**Experiment-1**

**Aim:** Start DevOps with a workflow that includes four phases: to do, in progress, code review, and done.

**Require Software & Tools:** JIRA, KANBAN.

**Procedure:**

**Phase 1: To Do**

Objective: Identify and prioritize tasks or features to be developed.

Key Actions:

* Define tasks clearly in a backlog.
* Prioritize tasks based on impact, urgency, and dependencies.
* Assign owners or teams to each task.

Tools: Jira, Trello, GitHub Issues, or Asana.

**Phase 2: In Progress**

Objective: Actively work on tasks selected from the "To Do" phase.

Key Actions:

* Begin coding or configuring based on task requirements.
* Update the task status to reflect ongoing work.
* Ensure team members collaborate effectively (e.g., stand-ups, pair programming).

Best Practices:

* Use branches in version control systems for individual tasks (e.g., Git feature branches).
* Write unit tests alongside development.

**Phase 3: Code Review**

Objective: Validate the quality, functionality, and security of the code.

Key Actions:

* Submit pull requests for peer review.
* Review code for adherence to standards, logic, and potential issues.
* Approve or request changes.

Tools: GitHub Pull Requests, GitLab Merge Requests, Bitbucket.

Automation: Integrate CI/CD pipelines to run tests automatically during reviews.

**Phase 4: Done**

Objective: Mark tasks as completed and deploy changes if necessary.

Key Actions:

* Merge the approved code into the main branch.
* Deploy to staging or production environments.
* Monitor deployment and validate functionality.

Post-Completion:

* Add documentation for the changes.
* Gather feedback from stakeholders or users.

**Workflow Visualization**

A Kanban board or similar visual representation can help track the status of tasks across these phases. For example:

1. To Do: Contains all pending tasks.

2. In Progress: Tasks currently being worked on.

3. Code Review: Tasks awaiting review or approval.

4. Done: Completed and deployed tasks.

**Tools:** Trello, Jira, Azure.

**Experiment-2**

**Aim:** Setups Eclipse for Devops

**Require Software & Tools:** Eclipse, Java jdk-17, Tomcat v.9, TestNG and Dependencies.

**Procedure:**

Step-1: Install Jdk-17 and set the java path in System environment

Step-2: Download eclipse zip file and extract the contents of all eclipse file

Step-3: Create a Maven Project from eclipse as:

OR

Click on File in left corner -> Click on new -> click on Maven Project and follow the given image steps.

b. Click Next and Search org.apache.maven.archetypes and select webapp file

c. In a group id you can type anything like name and in the artefact id: you can type anything like your roll number

d. Click Finish

e. Type Y and Press enter, you should see a Build Success message.

Step-4: now open your pom.xml file and add your dependencies (Given file, Copy and Paste)

* Maven testing dependency
* Maven junit dependency
* Javax servlet api dependency
* Maven surefire plugin
* Maven compiler plugin(set configuration)

Step-5: Update your project once (Right click on Project -> click on Maven -> click on Update Project)

Step-6: Download Apache tomcat v9 from Official website.

Step-7: After Download the Apache tomcat, Extract the .zip file and paste your apache-tomcat-9.0.98 folder in your folder

Step-8: Now click on your project option in Menu -> Click on Properties -> Click on Targeted Runtime

Step-9: Click on new

Step-10: Select Apache Tomcat v9.0

Step-11: Click on Browse and Select your Extracted file and then click on finish.

Step-12: Now Click on help Menu -> click on Install new Software.

Step-13: Click on Add and it will show a popup dialog box

In the place of Name type: TestNG

In the place of Location type: https://testng.org/testng-eclipse-update-site/

Step-14: Click on Add -> It will load a testNG Dependencies -> Select TestNg and then click Next.

It will take 10 minute to update TestNG in our Project

Step-15: After downloading the all the dependencies it will show some file select all and click on next.

Step-16: Accept Terms and condition and click on finish

Step-17: After finish it will show restart option (Restart the Project) otherwise just update once of your project.

Step-18: Now Login your GitHub Account.

Step-19: Create a New Repository and Copy your Repository and paste in notepad

Step-20: After that Click on your Profile in Right corner -> Click on Setting.

Step-21: It will show a new page, scroll down and select the developer setting -> click on personal access token -> select Token(Classic) -> click on Generate new token and select Generate new token(Classic) -> write your token name and select repo option and scroll down and click on Generate Token. (Follow the given Image)

Step-22: After Generating the token copy the token id and paste in a NotePad.

