff3b2f2fb: Pull complete
61ea69141092: Pull complete
5d590beb7c55: Pull complete
cb850744c392: Pull complete
4bb66c34e6b: Pull complete
2d5b993c554: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:f51d6604ae94717faf2bf5c63cb90429d445f787ff2ac33ada38c0a36d8f8afe
Status: Downloaded newer image for sonarqube:lts-community
7b66197c78fe: Pull complete
ubuntu@ip-172-31-45-131:~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
7b66197c78fe   sonarqube:lts-community             "/opt/sonarqube/dock..." 13 seconds ago Up 8 seconds   0.0.0.0:80->9000/tcp, :::80->9000/tcp sonar
ubuntu@ip-172-31-45-131:~$
```

i-04bbfd695e1c20cb6 (Jenkins)

PublicIP: 3.110.159.103 PrivateIP: 172.31.45.131

CloudShell Feedback © 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

- After pinging the VM public URL on port 80, the SonarQube dashboard will load. It will prompt you for a username and password. The default username is 'admin' and the default password is 'admin'. After entering the default credentials, the dashboard will load and prompt you to set up new credentials. Once set up, you can use Jenkins.
- Integrated with Jenkins

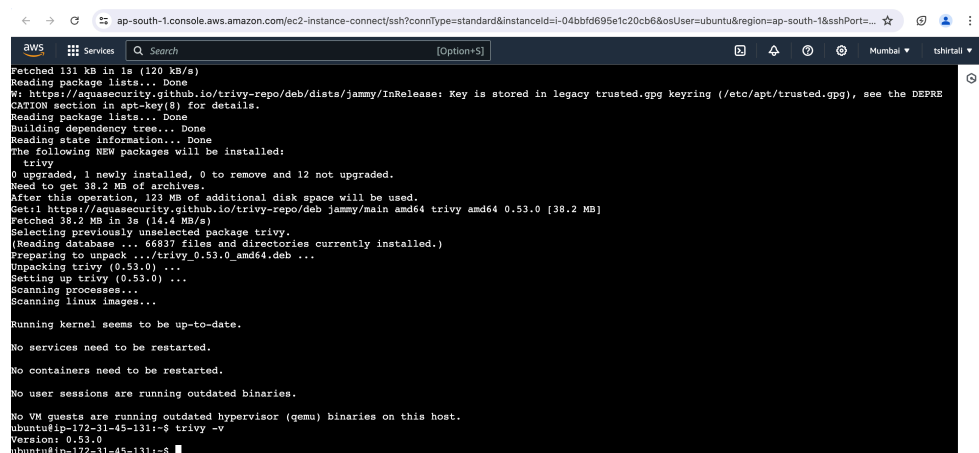
SonarQube Dashboard -> Administration -> Security -> User -> Tokens -> Enter name and generate token and copy it.

Load Jenkins -> Manage Jenkins -> Credential -> global -> add Credentials -> kind select secrete text -> set Secrete that generate in SonarQube -> give id and description

Setup Aqua Trivy

- To install Aqua Trivy on the Jenkins VM

```
sudo apt-get install wget apt-transport-https gnupg lsb-release
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo
apt-key add -
echo deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc)
main | sudo tee -a /etc/apt/sources.list.d/trivy.list
sudo apt-get update
sudo apt-get install trivy
```
- Check the Trivy version after installation:



