**End to End Node.js Web Application Deployment on K8s (AWS EKS /Local K8s Cluster) with Jenkins**

**🧪 Project-1: Node.js Web Application Deployment on Docker with Jenkins**

**Node.js Web Application Deployment on Docker with Jenkins** का project structure दिया है, मैं आपको same-to-same lab setup बनाकर दूँगा step-by-step, ताकि आप सीधे अपने test server पर try कर सको।

ये project 3 parts में होगा:

* 1. **Node.js App Setup on Server**
  2. **Dockerize the Node.js App**
  3. **Jenkins Pipeline for CI/CD**

**1️⃣ Project Directory Structure**

<https://github.com/htyagi2233/nodejs-docker-project.git>

nodejs-docker-project

├── package.json

├── package-lock.json

├── public

│   ├── images

│   │   └── nodejs-logo.png

│   ├── index.html

│   └── styles.css

├── README.md

├── src

│   ├── app.js

│   └── imageDisplay.js

└── tests

├── imageDisplay.test.js

├── package.json

└── package-lock.json

................................................... Installaion process on test server ................................................................................................................

* 1. Go to NodeJs Officially Site -> <https://nodejs.org/en/download>
  2. And install Node.js on test Server

Machine generated alternative text:
Download Node.js@ 
using V nvm 
with 
npm 
bash 
Copy to clipboard 
Get Node.js@ v22.19.o (LTS) 
for Linux 
Info Want new features sooner? Get the latest version instead and to' the latest improvements! 
1 
2 
3 
4 
5 
6 
7 
8 
9 
18 
11 
12 
13 
14 
15 
Bash 
# Download and install nvm: 
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v@.4@.3/insta11.sh 
# in lieu of restarting the shell 
X. "$HOME/ .nvm/nvm.sh" 
# Download and install Node .js: 
nvm install 22 
# Verify the Node. js version: 
node -v # Should print "v22.19.@" . 
nvm current # Should print "v22.19.@" 
# Verify npm version: 
npm -v # Should print "1@.9.3" 
"nvm" is a cross-platform Node.js version manager. If you encounter any issues please visit nvm•s website 7 
Or get a prebuilt Node.js@for 
Windows Installer (.msi) 
• Windows 
running a x64 
architecture. 
Standalone Binary (lip) 

* 1. **Copy first command and run on test Server-**

curl -o- <https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.3/install.sh> | bash

* 1. **Command install होने के बार output में हमें 3 command मिलेंगी अब हमें इस 3 command को run करना है test server पर**

export NVM\_DIR="$HOME/.nvm"

[ -s "$NVM\_DIR/nvm.sh" ] && \. "$NVM\_DIR/nvm.sh" # This loads nvm

[ -s "$NVM\_DIR/bash\_completion" ] && \. "$NVM\_DIR/bash\_completion" # This loads nvm bash\_completion

# in lieu of restarting the shell

\. "$HOME/.nvm/nvm.sh"

# Download and install Node.js:

nvm install 22

# Verify the Node.js version:

node -v # Should print "v22.19.0".

nvm current # Should print "v22.19.0".

# Verify npm version:

npm -v # Should print "10.9.3".

* 1. **This file is available in GitHub Repository**

<https://github.com/htyagi2233/nodejs-docker-project.git>

# cat package.json

{

"name": "my-nodejs-spa",

"version": "1.0.0",

"description": "A Node.js SPA project with Jest tests",

"main": "src/app.js",

"scripts": {

"start": "node src/app.js",

"test": "jest"

},

"dependencies": {

"express": "^4.19.2"

},

"devDependencies": {

"jest": "^27.5.1"

},

"directories": {

"test": "tests"

},

"keywords": [],

"author": "",

"license": "ISC"

}

* 1. **So based on this file we will installed dependency**

# npm install express

# npm install --save-dev jest

# npm start

* 1. **Copy the Jenkins Server ip and browse with 3000 port**

Server is running on <http://0.0.0.0:3000>

**Project- 2 Git Checkout stage**

**This project we will do on a jenkins inside a docker container** ……………………………………….….

**Step-1**

**🎯 Project Objective:**

* 1. Jenkins को Docker container में चलाना
  2. GitHub से Node.js Project को Jenkins pipeline के ज़रिए checkout करना
  3. Jenkins में Git integration और pipeline configuration को समझना
  4. GitHub token का उपयोग करके secure authentication करना
  5. Jenkins में Build Discarder जैसे options का इस्तेमाल करना

* 1. **Project Directory Structure on my local server**

<https://github.com/htyagi2233/nodejs-docker-project.git>

nodejs-docker-project

├── package.json

├── package-lock.json

├── public

│   ├── images

│   │   └── nodejs-logo.png

│   ├── index.html

│   └── styles.css

├── README.md

├── src

│   ├── app.js

│   └── imageDisplay.js

└── tests

├── imageDisplay.test.js

├── package.json

└── package-lock.json

* 1. **Create GitHub Repo**
     1. GitHub में login करो → **New Repository** पर click करो।
     2. Example: nodejs-docker-project नाम से repo बना लो।
     3. Repo public/~~private~~ कोई भी रख सकते हो।

