**Assignment**

**1.Linux :**

**1.Provide steps to create a directory inside a directory where the parent directory does not exist.**

**A)**open the terminal navigate to the desired location ‘cd’

-cd /path/to/swapna

-mkdir -p swapna/pasupuleti

-ls swapna

**2.How to install a package on a linux server when there is no internet conection?**

A)-For debian based system ubuntu

sudo dpkg -i package\_name.deb

-For Hat -based system centos

sudo dpkg -i package\_name.deb

**3.How to access specific folders of server A from server B and server c**

1. Using various methods like ssh,NFS,SAMB or SCP

Using ssh command-ssh-secure shall

* ssh username@serverA
* NFS-Network file system

Eg: mount serverA:/path/to/shared/folder /mnt/local\_mount\_point

-Samba (SMB/CIFS):

mount -t cifs //serverA/share /mnt/local\_mount\_point -o username=user,password=password

-SCP (Secure Copy Protocol):

scp username@serverA:/path/to/file username@serverB:/path/to/destination

**4.How to check all the running processes from a server?**

**A)**Using commands like-

* Ps aux - Display the all running processes on system.
* top command -This command dynamic real view on the system. It displays the information of cpu,memory and process ids
* htop command : Similar to top, but with a more user-friendly and interactive interface.

**5.Provide the command to delete all the files older than x days inside a specific directory.**

1. To delete all files older than a specific number of days inside a specific directory, you can use the find command along with the rm command.

find /path/to/directory -type f -mtime +X -exec rm {} \;

Find command : Used to search for files and directories within a specified directory hierarchy

/path/to/directory: Specifies the directory where you want to search for files.

type f: Specifies that only regular files should be considered (not directories or other types of files).

-mtime +x: Specifies files modified more than X days ago. Replace x with the desired number of days.

-exec rm {} \;: Executes the rm (remove) command on each file found by find , {} represents the file found, and \; marks the end of the -exe command.

**6.Create a shell script to identify the process ID a.script should as a user input for process ID b. If the process exists, the script should print the process ID and exit c. If the process doesn’t exist script should print the process doesn’t exist and ask for another input**

A)#!/bin/bash

# Prompt the user to enter a process ID

read -p "Enter the process ID: " pid

# Check if the process exists

if ps -p "$pid" > /dev/null; then

echo "Process with ID $pid exists."

else

echo "Process with ID $pid doesn't exist."

# Ask for another input

while true; do

read -p "Enter another process ID or 'exit' to quit: " pid

if [[ "$pid" == "exit" ]]; then

exit 0

elif ps -p "$pid" > /dev/null; then

echo "Process with ID $pid exists."

exit 0

else

echo "Process with ID $pid doesn't exist."

fi

done

fi

chmod +x check\_process.sh: Make the file executable

./check\_process.sh : To run the script . executable the script.

**2.Docker:**

**1. What is docker and why do we need it?**

A)Docker is a software platform that allows you to build,test and deploy applications quickly.Docker packages software into standardized units called containers that have everything the software needs to run including libraries , system tools ,code and runtime.

**2.write a docker file for a sample java/python application.**

1. First you can install python in your system

# Use an official Python runtime as a parent image

FROM python:3.8-slim

# Set the working directory in the container

WORKDIR /app

# Copy the current directory contents into the container at /app

COPY . /app

# Install any needed dependencies specified in requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

# Make port 8080 available to the world outside this container

EXPOSE 8080

# Define environment variable

ENV NAME World

# Run app.py when the container launches

CMD ["python", "app.py"]

**3. What is the docker lifecycle?**

**A)**The Docker lifecycle refers to the sequence of steps involved in the creation, management, and disposal of Docker containers.

1)Image creation : The “docker build” command is used to build the Docker image based on the instructions in the Dockerfile.

2)container initialization: Once an image is built, Docker containers can be instantiated from it.The docker run command is used to create and start a container based on a specific image.

3)container execution: Once initialized, containers execute the commands specified in their Dockerfile or any additional commands provided at runtime.

4)Monitoring and management: The docker ps command lists all running containers, while docker inspect provides detailed information about a specific container.

5)container updates and modifications: Containers can be updated by creating new images with updated configurations or application code.When updating containers, it's common to stop the existing container, pull the latest image, and restart the container with the new image.Docker Compose or container orchestration platforms like Kubernetes can automate the process of updating and scaling containers in a cluster.

6)container Removal : docker stop and docker rm command.