```
ap-south-1.console.aws.amazon.com/ec2-instance-connect?connType=standard&instanceId=i-04bbfd695e1c20cb6&osUser=ubuntu&region=ap-south-1&sshPort=...
AWS
Services
Search
[Option+S]
Mumbai
tshitali
Fetched 131 kB in 1s (120 kB/s)
Reading package lists... Done
W: https://aquasecurity.github.io/trivy-repo/deb/dists/jammy/InRelease: Key is stored in legacy trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRE
CATION section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  trivy
0 upgraded, 1 newly installed, 0 to remove and 12 not upgraded.
Need to get 38.2 MB of archives.
After this operation, 123 MB of additional disk space will be used.
Get:1 https://aquasecurity.github.io/trivy-repo/deb jammy/main amd64 trivy amd64 0.53.0 [38.2 MB]
Fetched 38.2 MB in 3s (14.4 MB/s)
Selecting previously unselected package trivy.
(Reading database ... 66837 files and directories currently installed.)
Preparing to unpack .../trivy_0.53.0_amd64.deb ...
Unpacking trivy (0.53.0) ...
Setting up trivy (0.53.0) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-45-131:~$ trivy -v
Version: 0.53.0
ubuntu@ip-172-31-45-131:~$
```

Setup Git

Integrate with Jenkins:

- Go to your GitHub account: Developer settings -> Tokens (Classic).
- Give a name for the token and select the scopes you need.
- Generate the token and copy it.
- Add the token to Jenkins Credentials as a 'Username with password' credential. Use your GitHub username as the username and the generated token as the password.

Setup web hooks

- Go to your GitHub repository: Settings -> Webhooks -> Add webhook.
- Set the Payload URL to `http://jenkinsurl:8080/github-webhook/`.
- Set the Content type to `application/json`.
- Save the webhook.

All configuration for the first stage is done. Now we need to create the pipeline.

Build Pipeline

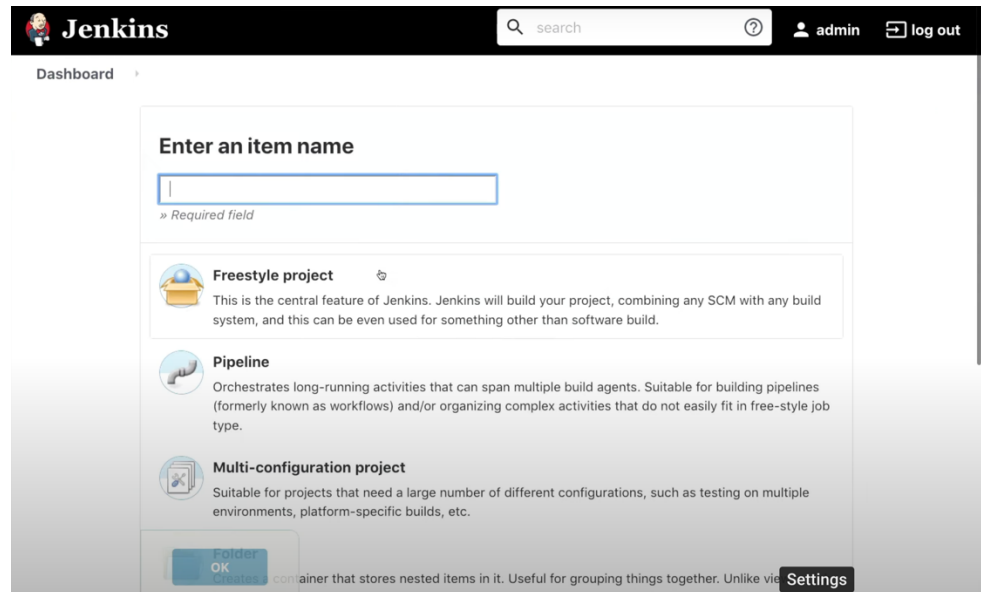
- Open Jenkins Dashboard: Navigate to your Jenkins dashboard in your web browser.
- Create New Item: For **Backend Pipeline** (Node application)

Click on "New Item" on the left sidebar.

Enter a name for your pipeline (e.g., NodeJS-CICD-Pipeline).

Select "Pipeline" from the list of options.

Click "OK"



- Under the "Pipeline" section, select "Pipeline script".

Use this script [Link](#)

- Copy and paste it into Pipeline script.
- Next step Save and Build pipeline.
- Create New Item: For **Frontend Pipeline** (React application)

Click on "New Item" on the left sidebar.

Enter a name for your pipeline (e.g., ReactFrontend-CICD-Pipeline).

Select "Pipeline" from the list of options.

Click "OK"

- In general -> Discard old builds -> Max # of builds to keep set as 2(best practice)
- Next, Build Triggers->tick to GitHub hook trigger for GITScm polling

- Under the "Pipeline" section, select "Pipeline script".

Use this script [Link](#)

Stage 01 completed.

Implement Stage 02

Setting Up Amazon EKS and Jenkins Pipeline for Deployment

This guide will walk you through setting up Amazon EKS, configuring a Kubernetes environment, and integrating it with a Jenkins pipeline for continuous deployment.

Step 1: Create an IAM User in AWS and Attach Policies

This stage ensures that your EKS cluster has the necessary permissions to perform various operations. The specific policies and custom policies provide the permissions required to manage EC2 instances, EKS clusters, CloudFormation stacks, and IAM resources.

1. **Sign in to the AWS Management Console.**
2. **Navigate to the IAM service.**
3. **Create an IAM User:**
 - Click on "Users" in the sidebar, then click "Add user."
 - Enter a user name (e.g., `eks-user`).
 - Select "Programmatic access" for Access type.
 - Click "Next: Permissions."
4. **Attach Managed Policies:**
 - Select "Attach existing policies directly."
 - Attach the following policies:
 - `AmazonEC2FullAccess`
 - `AmazonEKS_CNI_Policy`
 - `AmazonEKSClusterPolicy`
 - `AmazonEKSWorkerNodePolicy`
 - `AWSCloudFormationFullAccess`
 - `IAMFullAccess`
5. **Create and Attach Custom Policy:**
 - Click "Create policy."
 - Select the "JSON" tab and paste the following policy content:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": "eks:*",
      "Resource": "*"
    }
  ]
}
```