Step-23: Now come on your project and right click on your project -> Click on Team -> Click on Share Project.

Step- 24: It will open a Dialog Box for GitHub Setup, select the option Use or create repository in parent folder of project - > Select your Project and Click on Create Repository and click on Finish.

Step-25: After that again Right click on your Project and select the Team -> click on Commit -> and stage your all file -> and Write a comment (i.e. First Commit) and click on Commit and push -> after that it show an error dialog -> click OK -> now again click on Push Head Button

Step-26: After that again click on Push Head, it will show a dialog, paste your Repository URL in the URL section and type your GitHub User Id and Password in User, password section -> Click on preview -> Again click on Preview.

Step-27: After that it will again show a user Id and Password option -> just type your Github id in user section and paste your Token id in Password section -> click on push -> one more time it will ask user id and password just repeat your last step with user id and token id -> now check your repository on github, your file is uploaded or not

Step-28: Now you have to create a simple java code in SRC File, so first open your project from file manager -> open SRC - > Create two folder in SRC -> first name: java, second name: test -> now open test folder and create two more folder in test folder -> now come on your eclipse IDE and Update your project once -> After that create a java class file with a Statement “Hello World” in your SRC/TEST/java folder.

Step-29: Now Push again your all unstage files in your GitHub Repository with different version or Comment (it is just for Version Control).

Step-30: Now Check again your Repository your recent file is uploaded or not with different version.

**Experiment-3**

* Ec2 instance
* Ssh---custom---0.0.0.0/0
* Custom tcp---8080---anywhere---0.0.0.0/0
* http---anywhere---0.0.0.0/0
* connect ssh
* sudo wget -O /etc/apt/keyrings/jenkins-keyring.asc \

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

* echo "deb [signed-by=/etc/apt/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 update
* sudo apt install fontconfig openjdk-21-jre
* java -version
* sudo apt-get update
* sudo apt-get install Jenkins
* jenkins -version
* sudo systemctl enable jenkins
* sudo systemctl start jenkins
* sudo systemctl status jenkins

**Experiment-4**

* 2 ec2 instances (sit and exp3)
* On exp3 download java, Jenkins
* Go Jenkins.io -> download -> right click on generic java package (.war) ->copy link address
* In git bash enter: $wget <https://get.jenkins.io/war-stable/2.492.2/jenkins.war>
* Start the Jenkins: $java -jar jenkins.war
* Browser ip:8080
* Check java is installed or not. If not install then install java
* Check maven is installed or not. if not install then install maven
* Check jenkins is installed or not . if not install then install Jenkins
* Go to the root directory
* sudo su -
* cd /opt
* Open the browser type maven download
* wget <https://dlcdn.apache.org/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz>
* Unzip the maven zip file: tar -xvzf apache-maven-3.9.9-bin.tar.gz
* Rename apache-maven-3.9.9 to maven: mv apache-maven-3.9.9 maven
* cd maven
* ll
* Note down the maven path
* #pwd

Output: /opt/maven

* Move to: cd bin
* #ll
* Note down the maven bin path (/opt/maven/bin): #pwd
* Now check the maven is install or not

mvn --version (or) ./mvn --version

* Go to root dir: cd ~
* mvn --version
* It shows not found
* So we need to create environment variable
* Go to root directory with below command: cd ~
* ll
* sudo vim .profile
* Go to the insert mode( click on I) and give the maven, java home and m2 paths here
* M2\_HOME=/opt/maven
* M2=/opt/maven/bin
* JAVA\_HOME=/usr/lib/jvm/java-21-openjdk-amd64
* PATH=$PATH:HOME/bin:$JAVA:$M2:HOME:$M2
* :wq
* To get the java\_home path: #find / -name java-21\* (/usr/lib/jvm/java-21-openjdk-amd64)
* #echo $PATH
* U CAN’T SEE THE JAVA AND MAVEN PATH ABVOE . SO WE NEED TO RESTART THE .PROFILE FILE with below command: #source .profile
* Now go to the jenkins dashboard u need to install one plugin (maven integratin)

Managejenkins-->plugins-->available plugins-->maven integration plugin

* Without this plugin we can’t able to see the maven project
* Once installed, click on the restart Jenkins
* U need to add java and maven paths in the jenkins