7)Image maintenance and cleanup : docker prune and docker rmi command.

**4.What is the difference between an image and a container?**

**Docker image:​​**An image is a lightweight, standalone, and executable software package that contains everything needed to run a piece of software, including the code, runtime, libraries, dependencies, and configuration files.Docker images are built from a Dockerfile, which contains instructions for assembling the image layers.Images are stored in a registry, such as Docker Hub, and can be shared, versioned, and reused by developers and teams.

**Container :** A container is a runtime instance of a Docker image. It is a lightweight and portable executable environment that encapsulates an application anditsdependencies.Containers are created from Docker images using the “docker run” command. Each container is isolated from other containers and the host system, with its own filesystem, network interfaces, and process space.

**5.How to check docker container logs? Provide the command for the same.**

**-**Docker log <container\_name > –whose logs you want to check.

-docker logs my-container :display the logs of the specified container .

**3.Kubernetes:**

**1.What are different types of services?**

**A)**There 5 types of services:

1.clusterip :

2.Nodeport

3.Loadbalancer

4.External Name

5.Headless

**2. What is a pod?**

A) In Kubernetes, a pod is the smallest deployable unit that represents a single instance of a running process in the cluster. A pod encapsulates one or more containerized applications, storage resources, and networking components

Here some key points about pods in kubernets:

**Atomic Unit:** Pods are the smallest unit of deployment in Kubernetes. They encapsulate one or more containers, storage resources, a unique network IP, and configuration options.

**Grouping Containers:** Pods allow you to group containers that are tightly coupled and need to share resources, such as storage volumes or networking.

**Shared Context:** Containers in a pod share the same context, including network namespace, IPC namespace, and hostname. They can communicate with each other using localhost.

**Single Deployment Unit:** Kubernetes schedules and manages pods as a single unit. If multiple containers need to work together, they are usually deployed within the same pod.

**Lifecycles:** Containers in a pod share the same lifecycle. They are started together, stopped together, and can be scaled together.

**Pods Are Ephemeral:** Pods in Kubernetes are considered ephemeral. They can be created, destroyed, or replaced dynamically based on the needs of the system, such as scaling, rolling updates, or node failures

**3.Create a pod with the above created custom image when a pod dies k8s should automatically restart?**

A) To create a pod in Kubernetes with automatic restart upon failure, you need to define a Pod manifest that includes the necessary specifications, such as the container image, restart policy, and other settings. Assuming you have already created a custom Docker image and it's available in a container registry.

Eg: YAML manifest for creating a pod with automatic restart:

apiVersion: v1

kind: Pod

metadata:

name: my-pod

spec:

containers:

- name: my-container

image: your-custom-image:tag

# Add other container specifications like ports, volume mounts, etc. if needed

restartPolicy: Always

In this manifest:

* API version: Specifies the Kubernetes API version being used.
* Kind: Specifies the resource type, which in this case is a Pod.
* metadata: Contains metadata about the Pod, such as its name.
* spec: Contains the specification for the Pod, including the list of containers.
* containers: Specifies the containers running in the Pod. Here, we define one container named my-container with the specified image.
* restart policy: Specifies the restart policy for the Pod. Setting it to always ensures that Kubernetes will automatically restart the Pod if it terminates for any reason.

kubectl apply -f my-pod.yaml

**4.How to access the custom application with a specific port?**

1. To access a custom application running in a Kubernetes pod with a specific port, you typically need to expose that port either internally within the Kubernetes cluster or externally so that it can be accessed from outside the cluster. Here's how you can do it:

**1.Internal access with in the cluster service (cluster IPservice)**:Your application within the Kubernetes cluster using a ClusterIP service. This allows other pods within the cluster to access your application using the service's cluster-internal IP address.

Eg: apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: <port> # Specify the port your application listens on

targetPort: <targetPort> # Specify the port your application is listening on in the pod

2. External Access (Node port or Load balancer service):If you want to access your application from outside the Kubernetes cluster, you can use a NodePort or LoadBalancer service.apiVersion: v1

**NodePort:** Exposes the service on a static port on each cluster node's IP. You can access the service using <NodeIP>:<Nodeport>.

**LoadBalancer:** Automatically provisions an external load balancer (if supported by your cloud provider) to access the service. It provides a stable IP address or DNS name that can be used to access the service.

