exp-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"
- Key Actions: Phase
  - Begin coding or configuration based on task requirements.
  - Update the task status to reflect ongoing work.
  - Ensure team members collaborate effectively.

Best practices:-

- use branches in version control systems for individual tasks.
- 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 Visualisation

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

ex: 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.

exp-2

→ install JDK-17 Installer - set system variable path (till bin).

→ install eclipse and select Enterprise Java & web Developers.

→ Download Tomcat v9 → core → 64-bit → zip → extract it.

→ Adding Tomcat v9 to eclipse:

→ window → preferences → Servers → runtime → Add → Apache

→ Tomcat v9 → Next → Browse → select extracted Tomcat v9.

→ Finish → Apply & close.

→ Add JDK 17 : runtime → Tomcat → edit → installed jres →

JDK 17 → Apply & close.

6) create a Mave project.

File → New → Maven → Next.

filter → maven-archetype-webapp → Next

7) Group id: DevOpsCourse

artifact id: MyDevPipeline. → finish → Y → enter (Build success)

8) change/add the dependencies to the pom.xml file.

(change to 12 in last plugin as Java 17).

→ project → properties → Project Facets (Dynm, Java 17).

→ Apply & close.

9) project → Maven → update → check Force update snapshots → ok.

10) Targeted Runtimes :

Properties → Targeted Runtimes → select Tomcat-v9 → Apply & close

File → Restart eclipse -

11) Install TestNg :

Help → Install New s/w → Add → Name: TestNG location:

https://testng.org/testng-eclipse-update-site/ → Add.

→ Next (installs) → Next → accept → Finish. (Restart).

→ src → create TEST folder → java folder → class and write sample java program.

Run → Run configurations → ~~New config~~ Test → Run.

→ open github create new repo: MyDevPipeline, copy link.

→ Generate personal Access Token → profile → settings → Developer setting

→ PAT. Classic Token → Generate → given name and scope (repo, workflow, write). → Copy.

→ open eclipse. → ~~file~~ right click project → Team → Share project.

→ select use in Git → create → Finish.

→ Team → commit → stage all files (select and Enter) → write commit message → commit and Push → Push head fill details like url, username, password(PAT) → Push.

CDP-3 Setup Jenkins

-> open aws account -> go to EC2 instances and click on launch instance.

-> select the name (devm)

-> select Ubuntu

-> select all generations in instance type.

-> generate a new key pair and download it.

-> Network settings - click allow HTTP traffic. then click edit.

-> Add security group rule.- Type CustomTCP
port range 8080
source 0.0.0.0/0

-> select No. of instances (2) and launch. after launching change one instance to sit.

-> open file manager -> downloads where pem key is present and open git bash from there.

-> go to devm instance, click connect and SSH client copy the Example and paste in git bash to connect.

ssh -i "Exp3.pem" Ubuntu@ec2-16-16-27-22.eu-north-1.compute.amazonaws.com

-> type yes and press enter. It will be connected.

step-1 update and install java

-> give commands.

sudo apt update
sudo apt instal fontconfig openjdk-21-jre
java -version.

Step-2 install jenkins

> 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-get install jenkins

> sudo systemctl start jenkins.

Step-3 to get jenkins initial password and Accessing and setup

> sudo systemctl status jenkins (It shows active)

-> go to browser type <instance IP>:8080. (opens jenkins)

-> In git bash it shows address of the password copy it and add sudo cat at start.

sudo cat <path of pass>

sudo cat /var/lib/jenkins/secrets/initialAdminPassword.

-> copy the password and paste in jenkins.

-> click on install suggested plugins

-> Fill details like user, password and remember then click start jenkins.

exp-4 Build WAR File.

Continue after ~~setting~~ setting up jenkins.

Step-1 : Download and Extract Maven.

cd /opt

sudo wget https://dlcdn.apache.org/maven/maven-3.9.9-bin.tar.gz

sudo tar -xvzf apache-maven-3.9.9-bin.tar.gz

sudo mv apache-maven-3.9.9 maven.