Go to jenkins dashboard-->managejenkins-->tools Add JDK and add MAVEN

Java path: /usr/lib/jvm/java-21-openjdk-amd64

Maven path: /opt/maven

* Dashboard-->newitem-->war(give any name)
* Source code management-->select git--> give the github url project path(open my github account-->go to repositories)
* Select the repository select maven project click on ok
* Select the build goal and options write the command clean install click on apply and save
* Click on build now
* If job execution is taking lot of time
* Then logout the jenkins
* Stop the dev instance
* Restart the dev instanc
* Start the jenkins server

**Experiment-5**

**How to setup Ansible and SSH keys in AWS**

**⚙️ Ansible and SSH Key Setup in AWS**

**📌 Step 1: Launch EC2 Instances**

* Create **3 Ubuntu EC2 Instances**:
  + ansible (Control Node)
  + server1 (Managed Node)
  + server2 (Managed Node)
* HTTP---80---Anywhere

**📌 Step 2: Setup on Ansible EC2 Instance**

**🔧 Login and Install Ansible**

sudo su -

apt update -y

apt-add-repository ppa:ansible/ansible

apt update

apt install ansible -y

ansible --version

**📌 Step 3: Configure Hostnames for Easier Access**

Edit the hosts file:

nano /etc/hosts

Add the following lines:

server1-public-ip server1

server2-public-ip server2

**📌 Step 4: Generate SSH Keys and Share with Servers**

**🗝️ Generate SSH key on Ansible server:**

ssh-keygen -t rsa

# Press Enter three times to accept defaults

cat ~/.ssh/id\_rsa.pub

**🔑 Copy and paste the public key to server1 and server2:**

**On both server1 and server2:**

sudo apt update

mkdir -p ~/.ssh

nano ~/.ssh/authorized\_keys

# Paste the public key here (do not remove existing text)

**📌 Step 5: Verify SSH Connection from Ansible**

On Ansible server, test SSH access:

ssh ubuntu@server1

exit

ssh ubuntu@server2

exit

✅ If no password is asked, SSH keys are properly set up.

**📌 Step 6: Setup Ansible Inventory and Config**

**Create ansible directory:**

mkdir /root/ansible

cd /root/ansible

**Create inventory file:**

nano inventory

Add:

[webservers]

server1

server2

**Create ansible.cfg file:**

nano ansible.cfg

Add:

[defaults]

inventory=/root/ansible/inventory

remote\_user=ubuntu

ask\_pass=false

**📌 Step 7: Create YAML Playbook to Install Web Servers**

nano install\_webservers.yml

Paste:

---

- name: Install Web Servers

hosts: webservers

become: true

tasks:

- name: Install Nginx on server1

apt:

name: nginx

state: present

when: inventory\_hostname == 'server1'

- name: Install Apache on server2

apt:

name: apache2

state: present

when: inventory\_hostname == 'server2'

- name: Ensure Nginx is started and enabled on server1

service:

name: nginx

state: started

enabled: yes

when: inventory\_hostname == 'server1'

- name: Ensure Apache is started and enabled on server2

service:

name: apache2

state: started

enabled: yes

when: inventory\_hostname == 'server2'

**📌 Step 8: Run the Playbook**

ansible-playbook -i /root/ansible/inventory install\_webservers.yml

**✅ Verification:**

* Open server1 public IP in browser → You should see **Nginx**
* Open server2 public IP in browser → You should see **Apache2**

🎉 **Done!** You’ve successfully set up Ansible with SSH keys and deployed web servers.

**Experiment-6**

**Deploy artifcat into sit server**

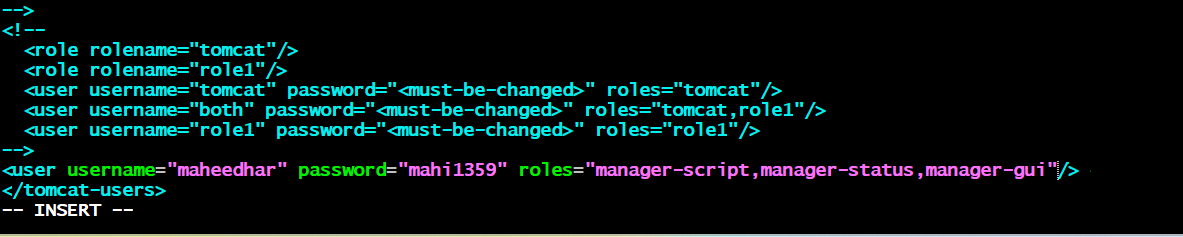
* Connect to sit machine with git bash
* Update the apt repository: sudo apt-get update
* Install tomcat 9: sudo apt-get install -y tomcat9
* Now install tomcat9 admin as well: sudo apt-get install tomcat9-admin
* how can we access the tomcat