Eg:apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: <port> # Specify the port you want to expose externally

targetPort: <targetPort> # Specify the port your application is listening on in the pod

type: NodePort

[4.CI/CD](http://4.ci/CD) :

​​**1.Set up a pipeline (Github actions/gitlab runner / jenkins or any open source tool) to build,test,create a docker image,publish and deploy to k8s.use the application present in this public repo http:**[**github.com/apiwizlabs/wizdesk**](http://github.com/apiwizlabs/wizdesk)**.**

**A)**Setting up a complete CI/CD pipeline involves several steps, including building, testing, creating a Docker image, publishing it to a container registry, and deploying it to Kubernetes. Below, I'll outline the general steps to achieve this using GitHub Actions as the CI/CD tool.

ChatGPT

Setting up a complete CI/CD pipeline involves several steps, including building, testing, creating a Docker image, publishing it to a container registry, and deploying it to Kubernetes. Below, I'll outline the general steps to achieve this using GitHub Actions as the CI/CD tool.

### **GitHub Actions Workflow**

1. **Create GitHub Actions Workflow:**Create a .github/workflows/main.yaml file in your repository with the following content:

name: CI/CD Pipeline

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- name: Checkout Repository

uses: actions/checkout@v2

- name: Set up Docker Buildx

uses: docker/setup-buildx-action@v1

- name: Build and Push Docker Image

uses: docker/build-push-action@v2

with:

context: .

push: true

tags: your-registry/your-image-name:latest

deploy:

needs: build

runs-on: ubuntu-latest

steps:

- name: Configure kubectl

uses: azure/k8s-set-context@v1

with:

kubeconfig: ${{ secrets.KUBE\_CONFIG\_DATA }}

- name: Deploy to Kubernetes

run: |

kubectl apply -f your-kubernetes-manifest.yaml

1. **Dockerfile:**Ensure that your repository contains a Docker fileat the root level to build the Docker image.

2.**Kubernetes Manifest:**Prepare your Kubernetes manifest file(your-kubernents-manifest.yaml) for deploying your application to Kubernetes. This should include the deployment, service, and any other resources necessary for your application.

1. **GitHub Secrets:**Add the KUBE\_CONFIG\_DATA secret to your GitHub repository secrets, which contains the Kubernetes configuration data necessary for deploying to your Kubernetes cluster.
2. **Trigger Workflow:**Push changes to your repository, and the GitHub Actions workflow will automatically trigger, building the Docker image, pushing it to your container registry, and deploying it to Kubernetes.

**2.Automate to spin up a network and virtual machine. Install the Nginx package and start the servielany cloud)**

To automate the process of spinning up a network and virtual machine, installing the Nginx package, and starting the service, you can use Infrastructure-as-Code (IaC) tools like Terraform in combination with configuration management tools like Ansible or shell scripts. Below, I'll outline a basic approach using Terraform to provision the infrastructure and Ansible to configure the VM.

### **Terraform for Infrastructure Provisioning:**

1. **Network and Virtual Machine Provisioning:**You'll create a Terraform configuration to define your network and virtual machine. Below is an example ‘main.tf’

provider "google" {

project = "your-project-id"

region = "us-central1"

}

resource "google\_compute\_network" "network" {

name = "example-network"

}

resource "google\_compute\_instance" "vm\_instance" {

name = "example-vm"

machine\_type = "n1-standard-1"

zone = "us-central1-a"

boot\_disk {

initialize\_params {

image = "debian-cloud/debian-10"

}

}

network\_interface {

network = google\_compute\_network.network.id

access\_config {

// Ephemeral IP

}

}

}

**2.Ansible for Configuration Management:**After provisioning the VM, you can use Ansible to install Nginx and start the service. Create an Ansible playbook ‘ngix.nf’

---

- name: Install Nginx

hosts: example-vm

become: true

tasks:

- name: Update apt cache

apt:

update\_cache: yes

- name: Install Nginx

apt:

name: nginx

state: present

- name: Start Nginx service

service:

name: nginx

state: started

**Running Terraform and Ansible:**You can use a shell script to orchestrate the Terraform and Ansible steps:

#!/bin/bash

# Provision infrastructure with Terraform

terraform init

terraform apply -auto-approve

# Run Ansible playbook to configure VM

ansible-playbook -i <vm-ip>, nginx.yml

echo "Nginx installed and started on the VM."

### **Execution:**

1. Save the Terraform configuration in a file named main.tf, the Ansible playbook in nginx.yml, and the shell script to orchestrate them.
2. Ensure you have Terraform and Ansible installed on your local machine.
3. Run the shell script to execute the provisioning and configuration steps.