Let's address each question step by step:  
  
**Linux :**

1. Create a directory inside a directory where the parent directory does not exist

mkdir -p /path/to/nonexistent\_parent/new\_directory

2. Install a package on a Linux server when there is no internet connection

sudo dpkg -i /path/to/package.deb

3. Access specific folders of Server A from Server B and Server C

You can use ssh (secure shell) to access Server A from Server B or Server C. Here’s an example command to access a folder (/path/to/folder) on Server A from Server B:

ssh username@serverA 'ls /path/to/folder'

4. Check all the running processes from a server

Use the ps command to list all running processes:

ps aux

5. Delete all the files older than X days inside a specific directory

find /path/to/directory -type f -mtime +30 -exec rm {} \;

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 asks for another input  
  
Here is a shell script that performs the requested operations:

```bash

#!/bin/bash

# Function to check if a process exists

check\_process() {

    if ps -p $1 > /dev/null 2>&1

    then

        echo "Process ID $1 exists."

        exit 0

    else

        echo "Process ID $1 does not exist."

# Main loop to continuously ask for input until a valid process ID is found

while true

do

    read -p "Enter process ID: " pid

    check\_process $pid

done

```

Save this script to a file, for example `check\_process.sh`, and make it executable with the following command:

```sh

chmod +x check\_process.sh

```

To run the script, use:

```sh

./check\_process.sh

```

The script will prompt the user to enter a process ID, check if the process exists, and provide appropriate feedback. If the process does not exist, it will continue to prompt the user until a valid process ID is provided.

**Docker :**

1. What is Docker and why do we need it?

Docker is a platform that allows you to develop, deploy, and run applications inside containers. Containers are lightweight, portable, and isolated environments that package everything needed to run an application (code, runtime, libraries, and dependencies).

Reasons we need Docker:

Consistency: Docker ensures applications run the same way everywhere, regardless of the environment.

Isolation: Containers provide a way to isolate applications and their dependencies, preventing conflicts.

Efficiency: Containers share the host OS kernel, making them lightweight compared to virtual machines.

Portability: Docker containers can run on any machine that has Docker installed, simplifying deployment and scaling.

DevOps practices: Docker supports modern DevOps practices like continuous integration, delivery, and deployment.

2. Write a Dockerfile for a sample Java/Python application.

Python Example:

dockerfile

# Use an official Python runtime as a parent image

FROM python:3.9-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

# Run app.py when the container launches

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

3. What is the Docker lifecycle?

The Docker lifecycle involves several key stages for managing containers:

Create: Define and build Docker images using Dockerfiles.

Manage: Run, stop, restart, and delete Docker containers.

Share: Push and pull Docker images from registries like Docker Hub.

Deploy: Distribute and deploy Dockerized applications to various environments.

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

Image: An image is a read-only template that contains the application and its dependencies. It includes everything needed to run an application (code, runtime, libraries, and environment variables).

Container: A container is a runtime instance of an image. It is a lightweight, executable package that includes everything needed to run a piece of software, including the application code and all its dependencies. Containers run isolated processes in the host OS.

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

docker logs <container\_name\_or\_id>

**Kubrenetes:**

1. What are different types of services?

In Kubernetes, services are used to expose your application to external or internal network traffic. There are several types of services:

ClusterIP: Exposes the service on a cluster-internal IP. This type makes the service only reachable within the cluster.

NodePort: Exposes the service on each Node's IP at a static port. This means the service is accessible via <NodeIP>:NodePort from outside the cluster.

LoadBalancer: Exposes the service externally using a cloud provider's load balancer. The external IP is provisioned and traffic is routed to the service.

ExternalName: Maps the service to the contents of the externalName field (e.g., my.database.example.com), allowing access to external services by DNS.

2. What is a pod?

A pod is the smallest deployable unit in Kubernetes. It represents a single instance of an application, along with its associated containers and shared resources (like volumes and network IP). Pods are ephemeral and can be scheduled and rescheduled by Kubernetes to run on nodes in the cluster.

3. Create a pod with a custom image that restarts automatically when it dies

To create a pod with a custom image and ensure it restarts automatically when it fails, you need to define a Kubernetes Deployment. A Deployment manages a ReplicaSet of pods, ensuring a specified number of pod replicas are running at any given time.

Here's an example YAML configuration for a Deployment:

yaml

Copy code

apiVersion: apps/v1

kind: Deployment

metadata:

name: my-custom-app

spec:

replicas: 1 # Number of pod replicas

selector:

matchLabels:

app: my-custom-app

template:

metadata:

labels:

app: my-custom-app

spec:

containers:

- name: my-custom-container

image: your\_registry/your\_image:tag # Replace with your custom image details

ports:

- containerPort: 8080 # Port your application listens on

Apply this configuration using kubectl apply -f deployment.yaml. Kubernetes will ensure that one instance of your custom image (your\_image:tag) is always running. If the pod crashes or terminates, Kubernetes will automatically restart it.

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

To access your custom application running inside a Kubernetes cluster with a specific port, you typically use a Service with either a NodePort or LoadBalancer type.

NodePort Service Example:

yaml

Copy code

apiVersion: v1

kind: Service

metadata:

name: my-custom-app-service

spec:

type: NodePort

ports:

- port: 8080 # Port exposed by the service

targetPort: 8080 # Port your application listens on inside the pod

nodePort: 30000 # Specify a port number (30000-32767) to access the service externally

selector:

app: my-custom-app # Selector matching your Deployment's pod labels

**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 [https://github.com/apiwizlabs/wizdesk](https://github.com/apiwizlabs/wizdesk" \t "_blank).

pipeline {

agent any

stages {

stage('Git Clone') {

steps {

git '(git@github.com:akhila40/APIwiz.git)'

}

}

stage('Build Docker Image') {

steps {

dir('APIwiz') {

sh 'docker build -t my-image:latest .'

}

}

}

stage('Publish Docker Image') {

steps {

sh 'docker tag my-image:latest ak407/my-image:latest'

sh 'docker push ak407/my-image:latest'

}

}

}

}

Tried but I did not get the exact o/p

**K8’s YAML file**

apiVersion: apps/v1

kind: Deployment

metadata:

name: image-name

spec:

replicas: 3

selector:

matchLabels:

app: image-name

template:

metadata:

labels:

app: image-name

spec:

containers:

- name: image-name

image: your-docker-hub-ak407/image-name:latest

ports:

- containerPort: 80

2. Automate to spin up a network and virtual machines. Install the Nginx package and start the service(any cloud) tried but did not got output

provider "aws" {

  region = "ap-south-1"

  access\_key = "AKIAU6GDYOO4JCFJOGCO"

  secret\_key = "U9vacA66V9enFeyGetkZJBewXP7gwHEcSMOp3le/"

}

module "aws\_instance" {

  source = "terraform-aws-modules/ec2-instance/aws"

  name = "my-ec2"

  ami = "ami-01376101673c89611"

  instance\_type = "t2.micro"

  key\_name = "Anvith.pem"

  monitoring = false

  subnet\_id = "subnet-0d04dc28408d745f3"

  vpc\_security\_group\_ids = ["sg-09916bcf961478b20"]

  associate\_public\_ip\_address = true

}

resource "null\_resource" "example" {

  connection {

    type = "ssh"

    host = module.aws\_instance.public\_ip

    user = "ubuntu"

  private\_key = file("C:\\Users\\Sulake\\Desktop\\PREP\\terraform1")

  }

  provisioner "remote-exec" {

    inline = [

      "sudo apt update",

      "sudo apt install nginx -y",

      "sudo systemctl start nginx",

      "sudo systemctl enable nginx"