Step-2 : Configure Environment variables.

>ls (check for maven).

>sudo vim ~/.profile (press i to Insert the code at bottom).

```
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
PATH=${M2_HOME}/bin:${PATH}
```

click esc, ~~Ctrl~~ :wq -> enter (saves file).

Reload

```
> source ~/.profile
> mvn --version
```

Configure jenkins Tools

Go to jenkins dashboard -> manage jenkins -> plugins -> available plugins -> search 'maven Integration' -> select and install.

Configure JDK & maven

-> manage jenkins -> Tools.

Add JDK:

Name : java-21

JDK HOME: /usr/lib/jvm/java-21-openjdk-amd64

Add maven

Name : Maven

HOME : /opt/maven.

click apply & save.

Create Maven Job (WAR project)

step-1 : create new job

→ dashboard → New item → select Maven project.

→ Give name & click ok.

step-2 : Configure Git SCM.

→ select git project.

→ go to ~~maven~~ your project and copy the url.
[ex: maheepalle practice].

→ under source code management select git and paste same url.

Step-3 : Add Goals.

"clean install" → add this in goals → apply & save

Step-4 : Run Build & check war creation.

~~apply & save.~~

click on build now.

If taking more time to build.

→ logout

→ reboot instance. [stop & start].

→ start Jenkins

→ Run job again. [login and run job again].

Exp-6 Deploy artifact

→ Connect the SIT server on aws.

Step-1: install Tomcat

> Sudo apt update

> Sudo apt install -y tomcat10 tomcat10-admin.

Step-2 Configure tomcat-users.xml

> cd /etc/tomcat10

copy the (ipaddress : 8080) check if it works.

> Sudo nano tomcat-users.xml

at last before </tomcat-users> add these.

<role rolename = "manager-gui" />

<role rolename = "manager-script" />

<user username = "admin" password = "admin" roles = "manager-gui, manager-script" />

.cntrl + x → y → enter.

Step-3 : Restart Tomcat.

> Sudo servic tomcat10 Restart.

Step-4 : Access Tomcat.

→ search in browser http://<Set-Ipadsess> : 8080

Deploy war to SIT from Jenkins

Step-1 : install Deploy to container plugin.

dashboard → manage jenkins → plugins → available plugins → Deploy to contain (search) → Install.

step-2 Add Post-Build Actions

Dashboard -> project -> configure -> (scroll) click Add Post-build Action -> select Deploy WAR/EAR to a container.

WAR/EAR : **/*.war

context path : sit.

click add container -> select Tomcat 9.x

click add credentials -> jenkins -> given user name password.

: click add.

Now, select the credentials, add Tomcat to url.

click apply & save.

-> click on Build now.

-> now visit http://<IP.address of sit>:8080/sit.

It displays our code html.

exp-7 perform automation

-> Dashboard -> select job -> configure -> Build Triggers ->

-> select Poll scm -> in schedule give * * * * *

apply & save.

-> Now, go to github forked repo -> src/main -> index.jsp (edit this file)

Now, wait two minutes the changes will be visible in http://<IP>:8080/sit

exp-5 Ansible - SSH Keys.

Step-1 setup instances.

→ open aws and create 3 instances.

Ansible (controller) [Allow HTTP].
server1 (Node)
server1 (Node).

→ Connect Ansible via SSH client - copy the address of SSH and paste in git bash.

Step-2 setup on Ansible.

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.

Ansible

> nano /etc/hosts.

add these two lines below local host.

< server1 public IP > server1
< server2 public IP > server2.

Generate SSH Keys.

> ssh-keygen -t rsa [press Enter for no passphrase]

> cat ~/.ssh/id_rsa.pub

copy and paste the public key on server1 and server2.

>sudo apt update.

>mkdir -p ~/.ssh

>nano ~/.ssh/authorized_keys

Now, paste the key, ctrl+x -> y -> enter. [on both servers].

~~Next, Or~~

Ansible verify both servers.

>ssh ubuntu@server1 [if no password asked then it means, successfully]

>exit

>ssh ubuntu@server2

>exit

Setup - Ansible inventory.