<https://github.com/htyagi2233/nodejs-docker-project.git>

* 1. **Create Token in GitHub (Personal Access Token - PAT)**

क्योंकि GitHub अब password-based authentication allow नहीं करता push के लिए।

* 1. GitHub → Profile → **Settings**
  2. Developer Settings → **Personal Access Tokens** → Tokens (classic) → **Generate new token**
  3. Scope select करो → कम से कम repo (for repo access)
  4. Token generate होने के बाद copy करके कहीं सुरक्षित रख लो।

* 1. **Connect Repo to Jenkins Server (Vmware)**
     1. Jenkins server में login करो।
     2. git client installed होना चाहिए (apt install git -y Ubuntu पर)।
     3. Repo को test करने के लिए Jenkins server पर run करो:  
          
        git clone <https://github.com/USERNAME/nodejs-docker-project.git>
     4. जब password मांगेगा → username = आपका GitHub username और password = **token**।

* 1. **Push Code into GitHub using Username + Token**

मान लो आपने local में project बनाया है nodejs-docker-project/

cd nodejs-docker-project  
git init  
git add .  
git commit -m "Initial commit"  
git remote add origin <https://github.com/USERNAME/nodejs-docker-project.git>  
git push <https://USERNAME:TOKEN@github.com/USERNAME/nodejs-docker-project.git> main

💡 यहाँ TOKEN वही होगा जो GitHub से generate किया था।

* 1. **Now Create the Jenkins on docker container**

docker run -d --name jenkins -p 8080:8080 -v jenkins\_home:/var/jenkins\_home jenkins/jenkins

* 1. **Check Running Jenkins Container**

पहले देखो आपका Jenkins container चल रहा है या नहीं:

docker ps

Output में कुछ ऐसा दिखेगा:

CONTAINER ID IMAGE COMMAND PORTS NAMES  
abcd1234efgh jenkins/jenkins ... 0.0.0.0:8080->8080/tcp my-jenkins

* 1. **Login into Jenkins Container**

Container में enter करने के लिए:

docker exec -it my-jenkins bash

👉 यहाँ my-jenkins आपका container का name/ID होगा।

* 1. **Get Initial Admin Password (First Time Login)**

अगर Jenkins पहली बार run किया है तो आपको admin password चाहिए।

Container के अंदर ये command चलाओ:

cat /var/jenkins\_home/secrets/initialAdminPassword

👉 यह long random string होगा → इसे copy करके ब्राउज़र में Jenkins UI पर paste करना होगा।

* 1. **Jenkins Web Login**
     1. **Browser में खोलो:**[**http://localhost:8080**](http://localhost:8080)
     2. **Username: admin (default)**
     3. **Password: ऊपर वाला initialAdminPassword**

* 1. **Install Jenkins Plugins**
  + Jenkins Dashboard → **Manage Jenkins → Plugins → Available Plugins**
  + Install करो:
    - **Pipeline** (अगर पहले से install नहीं है)
    - **Git Plugin**
    - **GitHub Plugin**

* + **Jenkins में Job Create करना**
    - Jenkins Dashboard → **New Item**
    - Job Name: Node-js-project
    - Type: **Pipeline** चुनो → OK

* + Node-js-project -> pipeline
    - Add option -> build discarder
    - Git Checkout in Stage

* + **Pipeline Script**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

}

}

* 1. **Build the Job**
     + **Jenkins Dashboard → Node-js-project पर click करें**
     + **Left side → Build Now पर click करें**

* 1. **Console Output Check करें**
     + **Build Number (#1, #2 etc.) पर click करें**
     + **फिर Console Output खोलें**

👉 यहाँ आपको ये steps दिखने चाहिए:

* + **Started by user admin** (या आपका user)
  + Jenkins workspace set करेगा
  + Git plugin → आपके repo <https://github.com/htyagi2233/nodejs-docker-project.git> से checkout करेगा
  + Branch → \*/main fetch होगी

अगर सब सही है तो output में कुछ ऐसा दिखेगा:

Cloning the remote Git repository  
Cloning repository <https://github.com/htyagi2233/nodejs-docker-project.git>  
 > git init ...  
 > git fetch ...  
 > git checkout main

* + **Build Discarder Verify करें**
    - **Dashboard → Job → Build History में देखो**
    - **अगर आपने कई बार build किया है, तो केवल last 2 builds ही दिखेंगी (बाकी Jenkins ने auto delete कर दी होंगी, क्योंकि buildDiscarder option लगाया है)।**

**Project-3: Node Build stage**

* 1. **Add nodejs -v stage in pipeline and build the job, here we got the error**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

}

}

}

}

**❌ Error Explanation**

/var/jenkins\_home/workspace/node-js-project@tmp/durable-bf128a2d/script.sh.copy: 1: node: not found

**✅ Solution – Install NodeJS in Jenkins**

इसको solve करने के लिए आपके पास 2 तरीके हैं:

**🔹 NodeJS Plugin in Jenkins**

* 1. **Jenkins Dashboard → Manage Jenkins → Plugins → Available plugins**
     + **Install NodeJS Plugin**

* 1. **Configure NodeJS in Jenkins**
     + **Manage Jenkins → Tools**
     + **Scroll down to NodeJS installations**
     + **Add → Name: Nodejs**
     + **Check ✅ "Install automatically"**
     + **Save**

* 1. **अब pipeline में tools section add करें:**

👉 अब Jenkins पहले NodeJS tool install करेगा और फिर वो path में add करके node command run कर पाएगा।