- Click "Review policy."
- Enter a name for the policy (e.g., `EKSFullAccessPolicy`).
- Click "Create policy."

- Go back to the user creation wizard, click "Refresh" to see the new policy, and attach it.
- 6. Finish User Creation:**
 - Click "Next: Tags" (optional).
 - Click "Next: Review."
 - Click "Create user."
 - Note down the Access key ID and Secret access key.

Step 2: Install AWS CLI, kubectl, and eksctl

1. Install AWS CLI:

This tool allows you to interact with AWS services from your command line, enabling you to automate various AWS operations.

```
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
```

- Enter the Access key ID and Secret access key.
- Set the default region (e.g., ap-south-1).

2. Install kubectl:

This is the Kubernetes command-line tool that allows you to run commands against Kubernetes clusters. You use kubectl to deploy applications, inspect and manage cluster resources, and view logs.

```
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
```

3. Install eksctl:

This is a command-line utility for creating and managing EKS clusters. It simplifies many of the tasks involved in setting up and maintaining Kubernetes clusters on AWS.

```
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
```

Step 3: Create an EKS Cluster

1. Create the Cluster:

This initializes your Kubernetes cluster within EKS, specifying details such as the cluster name, region, zones, and Kubernetes version

```
eksctl create cluster --name=eks1 --region=ap-south-1 --zones=ap-south-1a,ap-south-1b --version=1.30 --without-nodegroup
```

2. Associate IAM OIDC Provider:

This step associates an OpenID Connect (OIDC) provider with your EKS cluster, enabling IAM roles to be used by Kubernetes service accounts.

```
eksctl utils associate-iam-oidc-provider --region ap-south-1 --cluster eks1 --approve
```

3. Create Node Group:

This creates a group of worker nodes (EC2 instances) that will run your Kubernetes workloads. Various parameters, such as instance type, number of nodes, volume size, and access policies, are specified.

```
eksctl create nodegroup --cluster=eks1 --region=ap-south-1 --name=node2 --node-type=t3.medium --nodes=3 --nodes-min=2 --nodes-max=4 --node-volume-size=20 --ssh-access --ssh-public-key=ci/cd --managed --asg-access --external-dns-access --full-ecr-access --appmesh-access --alb-ingress-access
```

Step 4: Configure Kubernetes

1. Create a Namespace:

This step creates a Kubernetes namespace for organizing and isolating resources within the cluster.

```
kubectl create namespace webapps
```

2. Create Service Account:

This defines a service account in the Kubernetes namespace, which can be used by Jenkins to interact with the cluster.