Take the publicip of sit server and add 8080(copy paste this command on the browser)

* Now go to below path

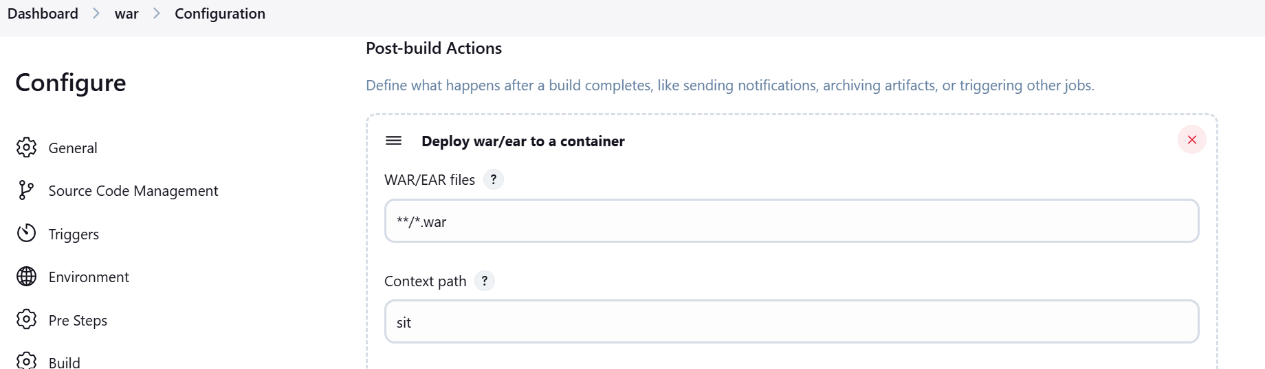
cd /etc/tomcat9

* $ll
* now we need to add user in the tomcat-users.xml file: sudo vim tomcat-users.xml
* Go to the insert mode



:wq

* Now restart the tomcat service: sudo service tomcat9 restart
* Now u can add one plugin in the jenkins (deploy to container)
* Manage jenkins-->plugins-->available plugins-->select deploy to container and install it
* Now go to the jenkins dashboard
* Select the war job
* Select the configure
* Select the post build action and search deploy war/ear to a container



* Add container tomcat 9
* Add the credentials
* Select the credentials and give the tomcat url (sit server path along with 8080 port no)
* Click on apply and save
* And run the build now
* How can we check the artifact is deployed or not
* Take the sit public ip address and port no and give the context path name
* Ex: http://3.110.55.250:8080/sit

**Experiment-7**

$ git clone https://github.com/RamyaReddyM/DevOps-2.git

$ cd Devops-2

$ ll

$ cd src

$ cd main

$ cd webapp

$ vim index.jsp

$ git add .

$ git status

$ git commit -m “commit msg”

$ git status

$ git push origin master

* Go to Jenkins
* Configure project
* Triggers
* Untick all
* Tick only poll scm
* Schedule: \* \* \* \* \*
* Apply and save
* Don’t build now
* It will automatically build
* Refresh tomcat
* Changes will be seen automatically

**Experiment-8**

**Build And Deploy a grid for Chrome and Firefox based testing**

**Step 1: Launch AWS EC2 Instance:-**

Go to AWS EC2 Console.

Launch a new instance with the following settings:

-Name: SeleniumGridServer

- Amazon Machine Image: Ubuntu Server 22.04 LTS (Free Tier)

- Instance Type: t2.micro

- Key Pair: Create New Key Pair or select one

- Add Security Group Rule:

1)SSH (default)

2)Custom TCP (port:4444, Source type: Anywhere)

**Step 2: Connect to EC2 Instance**

**Step 3: Install Docker and Docker Compose:-**

1. sudo apt update

**To Install Docker**

1. sudo apt install -y docker.io

**To start and enable Docker**

1. sudo systemctl start docker
2. sudo systemctl enable docker

**To Install Docker Compose**

1. sudo curl -L "https://github.com/docker/compose/releases/download/v2.17.3/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
2. sudo chmod +x /usr/local/bin/docker-compose

**To Check versions**

1. docker –version
2. docker-compose –version

**Step 4: Create Selenium Grid with Docker Compose:-**

1. mkdir selenium-grid && cd selenium-grid
2. nano docker-compose.yml

**Paste this code:**

version: "3"

services:

selenium-hub:

image: selenium/hub:4.0.0-rc-2-20210930

container\_name: seleniumHub

ports:

- "4444:4444"

chrome:

image: selenium/node-chrome:4.0.0-rc-2-20210930

container\_name: chromeNode

depends\_on:

- selenium-hub

environment:

- SE\_EVENT\_BUS\_HOST=selenium-hub

- SE\_EVENT\_BUS\_PUBLISH\_PORT=4442

- SE\_EVENT\_BUS\_SUBSCRIBE\_PORT=4443

shm\_size: 2g

firefox:

image: selenium/node-firefox:4.0.0-rc-2-20210930

container\_name: firefoxNode

depends\_on:

- selenium-hub

environment:

- SE\_EVENT\_BUS\_HOST=selenium-hub

- SE\_EVENT\_BUS\_PUBLISH\_PORT=4442

- SE\_EVENT\_BUS\_SUBSCRIBE\_PORT=4443

shm\_size: 2g

(OR)

version: "3"

services:

selenium-hub:

image: selenium/hub:4.0.0-rc-2-20210930

container\_name: selniumHub

ports:

- "4444:4444"

chrome:

image: selenium/node-chrome:4.0.0-rc-2-20210930

container\_name: chromeNode

depends\_on:

- selenium-hub

environment:

SE\_EVENT\_BUS\_HOST: selenium-hub

SE\_EVENT\_BUS\_PUBLISH\_PORT: 4442

SE\_EVENT\_BUS\_SUBSCRIBE\_PORT: 4443

firefox:

image: selenium/node-firefox:4.0.0-rc-2-20210930

container\_name: firefoxNode

depends\_on:

- selenium-hub

environment:

SE\_EVENT\_BUS\_HOST: selenium-hub

SE\_EVENT\_BUS\_PUBLISH\_PORT: 4442

SE\_EVENT\_BUS\_SUBSCRIBE\_PORT: 4443

**Save (Ctrl + O, Enter), then exit (Ctrl + X)**

**Step 5: Start the Selenium Grid:-**

**To Run the grid:**

sudo docker-compose up -d

**To Check containers:**

sudo docker ps

**Step 6: Access Selenium Grid UI:-**

**Open in your browser:** http://<Your-EC2-Public-IP>:4444/ui

**Step 7: Run a Sample Python Test:-  
Create a Virtual Environment:**

**Install Python and Selenium:**

sudo apt install python3-venv python3-full -y

python3 -m venv venv

source venv/bin/activatepip install selenium  
nano test\_grid.py

**Paste below code:**

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = "chrome"

GRID\_URL = "http://localhost:4444/wd/hub"

options = None

if browser == "chrome":

options = webdriver.ChromeOptions()

elif browser == "firefox":

options = webdriver.FirefoxOptions()

else:

raise Exception("Unsupported browser!")

driver = webdriver.Remote(

command\_executor=GRID\_URL,

options=options

)

driver.get("https://www.google.com")

print("Title:", driver.title)

driver.quit()

**Save (Ctrl + O, Enter), then exit (Ctrl + X)**

python3 test\_grid.py

**Experiment-9**

* Create the **google cloud console** free account
* It is a two step process
* It is deducting the 2 rupees from your account and it will give the 330$ free credit points.
* **NOTE: Don’t active the full account**
* Once the account is created u can login to google cloud console
* NOW CREATE THE KUBERNETES CLUSTER
* Open the cloud shell
* To see the cluster list run the below command:

gcloud container clusters list

(no clusters are there)

* You create the cluster with below command:

gcloud container clusters create my-cluster --zone us-central1-a

* Cluster creation is taking 5 to 10 min time
* Once the cluster is created u can see the below message automatically
* Now u go and check kubernetes engine--->cluster, you can see the my-cluster is running
* Run the below command

gcloud container clusters get-credentials my-cluster --zone us-central1-a

* To see the list of nodes:

kubectl get nodes

* Create the pods

kubectl run --image tomcat webserver

* To see the pods list

kubectl get pods

* To get the list of pods along with ip address and which node the pod is running

kubectl get pods -o wide

* Actually u can create the pod using definition file Create pd-df1.yaml

vim pd-df1.yaml

apiVersion: v1

kind: Pod

metadata:

name: jenkins-pod

spec:

containers:

- name: myjenkins

image: jenkins/jenkins

ports:

- containerPort: 8080

hostPort: 8080

* for accessing the application u need to open the port
* How to open the port:

gcloud compute firewall-rules create rule2 --allow tcp:8080

kubectl create -f pd-df1.yaml

kubectl get pods -o wide

kubectl get nodes -o wide

* How can we access the pod
* Take the external ip add the port no 8080
* Open the browser paste ipaddress:8080
* Now u can able to see the jenkins

**Experiment-10**

**✅ A) Create an EC2 Instance**

* Launch an **Ubuntu** EC2 instance on AWS.
* Select appropriate key pair and security group (allow port 22(SSH), 80(HTTP), and 3000(Custom TCP)).