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

}

}

}

}

**Build the Job**

* + Jenkins Dashboard → Node-js-project पर click करें
  + Left side → **Build Now** पर click करें

**Build Now → Console Output**

अगर सबकुछ सही है तो आपको output मिलेगा:

[Pipeline] sh  
+ node -v  
v22.19.0

👉 इसका मतलब है Jenkins ने NodeJS plugin से Node और npm को PATH में load कर लिया है।

---------------------------------------------------------------------------------------------------------------

**⚠️ Common Issues**

* + अगर फिर भी node: not found आता है →  
    🔹 Check करो कि pipeline में tools { nodejs 'Nodejs' } लिखा है और नाम सही है।  
    🔹 Plugin install करने के बाद Jenkins restart करना मत भूलना।

**Project-4:**

**Now check npm install and npm version**

**This time we will check npm version and npm install**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

}

}

**Build Now → Console Output**

अगर सबकुछ सही है तो आपको output मिलेगा:

[Pipeline] sh  
+ node -v  
v22.19.0

[Pipeline] sh  
+ npm -v  
10.9.3

👉 इसका मतलब है Jenkins ने NodeJS plugin से Node और npm को PATH में load कर लिया है।

**Project-5: Dockerfile for NodeJS**

**🐳 Node.js App Dockerfile Setup**

**1️⃣ Create Dockerfile**

Project directory (nodejs-docker-project/) में Dockerfile बनाइये 👇

# Use official Node.js base image  
FROM node:latest

# Create app directory inside container  
WORKDIR /usr/src/app

# Copy package.json and package-lock.json  
COPY package\*.json ./

# Install app dependencies  
RUN npm install

# Copy application source code  
COPY . .

# Expose app port  
EXPOSE 3000

# Start the app  
CMD ["npm", "start"]

**Step- Build the Docker Image and verify docker version for testing using jenkins pipeline**

* + Above Dockerfile will use to create image and push into DockerHub using Jenkins Pipeline and that image will use to deploy application in k8s

**✅ Build the Docker Image for NodeJS**→ Source code से Dockerfile का उपयोग करके Node.js ऐप की Docker image तैयार करना।

docker build -t jenkins/jenkins .

**✅ Create a DockerHub Repo - htyagi2233/jenkins-nodejs**→ DockerHub पर एक remote repository बनाना जहाँ image को push किया जाएगा।

- htyagi2233/jenkins-nodejs

**✅ Jenkins Pipeline**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

}

}

}

}

**❌ Error Explanation**

+ docker -v  
/var/jenkins\_home/workspace/node-js-project@tmp/durable-487b5ad6/script.sh.copy: 1: docker: not found

classic **Docker-in-Docker** (DinD) issue है।

Jenkins container में Docker CLI तो install हो सकती है, लेकिन उसे चलाने के लिए actual Docker daemon चाहिए — जो container में नहीं चल सकती।

**Project-6:**

**Step-✅ Solution**

* + We got this error because docker can nor install inside the docker container
  + So we will create a Dockerfile and build an image

* + Jenkins (jenkins/jenkins:jdk21) बेस इमेज है
  + Docker CLI, plugins आदि install हो रहे हैं
  + Jenkins user को docker group में add किया गया है

# vi Dockerfile

FROM jenkins/jenkins:jdk21

USER root

# Install dependencies

RUN apt-get update && \

apt-get install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# Add Docker’s official GPG key

RUN install -m 0755 -d /etc/apt/keyrings && \

curl -fsSL <https://download.docker.com/linux/debian/gpg> | gpg --dearmor -o /etc/apt/keyrings/docker.gpg && \

chmod a+r /etc/apt/keyrings/docker.gpg

# Add Docker repo for Debian (not Ubuntu)

RUN echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] <https://download.docker.com/linux/debian> \

bookworm stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker

RUN apt-get update && \

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

apt-get clean

# Add Jenkins user to Docker group (safe)

RUN getent group docker || groupadd docker && usermod -aG docker jenkins

USER jenkins

**✅ अब क्या हो पाएगा?**

* + Jenkins pipeline में docker build, docker push, docker images, etc. commands चल पाएंगी।
  + Docker daemon host machine पर ही रहेगा — container सिर्फ CLI से access करेगा।

**🛠️ 1. ✅ Build the Docker Image for Docker Jenkins Container**→ Source code से Dockerfile का उपयोग करके Node.js ऐप की Docker image तैयार करना।

root@test:/htyagi2233# docker build -t jenkins/jenkins .

root@test:/htyagi2233# docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

jenkins/jenkins jdk21 ac09998055cc 6 days ago 480MB

**▶️ 2. Run the container with Docker socket mount:**

docker run -d --name jenkins -p 8080:8080 -v jenkins\_home:/var/jenkins\_home jenkins/jenkins

**▶️ 3. Login Jenkins & Create Jenkins Pipeline**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

}

}

}

}

**✅ 4. Job Run करें**

* + Left menu → Build Now पर क्लिक करें

**✅ 5. Console Output चेक करें**

* + Left में Build Number (#1) पर क्लिक करें
  + फिर → Console Output पर क्लिक करें

**✅ Expected Output:**

🔍 Checking Docker version inside Jenkins container...  
Docker version 24.0.5, build abcdef1234

✅ इसका मतलब Jenkins container के अंदर से Docker CLI successfully काम कर रहा है।

**Project-7:**

**Step- Now build the image For NodeJS to deploy and push into DockerHub**

* + **Add the stage to build the image and push into DockerHub in jenkins Pipeline**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

}

}

}

}

* + **Error-**

+ docker build -t htyagi2233/jenkins-nodejs:latest .  
ERROR: Cannot connect to the Docker daemon at unix://**/var/run/docker.sock**. Is the docker daemon running?