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

3. Create Role:

These steps define and apply role-based access control (RBAC) policies. They grant specific permissions to the service account, allowing it to perform actions like creating and managing pods, services, and other resources within the namespace.

```
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", "nodes", "persistentvolumes", "persistentvolumeclaims",
    "resourcequotas", "replicasets", "replicationcontrollers",
    "serviceaccounts", "services"]
    verbs: ["get", "list", "watch", "create", "update", "patch",
    "delete"]
```

4. Bind the Role to the Service Account:

```
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:
  - kind: ServiceAccount
    name: jenkins
    namespace: webapps
```

5. Generate Token Using Service Account:

This step generates an access token for the service account, which will be used by Jenkins to authenticate and interact with the Kubernetes cluster.

```
yaml
Copy code
apiVersion: v1
kind: Secret
type: kubernetes.io/service-account-token
metadata:
  name: mysecretname
  annotations:
    kubernetes.io/service-account.name: Jenkins

kubectl describe secret mysecretname -n webapps
```

- Copy the secret token.
- 6. **Integrate with Jenkins:**
 - Go to Jenkins -> Credentials -> give name as 'k8-token' -> Kind -> Secret text -> Paste the token in the secret -> Save.
- 7. Configure Pipeline:

Go to backend and frontend builds and configure and edit script and add below stages

Link for Script:- [Script](#)

```

1      stage('Deploy to Kubernetes') {
2          steps {
3              dir('backend') {
4                  script {
5                      withKubeCredentials(kubeCtlCredentials: [[caCertificate: '', clusterName: 'my-EKS-1', contextName: '', credentialsId: 'k8-token', namespace: 'webapps', serverUrl: 'https://2311BF0A95
6                      sh "kubectl delete pods --all"
7                      sh "kubectl apply -f deployment.yml"
8                      }
9                  }
10             }
11         }
12     }
13
14     stage('Verify K8s Service') {
15         steps {
16
17             script {
18                 withKubeCredentials(kubeCtlCredentials: [[caCertificate: '', clusterName: 'my-EKS-1', contextName: '', credentialsId: 'k8-token', namespace: 'webapps', serverUrl: 'https://2311BF0A95
19                 sh "kubectl get svc"
20                 }
21             }
22         }
23     }
24 }

```

1. Give name of when you created EKS cluster name.
2. Give name of you given credential name of you configured in Jenkins.
3. Give name of you created namespace.
4. Give endpoint URL of your EKS.

- **Sign in to the AWS Management Console.**
- **Navigate to the EKS Service:**
 - In the AWS Management Console, type "EKS" in the search bar and select "Elastic Kubernetes Service" from the dropdown.
- **Select Your Cluster:**
 - In the EKS console, click on the name of your cluster (e.g., eks1).
- **Find the Cluster Endpoint:**
 - On the cluster details page, you will find the "Endpoint" URL listed under the "Cluster API server endpoint" section.

Save pipeline configuration -> Build Pipeline again.

Deployment.yml files also in repo

Access k8s pods

Kubectl get pods -n webapps

Accesss k8s services

Kubectl get svc -n webapps

Using External ip address can access application

Access k8s pods

Kubectl get pods -n webapps

Accesss k8s services

Kubectl get svc -n webapps

Using External ip address can access application

The screenshot shows the AWS CloudShell interface. The terminal output displays the command `kubectl get svc -n webapps` and its results:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
backend-service	LoadBalancer	10.100.78.252	a4f9d6193e8674a3596360ef190a9b4b-1767497541.ap-south-1.elb.amazonaws.com	85:32417/TCP	26h
frontend-service	LoadBalancer	10.100.147.19	a097e0ad9492247498d17df08548c9dc-1710398779.ap-south-1.elb.amazonaws.com	443:31898/TCP	26h

A text box highlights the URL and port: "Using this URL and port :443 (URL:443) can access application."

Below the terminal, a box shows the instance details for `i-0d8dbbeb4e947b101 (Jenkins)`:
PublicIPs: 13.235.242.149 PrivateIPs: 172.31.32.42

The terminal also shows the command `kubectl get pods -n webapps` and its results:

NAME	READY	STATUS	RESTARTS	AGE
backend-deployment-7886678fcb-shhvs	1/1	Running	0	24h
frontend-deployment-5db777f999-s9p8c	1/1	Running	0	24h

- If you refresh console or something that configuration of aws cli kubectl removed run this command again

Aws configure

aws eks --region ap-south-1 update-kubeconfig --name nodekubernetes

- **Delete Cluster After done all things**

eksctl delete cluster --name=nodekubernetes --region=ap-south-1