Now, we use Ansible inventory to manage both nodes using ansible playbooks.

>mkdir /root/ansible

>cd /root/ansible

>nano inventory

Add: [webservers]
server1
server2

>nano ansible.cfg

Add: [defaults]
inventory = /root/ansible/inventory
remote_user = ubuntu
ask_pass = false.

>nano install_webservers.yml

Paste your code.

>ansible-playbook -i /root/ansible/inventory install_webservers.yml.

server1 -> Nginx
server2 -> Apache2.

use: http://<IP>.

Exp-8 Build & deploy grid (selenium)

→ Open aws and launch an instances.
grid. [allow both HTTP & HTTPS].
edit inbound and create custom port.
4444
0.0.0.0/0.

→ connect the instance with gitbash using SSH client address.

→ install Docker & Docker compose.

>sudo apt update.
>sudo apt install -y docker.io
>sudo systemctl start docker
>sudo systemctl enable docker.

>Sudo curl -L https://github.com (Url of compose) >
> Sudo chmod +x /usr/local/bin/docker-compose.
> docker --version
> docker-compose --version.

→ Create selenium Grid

> mkdir selenium-grid && cd selenium-grid
> nano docker-compose.yml.
paste your code.

→ start selenium Grid.
>sudo docker-compose up -d
> sudo docker PS [verifies running containers]

Go to browser and search.

http://<IP>:4444/ui

-> If you want run pythonTest, install pytn & selenium.

>sudo apt install python3-venv python3-full -y

> Python3 -m venv venv

>source venv/bin/activate

>pip install selenium

> nano test_grid.py.

Paste code.

> python3 test_grid.py [runs].

exp-9 Amrutha Kubernetes

-> open google cloud console. -> cloud shell.

-> create kubernetes cluster.

>gcloud container clusters list.

>gcloud container clusters create my-cluster\
--zone us-central1-a\
--disk-type=pd-standard.

-> connect cluster.

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

-> view Nodes.

kubectl get nodes.

-> create pod.

kubectl run --image=tomcat webserver

-> view Pod

kubectl get pods. -o wide.

-> creat pod using using yaml.

vim pd-df1.yaml

paste code.

-> open port 8080

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

-> kubectl create -f pd-df1.yaml

-> kubectl get pods -o wide.

nodes

http://<EP>:8080

exp-10 Docker image

-) open aws create an instance
Docker [allow HTTP traffic].

-) open git bash and connect with instance
via SSH client.

> Sudo su - (switch to root user)
> apt update -y

install packages. required.

>apt install nginx -y
>curl -o- <github link>bash >.
> export NVM_DIR ="$HOME/.nvm"
[-s "$NVM_DIR/nvm.sh"] && \. "$NVM_DIR/nvm.sh"
nvm install 22
npm -v
npm install -g pm2
apt update.

Create a node.js application.

cd /home
mkdir node
cd node
nano hello.js

paste code

start app

> node hello.js
run with PM2
> pm2 start hello.js --name app

D) Configure Nginx as Reverse Proxy.

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

Add the code. [server {
listen 80;
server_name.
location
- - -
- -

enable site & restart Nginx:

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

systemctl restart nginx.

install docker & docker compose.

apt install -y docker.io

apt install -y docker-compose

create Dockerfile

cd /home/node

nano Dockerfile

Paste Code

create dockerignore

nano .dockerignore.

node_modules
npm-debug.log

build and Push

docker build -t username/nod-app:latest.

docker login

username:
Password: [enter details] and confirm the code.

docker Push username/nod-app:latest [confirm your image]

Exp-11 prometheus

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

> helm repo update.

> helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace.

> kubectl get pods -n monitoring

> kubectl get svc -n monitoring.

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

Exp-12 Grafana

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

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

> kubectl port-forward svc/my-grafana 3000:80 -n monitoring.

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

> helm repo update.

> helm install prometheus prometheus-community/kube-prometheus-stack --namespace monitoring --create-namespace.

> kubectl get pods -n monitoring