**Project-8:**

**Step- Building Docker Image For NodeJS- Permission Access**

* + **Stop and delete the old container**

docker stop <jenkins container ID>

docker rm <jenkins container ID>

**🔧 Target: Jenkins container को Docker daemon (/var/run/docker.sock) access करने की permission देना**

Docker socket को access करने के लिए Jenkins container को उसी **group ID** (GID) में शामिल करना होता है, जो host system पर Docker socket की है।

**✅ Step-by-step: Docker Group Permission देना**

**🔹 Step 1: Host system पर Docker socket की group ID पता करें**

Host (जहाँ Docker install है) पर command चलाएँ:

stat -c '%g' /var/run/docker.sock

उदाहरण output:

998

यहाँ 998 Docker socket का group ID (GID) है।

**🔹 Step 2: Jenkins container को Docker run करते समय यह GID दें**

**✅ तरीका A: Auto-detect and pass using --group-add**

**Create a docker jenkins container**

docker run -d \  
 --name jenkins \  
 -p 8080:8080 \  
 -v jenkins\_home:/var/jenkins\_home \  
 -v /var/run/docker.sock:/var/run/docker.sock \  
 -v ~/.kube/config:/var/jenkins\_home/.kube/config \  
 --group-add $(stat -c '%g' /var/run/docker.sock) \  
 jenkins/jenkins:lts

🔐 --group-add flag Jenkins container को host Docker socket वाली group में डाल देता है।

**Login into Jenkins**

**<Server ip>:8080**

**Again build the image and push into DockerHub**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

}

}

}

}

**Project-9:**

**Step- Pushing the Docker Image in Docker Hub**



**🐳 STEP 1: DockerHub Token Generate करना**

DockerHub अब Token-based authentication को recommend करता है (password की जगह) — ज़्यादा secure होता है।

**🔧 कैसे करें:**

* + Login करें:  
    👉 <https://hub.docker.com>
  + Top-right → Click on your **profile icon** → Go to **Account Settings**
  + Left side में → Click on **Security**
  + Under **Access Tokens** → Click on **New Access Token**
  + Fill Details:
    - **Token Description:** e.g. jenkins-pipeline-token
    - **Access Level:**✅ Select **Read/Write**
  + Click on **Generate**
  + **Important:**
    - **जो token आपको मिलेगा, उसे कहीं copy/save कर लें**
    - **यह सिर्फ एक बार दिखता है!**

**🔐 STEP 2: Jenkins में DockerHub Token Add करना**

**Login into Jenkins**

अब हमें ये token Jenkins को securely देना है।

**📥 Jenkins में Credential Add करें:**

* 1. Jenkins UI → Manage Jenkins → Credentials
  2. Select:  
     Global → (global) → Add Credentials
  3. Fill the form:
     + **Kind:** Username with password
     + **Username:** *Your DockerHub username*(e.g. htyagi2233)
     + **Password:** *Paste the token you generated*
     + **ID:**dockerhub-creds  
         
       *(या कोई और ID जो आप pipeline में use करना चाहें)*
     + **Description:**e.g. DockerHub token for Jenkins pipeline
  4. Click **Save**

# dockerhub-creds

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

environment{

dockercred = credentials('dockerhub-creds')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

sh 'echo $dockercred\_PSW | docker login -u $dockercred\_USR --password-stdin'

sh 'docker push htyagi2233/jenkins-nodejs:latest'

}

}

}

}

**🎉 अब आप क्या कर सकते हैं?**

* 1. Jenkins pipeline से DockerHub login होगा
  2. Image build होगी
  3. Secure तरीके से push भी होगी — बिना token को hardcode किए

**DockerHub Repository verify - image successfully push**

* 1. DockerHub में login करें।
  2. ऊपर-right में अपने **username** पर क्लिक करके **Repositories** में जाएँ।
  3. htyagi2233/jenkins-nodejs रपो खोलें।
  4. वहाँ आपको **Tags** section में आपके pushed image के टैग (जैसे latest) और digest दिखाई देंगे।  
     मतलब image successfully uploaded हो चुका है

**Project-10:**

**Step- AWS Integration with Jenkins on Docker**

* 1. **Update the Dockerfile and add the steps to install aws cli**
  2. **AWS CLI installation reference document**

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-version.html>

# vi dockerfile

FROM jenkins/jenkins:jdk21

USER root

# Install dependencies

RUN apt-get update && \

apt-get install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# Add Docker’s official GPG key

RUN install -m 0755 -d /etc/apt/keyrings && \

curl -fsSL <https://download.docker.com/linux/debian/gpg> | gpg --dearmor -o /etc/apt/keyrings/docker.gpg && \

chmod a+r /etc/apt/keyrings/docker.gpg

# Add Docker repo for Debian (not Ubuntu)

RUN echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] <https://download.docker.com/linux/debian> \

bookworm stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker

RUN apt-get update && \

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

apt-get clean

# Add Jenkins user to Docker group (safe)

RUN getent group docker || groupadd docker && usermod -aG docker jenkins

# Install AWS CLI v2

RUN curl "<https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip>" -o "awscliv2.zip" && \

apt-get install -y unzip && \

unzip awscliv2.zip && \

./aws/install && \

rm -rf awscliv2.zip aws

USER jenkins

* 1. **Build the Image**

docker build -t jenkins/jenkins .

