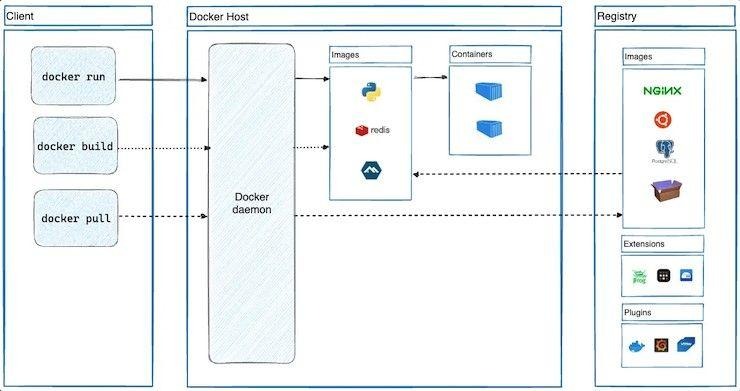
**DEVOPS INTERNAL-2**

**EXP-6:** To understand Docker Architecture and Container Life Cycle, install Docker and execute docker commands to manage images and interact with containers



INSTALL DOCKER:

sudo apt update

sudo apt install docker.io -y

sudo systemctl start docker

sudo systemctl enable docker

docker –version

View Images & Pull Image:

docker images

docker pull ubuntu

Run a Container:

docker run -it --name mycontainer <container\_name>

Check Containers:

docker ps

docker ps -a

Manage Container Life Cycle:

docker stop mycontainer # Stop container

docker start mycontainer # Start container

docker restart mycontainer # Restart container

docker pause mycontainer # Pause container

docker unpause mycontainer # Resume container

Delete Container:

docker stop mycontainer

docker rm mycontainer

docker rmi ubuntu # Remove image (optional)

**EXP-7:** **To learn Docker file instructions, build an image for a sample web application using Docker file.**

Step 1: Create Project Folder and HTML File

mkdir myhtmlapp

cd myhtmlapp

nano index.html (or ) vi index.html

<!DOCTYPE html>

<html>

<head>

<title>Docker HTML App</title>

</head>

<body>

<h1>Hello from Dockerized HTML app!</h1>

</body>

</html>

If using nano: Ctrl + O, Enter, Ctrl + X to save

Step 2: Create Dockerfile

nano Dockerfile or vi Dockerfile

FROM python:3.9-slim

WORKDIR /app

COPY . .

EXPOSE 8000

CMD ["python", "-m", "http.server", "8000"]

If using nano: Ctrl + O, Enter, Ctrl + X to save

Step 3: Build Docker Image:

docker build -t html-python-server .

Step 4: Run Docker Container:

docker run -d -p 9090:8000 html-python-server

can be seen on : http://localhost:9090

Step 5: Check Running Containers

docker ps

Step 7: Stop and Remove Container

docker stop <container\_id\_or\_name>

docker rm <container\_id\_or\_name>

(container\_id or name can be seen using docker ps)

**EXP-8:** To deploy a containerized application on Kubernetes

**1. Install Docker**

sudo apt update

sudo apt install docker.io -y

sudo systemctl start docker

sudo usermod -aG docker $USER

docker run hello-world

**2. Install Minikube**

curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

minikube version

**3. Install kubectl**

curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x kubectl

sudo mv kubectl /usr/local/bin/

kubectl version --client

**4. Start Minikube**

minikube start --driver=docker

minikube status

**5. Create HTML App**

mkdir html-k8s && cd html-k8s

nano index.html

**index.html**

<!DOCTYPE html>

<html>

<head><title>K8s HTML Page</title></head>

<body>

<h1>Hello from Kubernetes on Linux!</h1>

<p>Served by Nginx inside a container.</p>

</body>

</html>

**Dockerfile**

FROM nginx:alpine

COPY index.html /usr/share/nginx/html/index.html

**6. Build Docker Image in Minikube**

eval $(minikube docker-env)

docker build -t html-k8s-page:v1 .

**7. Create Kubernetes Manifests**

**deployment.yaml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: html-page

spec:

replicas: 2

selector:

matchLabels:

app: html-page

template:

metadata:

labels:

app: html-page

spec:

containers:

- name: nginx

image: html-k8s-page:v1

ports:

- containerPort: 80

**service.yaml**

apiVersion: v1

kind: Service

metadata:

name: html-page-service

spec:

selector:

app: html-page

ports:

- protocol: TCP

port: 80

targetPort: 80

type: NodePort

**8. Deploy on Kubernetes**

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

kubectl get pods

kubectl get svc

**9. Access Application**

minikube service html-page-service

# or

minikube ip

kubectl get svc html-page-service

**EXP-9: To install and configure Pull based Software Configuration Management and provisioning tools using Puppet.**

# Step 1: Install Puppet Repository

wget https://apt.puppet.com/puppet8-release-bookworm.deb

sudo dpkg -i puppet8-release-bookworm.deb

sudo apt update

# Step 2: Install Puppet Server

sudo apt install puppetserver -y

# Step 3: Start and Enable Puppet Server

sudo systemctl start puppetserver

sudo systemctl enable puppetserver

sudo systemctl status puppetserver

# Step 4: Configure Puppet Agent

sudo nano /etc/puppetlabs/puppet/puppet.conf

# Step 5: Start and Enable Puppet Agent

sudo systemctl start puppet

sudo systemctl enable puppet

# Step 6: Test Puppet Agent Communication

sudo /opt/puppetlabs/bin/puppet agent –test

**EXP-10: To learn Software Configuration Management and provisioning using Puppet Blocks(Manifest, Modules, Classes, Function)**

# 1. Verify Puppet

puppet --version

# 2. Create module structure

mkdir -p ~/puppet-demo/modules/webserver/{manifests,lib/puppet/functions/webserver}

**Create class manifest**

nano ~/puppet-demo/modules/webserver/manifests/init.pp

class webserver {

package { 'apache2': ensure => installed }

service { 'apache2': ensure => running, enable => true }

file { '/var/www/html/index.html':

ensure => file,

content => "<h1>Hello from Puppet Webserver!</h1>"

}

notify { call\_function('webserver::greet'): }

}

**Create function**

nano ~/puppet-demo/modules/webserver/lib/puppet/functions/webserver/greet.rb

Puppet::Functions.create\_function(:'webserver::greet') do

def greet()

"Webserver setup done!"

end

end

**Create main manifest**

nano ~/puppet-demo/site.pp

include webserver

# 3. Apply Puppet manifest

sudo /opt/puppetlabs/bin/puppet apply ~/puppet-demo/site.pp --modulepath=~/puppet-demo/modules

# 4. Verify Apache and HTML

systemctl status apache2

cat /var/www/html/index.html