**✅ B) Install Required Packages**

SSH into your EC2 instance:

ssh -i your-key.pem ubuntu@your-ec2-public-ip

Update and install packages:

# 1. Update system and install Nginx

sudo apt update -y

sudo apt install nginx -y

# 2. Install Node.js (latest stable) using NVM (preferred)

curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

source ~/.bashrc

nvm install 18

nvm use 18

# 3. Install PM2 globally

npm install -g pm2

sudo apt update -y

sudo apt install nginx -y

sudo apt install nodejs -y

sudo apt install npm -y

sudo npm install -g pm2

**✅ C) Create a Node.js App**

**1. Create your app directory and file:**

cd /home/ubuntu

nano hello.js

Paste this code:

const http = require('http');

const hostname = '0.0.0.0';

const port = 3000;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello World!\n');

});

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

Save with Ctrl+O, then Enter, then Ctrl+X.

**2. Run the app with PM2:**

pm2 start hello.js --name app

**✅ D) Set Up Nginx as Reverse Proxy**sudo rm /etc/nginx/sites-enabled/default  
sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/

sudo systemctl restart nginx

Edit Nginx config:

sudo nano /etc/nginx/sites-available/example.com

Paste this (replace with your EC2 public IP):

server {

listen 80;

server\_name YOUR\_EC2\_PUBLIC\_IP;

location / {

proxy\_pass http://localhost:3000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

sudo nginx -t

sudo systemctl reload nginx

Create a symlink:

sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/

Restart Nginx:

sudo systemctl restart nginx

Test your app in a browser: http://YOUR\_EC2\_PUBLIC\_IP  
You should see: **Hello World!**

**✅ E) Set Up Docker**

**1. Install Docker and Docker Compose:**

sudo apt install -y docker.io

sudo apt install -y docker-compose

**2. Prepare app directory for Docker:**

mkdir -p /home/ubuntu/node

cd /home/ubuntu/node

Move your hello.js file:

cp /home/ubuntu/hello.js .

**3. Create Dockerfile:**

nano Dockerfile

Paste this:

FROM node:12

WORKDIR /app

COPY . .

RUN npm install

EXPOSE 3000

CMD ["node", "hello.js"]

Save and exit.

**4. Create .dockerignore:**

nano .dockerignore

Paste:

node\_modules

npm-debug.log

**✅ F) Build and Push Docker Image**

**1. Build Docker image:**

sudo docker build -t your\_dockerhub\_username/node-app:latest .

Replace your\_dockerhub\_username with your real Docker Hub username.

**2. Check image is built:**

sudo docker images

**3. Log in to Docker Hub:**

sudo docker login

Enter your Docker Hub username and password.

**4. Push the image:**

sudo docker push your\_dockerhub\_username/node-app:latest

**5. Verify on Docker Hub:**

Go to: <https://hub.docker.com/repositories>  
Check if your image node-app is listed under your username.

**Experiment-11**

**ACCESS GRAFANA:**

* kubectl get secret prometheus-grafana -n monitoring -o jsonpath="{.data.admin-user}" | base64 --decode ; echo

If you run the above command u can see the username for grafana (**admin**)

* kubectl get secret prometheus-grafana -n monitoring -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

If you run the abvoe command u can see the password for grafana (**prom-operator**)

* PORT FORWARDING

kubectl port-forward svc/prometheus-grafana 3000:80 -n monitoring

* Click on the web preview give the port no 3000 and click on change and preview. u can see the grafana
* You can login with admin and prom-operator

**Experiment-12**

**PROMETHEUS SETUP:**

* helm repo add prometheus <https://prometheus-community.github.io/helm-charts>
* helm repo update
* helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>
* helm repo update
* helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace

This will install prometheus, alertmanager and grafana

* Check the prometheus pods and services:

kubectl get pods -n monitoring

kubectl get svc -n monitoring

* Access prometheus and port forwarding

kubectl port-forward svc/prometheus-kube-prometheus-prometheus 9090:9090 -n monitoring

* Click on the webpreview
* Change port no to 9090
* Click on change and preview
* Now u can able to see prometheus in the browser