* 1. **Stop and delete the old container**

docker stop 88

docker rm 88

* 1. **Recreate the Container with the latest build Image**

docker run -d --name jenkins -p 8080:8080 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock --group-add $(stat -c '%g' /var/run/docker.sock) jenkins/jenkins

* 1. **# Jenkins Pipeline**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

environment{

dockercred = credentials('dockerhub-creds')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

sh 'echo $dockercred\_PSW | docker login -u $dockercred\_USR --password-stdin'

sh 'docker push htyagi2233/jenkins-nodejs:latest'

}

}

stage('aws integration'){

steps{

sh 'aws --version'

}

}

}

}

**❌ Error Explanation**

**We got the error here**

**✅ Solution**

**✅ STEP 1: Install AWS Credentials Plugin in Jenkins**

**🔧 Install Plugin:**

* 1. Jenkins UI → **Manage Jenkins**
  2. Go to → **Plugins**
  3. Click → **Available** tab
  4. Search: AWS Credentials
  5. ✅ Select and click **Install without restart**

**✅ STEP 2: Add AWS Access Key Credentials in Jenkins**

**🔐 Add AWS Credentials:**

* 1. Jenkins → **Manage Jenkins** → **Credentials**
  2. Choose → (global) scope
  3. Click **Add Credentials**

**Fill Details:**

* 1. **Kind:** AWS Credentials
  2. **Access Key ID:** Your AWS IAM access key
  3. **Secret Access Key:** Your AWS IAM secret
  4. **ID:**Example: aws-eks-creds  
     *(You’ll use this ID in your pipeline)*
  5. Click **Save**

**✅ STEP 3: Jenkins Pipeline to Verify AWS CLI is Working**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

environment{

dockercred = credentials('dockerhub-creds')

awscred = credentials('aws-creds')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

// stage('node stage'){

// steps{

// sh 'node -v'

// sh 'npm -v'

// sh 'npm install'

// }

// }

// stage('docker build'){

// steps{

// sh 'docker -v'

// sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

// sh 'echo $dockercred\_PSW | docker login -u $dockercred\_USR --password-stdin'

// sh 'docker push htyagi2233/jenkins-nodejs:latest'

// }

// }

stage('aws integration'){

steps{

sh 'aws --version'

sh 'aws ec2 describe-instances --region us-east-1'

}

}

}

}

**✅ Output**

* 1. Now run the Pipeline and verify this time it's work

**Project-11:**

**Step- Kubernetes Manifest Files for NodeJS for AWS EKS Deployment**

* 1. **Create a K8s deployment file**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: project-k8s

spec:

  replicas: 3

  selector:

    matchLabels:

      app: project-k8s

  template:

    metadata:

      labels:

        app: project-k8s

    spec:

      containers:

      - name: project-k8s

        image: htyagi2233/jenkins-nodejs:latest

        ports:

        - containerPort: 3000

---

apiVersion: v1

kind: Service

metadata:

  name: project-k8s

spec:

  type: Loadbalancer

  selector:

    app: project-k8s

  ports:

  - protocol: TCP

    port: 80

    targetPort: 3000 #container port

    nodePort: 30080

**Step- Kubernetes Setup & Installation**

**Step 1: Prerequisites**

* 1. **AWS Account**: Ensure you have an AWS account.
  2. **IAM Permissions**: Ensure you have the necessary IAM permissions to create EKS clusters, IAM roles, and other related resources.

**Step 2: Create IAM Roles for EKS**

**Create the EKS Service Role**

* 1. **Navigate to IAM**: Open the IAM console at [**https://console.aws.amazon.com/iam/**](https://console.aws.amazon.com/iam/).
  2. **Create Role**:
     + **In the left navigation pane, click on "Roles".**
     + **Click on the "Create role" button.**
     + **Select EKS - Cluster as the use case.**
     + **Click "Next".**
  3. **Attach Policy**:
     + **Select the AmazonEKSClusterPolicy.**
     + **Click "Next".**
  4. **Review and Create**:
     + **Role Name - RoleForEKSCluster**
     + **Click "Create role".**

**Create the EKS Nodegroup Role**

* 1. **Create Another Role**:
     + **In the left navigation pane, click on "Roles".**
     + **Click on the "Create role" button.**
     + **Select EC2 as the use case.**
     + **Click "Next".**
  2. **Attach Policies**:
     + **Select the AmazonEKSWorkerNodePolicy, AmazonEC2ContainerRegistryReadOnly, and AmazonEKS\_CNI\_Policy.**
     + **Click "Next".**
  3. **Review and Create**:
     + **The role name will be automatically generated. Review the settings.**
     + **Click "Create role".**
     + **Role Name - RoleForEKSNodegroup**

**Step 3: Create an EKS Cluster**

* 1. **Navigate to EKS**: Open the EKS console at [**https://console.aws.amazon.com/eks/**](https://console.aws.amazon.com/eks/).
  2. **Create Cluster**:
     + **Click on the "Create cluster" button.**
     + **Select "Create cluster" under the "Amazon EKS" section.**
  3. **Choose Cluster Configuration**:
     + **Select "Provision your own infrastructure" and click "Next".**
  4. **Configure Cluster**:
     + **Enter a Cluster name (e.g., my-eks-cluster).**
     + **Choose a Kubernetes version (e.g., 1.30).**
     + **Choose Cluster endpoint access (public or private).**
     + **Choose Cluster logging (optional).**
  5. **Configure Node Groups**:
     + **Click on "Add node group".**
     + **Enter a Node group name (e.g., my-nodegroup).**
     + **Choose an AMI type (e.g., Amazon Linux 2025).**
     + **Choose an Instance type (e.g., t4g.medium).**
     + **Specify the Number of nodes (e.g., 2).**
     + **Choose a Subnet (e.g., default VPC or create a new one).**
     + **Choose an IAM role (select the EKSNodegroupRole created earlier).**
  6. **Review and Create**:
     + **Review all the settings.**
     + **Click on the "Create" button.**

**Step 4: Configure kubectl**

* 1. **Update kubeconfig**:
     + **In the EKS console, select your cluster.**
     + **Click on the "Update kubeconfig" button.**
     + **This will download the kubeconfig file to your local machine.**
  2. **Verify kubectl Configuration**:
     + **Open a terminal and run:**

Shell

* 1. kubectl get svc  
     You should see the Kubernetes services running in your cluster.

**Step 5: Deploy an Application**

* 1. **Create a Deployment**:

Shell

* 1. kubectl create deployment nginx-deployment --image=nginx:1.30
  2. **Expose the Deployment as a Service**:

Shell

* 1. kubectl expose deployment nginx-deployment --type=LoadBalancer --port=80
  2. **Check the External IP**:

Shell

* 1. kubectl get svc  
     Wait for the EXTERNAL-IP to be assigned. You can then access the NGINX server by navigating to the external IP in your web browser.

**Step 6: Clean Up**

* 1. **Delete the Node Group**:
     + **In the EKS console, select your cluster.**
     + **Click on the "Node groups" tab.**
     + **Select the node group and click on the "Delete" button.**
  2. **Delete the Cluster**:
     + **In the EKS console, select your cluster.**
     + **Click on the "Delete" button.**
     + **Confirm the deletion.**
  3. **Delete IAM Roles**:
     + **In the IAM console, delete the EKSServiceRole and EKSNodegroupRole.**

**Jenkins Pipeline -**

Go to Jenkins and create a new Pipeline for testing Test-kubectl-version

pipeline{

agent any

stages{

stage('k8s version'){

steps{

sh 'kubectl version --client'

}

}

}

}

**Error-**

+ kubectl -v  
/var/jenkins\_home/workspace/NodeJs-k8s@tmp/durable-63259db9/script.sh.copy: 1: kubectl: not found

**Project-12:**

**Update our Dockerfile -**

* 1. **We will add the kubect**
  2. **Officially Website**

<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/>

FROM jenkins/jenkins:jdk21

USER root

# Install dependencies

RUN apt-get update && \

apt-get install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# Add Docker’s official GPG key

RUN install -m 0755 -d /etc/apt/keyrings && \

curl -fsSL <https://download.docker.com/linux/debian/gpg> | gpg --dearmor -o /etc/apt/keyrings/docker.gpg && \

chmod a+r /etc/apt/keyrings/docker.gpg

# Add Docker repo for Debian (not Ubuntu)

RUN echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] <https://download.docker.com/linux/debian> \

bookworm stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker

RUN apt-get update && \

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

apt-get clean

# Add Jenkins user to Docker group (safe)

RUN getent group docker || groupadd docker && usermod -aG docker jenkins

**# Install AWS CLI v2**

RUN curl "<https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip>" -o "awscliv2.zip" && \

apt-get install -y unzip && \

unzip awscliv2.zip && \

./aws/install && \

rm -rf awscliv2.zip aws

# Install the kubectl

RUN curl -LO "[https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https:/dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)" && \

install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl && \

chmod +x kubectl && \

mkdir -p ~/.local/bin && \

mv ./kubectl ~/.local/bin/kubectl

USER jenkins

* 1. **Build the Image**

docker build -t jenkins/jenkins .

* 1. **Stop and delete the old container**

docker stop 88

docker rm 88

* 1. **Recreate the Container with the latest build Image**

docker run -d --name jenkins -p 8080:8080 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock --group-add $(stat -c '%g' /var/run/docker.sock) jenkins/jenkins

261681eb0b030c2f24e90f720e70702e1337ead640396da9509b78b907b27ba2

* 1. **Login into Jenkins**
  2. **Go to Jenkins and create a new Pipeline for testing Test-kubectl-version**

pipeline{

agent any

stages{

stage('k8s version'){

steps{

sh 'kubectl version --client'

}

}

}

}

* 1. **Build the Job and see Job has run success**

kubectl version --client

+ kubectl version --client

Client Version: v1.34.0

Kustomize Version: v5.7.1

**Project-13:**

**Step- NodeJS Project End to End Deployment**

**Part- A ) For AWS EKS Cluster deployment**

**🧱 Prerequisites:**

* 1. ✅ Jenkins Docker container में चल रहा है।
  2. ✅ आपके पास एक Kubernetes cluster है -EKS
  3. ✅ Jenkins container के पास kubeconfig access होना चाहिए ताकि वह Kubernetes API से बात कर सके।
  4. ✅ Jenkins में kubectl और kubectl credentials properly configured हों।

इसके लिए हम अपनी pipeline में ये stage add करेंगे

aws eks update-kubeconfig --region region-code --name my-cluster

* 1. Github repo में updated Dockerfile होनी चाहिए

* 1. ✅ Jenkins में एक pipeline job बना हो।

**✅ Jenkins Docker container में चल रहा है on local system**

**Dockerfile**

FROM jenkins/jenkins:jdk21

USER root

# Install dependencies

RUN apt-get update && \

apt-get install -y \

ca-certificates \

curl \

gnupg \

lsb-release

# Add Docker’s official GPG key

RUN install -m 0755 -d /etc/apt/keyrings && \

curl -fsSL <https://download.docker.com/linux/debian/gpg> | gpg --dearmor -o /etc/apt/keyrings/docker.gpg && \

chmod a+r /etc/apt/keyrings/docker.gpg

# Add Docker repo for Debian (not Ubuntu)

RUN echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] <https://download.docker.com/linux/debian> \

bookworm stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# Install Docker

RUN apt-get update && \

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

apt-get clean

# Add Jenkins user to Docker group (safe)

RUN getent group docker || groupadd docker && usermod -aG docker jenkins

# Install AWS CLI v2

RUN curl "<https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip>" -o "awscliv2.zip" && \

apt-get install -y unzip && \

unzip awscliv2.zip && \

./aws/install && \

rm -rf awscliv2.zip aws

# Install the kubectl

RUN curl -LO "[https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl](https://dl.k8s.io/release/$(curl%20-L%20-s%20https:/dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl)" && \

install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl && \

chmod +x kubectl && \

mkdir -p ~/.local/bin && \

mv ./kubectl ~/.local/bin/kubectl

USER jenkins

* + **Build the image on local system and run the jenkins docker container**

docker ps

docker stop 26

docker rm 26

docker run -d --name jenkins -p 8080:8080 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock --group-add $(stat -c '%g' /var/run/docker.sock) jenkins/jenkins

* + **Dockerfile for Build the image in gitHub**
  + **Note ये Dockerfile हमारे docker container के लिए जो dockerfile create की थी उससे अलग रहेगी इसके through हम only aws eks cluster में deployment के लिए लिए करेनेगे जिस image में only हमारा NodeJS**

#BASE Image

FROM node:latest

#Working directory

WORKDIR /usr/src/app

#COPY package.json

COPY package\*.json ./

#TO get dependencies

RUN npm install

#COPY all app code

COPY . .

#Expose port

EXPOSE 3000

#Start server

CMD ["npm", "start"]

* + **Deployment file in GitHub Repo Application.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: project-k8s

spec:

replicas: 3

selector:

matchLabels:

app: project-k8s

template:

metadata:

labels:

app: project-k8s

spec:

containers:

- name: project-k8s

image: tampoohoonm/jenkins-nodejs:latest

ports:

- containerPort: 3000

---

apiVersion: v1

kind: Service

metadata:

name: project-k8s

spec:

type: LoadBalancer

selector:

app: project-k8s

ports:

- protocol: TCP

port: 80

targetPort: 3000 #container port

nodePort: 30080

* + **Push into Github**

git add .

git commit -m "Push Dockerfile and Deployment file"

git push origin main

* + **Jenkins Pipeline - we will update EKS kubeconfig**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

environment{

dockercred = credentials('dockerhub-creds')

awscred = credentials('aws-creds')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

sh 'echo $dockercred\_PSW | docker login -u $dockercred\_USR --password-stdin'

sh 'docker push htyagi2233/jenkins-nodejs:latest'

}

}

stage('aws integration'){

steps{

sh 'aws --version'

sh 'aws ec2 describe-instances --region us-east-1'

}

}

**stage('EKS kubeconfig update'){**

steps{

sh 'aws eks update-kubeconfig --region <region-code> --name <my-cluster>'

sh 'kubectl apply -f Application.yaml'

}

}

}

}

* + **Build the Job and check our AWS EKS Cluster**

**Project-14:**

**Part- B ) For K8s server deployment**

**🧱 Prerequisites:**

* + ✅ Jenkins Docker container में चल रहा है।
  + ✅ आपके पास एक Kubernetes cluster है (minikube, kind, EKS, GKE, etc.) - हमने step - 12 में कर चुके

* + ✅ Jenkins container के पास kubeconfig access होना चाहिए ताकि वह Kubernetes API से बात कर सके।
  + ✅ Jenkins में kubectl और kubectl credentials properly configured हों।
  + ✅ Jenkins में एक pipeline job बना हो।

**✅ Jenkins Docker container में चल रहा है।**

* + Stop and delete the old container

docker stop 26

docker rm 26

**📁 Step : kubeconfig Mount करें Jenkins container में**

आपको अपने Jenkins container को run करते वक्त ~/.kube/config या cluster की credentials को mount करना होगा ताकि kubectl cluster से बात कर सके।

* + **1. k8s server (192.168.192.133) पर kubeconfig file कहाँ है, यह पता करें**

आमतौर पर यह होती है:

* + /home/<user>/.kube/config  
    या
  + /etc/kubernetes/admin.conf

मान लेते हैं आपके k8s server पर kubeconfig है /home/ubuntu/.kube/config

* + 2. Jenkins server (जहा docker container jenkins चल रहा है ) (192.168.192.137) पर kubeconfig copy करें
  + Or हम menual copy भी कर सकते है

scp root@192.168.192.133:/root/.kube/config root@192.168.192.137:/root/.kube/config

* + हम एक और volume को add करेंगे   
    -v ~/.kube/config:/var/jenkins\_home/.kube/config

* + Recreate the Container with the latest build Image

docker run -d --name jenkins -p 8080:8080 -v jenkins\_home:/var/jenkins\_home -v /var/run/docker.sock:/var/run/docker.sock --group-add $(stat -c '%g' /var/run/docker.sock) -v ~/.kube/config:/var/jenkins\_home/.kube/config jenkins/jenkins

* + **Login into Jenkins**

**✅ Jenkins में kubectl और kubectl credentials properly configured हों।**

**Option B: Jenkins Credentials में Add करना (Recommended for Security)**

* + Jenkins → **Manage Jenkins** → **Credentials** → (Global or Folder Level)
  + **Add Credentials** → **Kind: Secret file**
  1. Upload करें आपका kubeconfig file
  2. उसे एक ID दें, जैसे: kubeconfig

* + **Go to Jenkins and create a new Pipeline for testing tset-k8s-cluster**

pipeline{

agent any

stages{

stage('k8s version'){

steps{

withCredentials([file(credentialsId: 'k8s-creds', variable: 'KUBECONFIG')]) {

sh 'kubectl cluster-info'

sh 'kubectl get nodes'

}

}

}

}

}

* + **Build the Job and see Job has run success**

✅ kubectl cluster-info

+ kubectl cluster-info

Kubernetes control plane is running at <https://192.168.192.133:6443>

CoreDNS is running at <https://192.168.192.133:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy>

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

✅ kubectl get nodes

+ kubectl get nodes

NAME         STATUS   ROLES           AGE     VERSION

k8s          Ready    control-plane   4d18h   v1.33.4

k8s-worker   Ready    <none>          4d17h   v1.33.4

* + **Now final deployment on k8s**

pipeline{

agent any

options {

buildDiscarder logRotator(artifactDaysToKeepStr: '', artifactNumToKeepStr: '', daysToKeepStr: '7', numToKeepStr: '2')

}

tools{

nodejs 'Nodejs'

}

environment{

dockercred = credentials('dockerhub-creds')

awscred = credentials('aws-creds')

}

stages{

stage('Git Checkout'){

steps{

checkout scmGit(branches: [[name: '\*/main']], extensions: [], userRemoteConfigs: [[url: '<https://github.com/htyagi2233/nodejs-docker-project.git>']])

}

}

stage('node stage'){

steps{

sh 'node -v'

sh 'npm -v'

sh 'npm install'

}

}

stage('docker build'){

steps{

sh 'docker -v'

sh 'docker build -t htyagi2233/jenkins-nodejs:latest .'

sh 'echo $dockercred\_PSW | docker login -u $dockercred\_USR --password-stdin'

sh 'docker push htyagi2233/jenkins-nodejs:latest'

}

}

stage('aws integration'){

steps{

sh 'aws --version'

sh 'aws ec2 describe-instances --region us-east-1'

}

}

stage('k8s version'){

steps{

withCredentials([file(credentialsId: 'k8s-creds', variable: 'KUBECONFIG')]) {

sh 'kubectl cluster-info'

sh 'kubectl get nodes'

sh 'kubectl apply -f Application.yaml'

}

}

}

}

}

* + **Dockerfile for Build the image**
  + Note ये Dockerfile हमारे docker container के लिए जो dockerfile create की थी उससे अलग रहेगी इसके through हम only aws eks cluster में deployment के लिए लिए करेनेगे जिस image में only हमारा NodeJS

#BASE Image

FROM node:latest

#Working directory

WORKDIR /usr/src/app

#COPY package.json

COPY package\*.json ./

#TO get dependencies

RUN npm install

#COPY all app code

COPY . .

#Expose port

EXPOSE 3000

#Start server

CMD ["npm", "start"]

* + **Create a K8s deployment file**

apiVersion: apps/v1

kind: Deployment

metadata:

  name: project-k8s

spec:

  replicas: 3

  selector:

    matchLabels:

      app: project-k8s

  template:

    metadata:

      labels:

        app: project-k8s

    spec:

      containers:

      - name: project-k8s

        image: htyagi2233/jenkins-nodejs:latest

        ports:

        - containerPort: 3000

---

apiVersion: v1

kind: Service

metadata:

  name: project-k8s

spec:

  type: NodePort

  selector:

    app: project-k8s

  ports:

  - protocol: TCP

    port: 80

    targetPort: 3000 #container port

    nodePort: 30080

* + **Build the Job and verify** 
    - Latest Docker image has pushed into DockerHub Repo
    - On K8s Server Deployment has created with NodePort Service
    - So copy the Node Ip and Node Port

* + Open the Browser and test the application

<http://192.168.192.133:30080/>

* + **Bonus-**

docker exec -it <container\_name\_or\_id> bash

docker exec -it jenkins-cicd cat /var/jenkins\_home/secrets/initialAdminPassword

**docker container rm -f $(docker container ls -aq)**