Session 1:

1. Introduction to Virtualization

Virtualization ek technique hai jisme ek physical machine ko multiple virtual machines (VMs) mein divide kiya jaata hai. Yeh process ek layer of abstraction create karta hai, jisse ek single physical server par multiple operating systems run kar sakte hain, bina ek dusre se interfere kiye.

- **Example**: Agar aapke paas ek powerful physical server hai, toh aap usse multiple virtual machines (VMs) create kar sakte hain. Har VM apna independent OS aur applications run karega.

2. Types of Hypervisors

Hypervisor ek software layer hota hai jo virtual machines ko manage karta hai. Hypervisor ke do types hote hain:

1. **Type 1 Hypervisor (Bare-metal Hypervisor)**:

- Yeh directly hardware par run karta hai. Ismein host OS ki zarurat nahi hoti.
- Example: VMware ESXi, Microsoft Hyper-V, Xen.

1. **Type 2 Hypervisor (Hosted Hypervisor)**:

- Yeh ek existing OS ke upar run karta hai.
- Example: VMware Workstation, Oracle VirtualBox.

3. Types of Multi-Tenancy in Cloud Computing

Multi-tenancy ek architecture model hai jisme ek single instance of software aur infrastructure multiple users (tenants) ke liye share kiya jaata hai. Ismein 3 primary types hain:

1. **Single-Tenant**:

- Har user ke liye separate infrastructure ho. Example: Ek server per sirf ek customer ka data aur applications hote hain.

1. **Multi-Tenant**:

- Ek hi infrastructure ya instance mein multiple customers ke data aur resources hoti hain. Yahan par resources efficiently share kiye jaate hain.
- Example: Cloud providers jaise AWS, Azure, jahan par ek hi server multiple users ko serve karta hai.

1. **Hybrid Multi-Tenant**:

- Yeh combination hota hai single aur multi-tenant architecture ka. Kuch resources shared hote hain, aur kuch private hote hain.
- Example: Cloud hosting services jo private aur public cloud ka mix use karte hain.

4. Need of Virtualization Provisioning

Virtualization provisioning ka matlab hai virtual resources ka allocation, jisme physical resources ko virtual machines ko allocate kiya jaata hai. Virtualization ki need kuch specific reasons ki wajah se hoti hai:

1. **Efficient Resource Utilization**:

- Physical resources ko efficiently utilize karne ke liye virtual machines create ki jaati hain, jo ek machine ko multiple environments mein divide karti hain.

1. **Cost Efficiency**:

- Ek single physical server par multiple VMs run karne se hardware cost save hoti hai.

1. **Flexibility and Scalability**:

- Virtualization se systems ko easily scale kiya ja sakta hai aur naye VMs ko quickly deploy kiya ja sakta hai.

Session 2:

1. Introduction to Cloud

Cloud Computing ek technology hai jisme data aur applications internet (cloud) par store kiye jaate hain, aur users ko un tak remotely access dene ka kaam kiya jaata hai. Cloud computing me users ko physical servers, storage, aur applications ki zarurat nahi hoti, kyunki yeh sab services cloud providers ke through available hoti hain.

2. Advantages of Cloud

Cloud computing ke kuch main advantages hain:

1. **Cost Efficiency**:

- Traditional infrastructure ke comparison mein cloud computing bahut cost-effective hoti hai kyunki aapko hardware aur maintenance ka kharcha nahi uthana padta.

1. **Scalability**:

- Cloud resources ko easily scale kiya ja sakta hai. Agar aapko zyada resources chahiye toh aap cloud provider se unko instant access kar sakte hain.

1. **Accessibility**:

- Aap kahin se bhi apne cloud data aur applications ko access kar sakte hain, bas internet connection hona zaroori hai.

1. **Reliability**:

- Cloud providers redundancy aur failover systems provide karte hain, jisse data loss aur downtime ki chances kam ho jaati hain.

3. Cloud Types and Models

Cloud computing ke main types aur models hain:

1. **Private Cloud**:

- Yeh ek single organization ke liye reserved hota hai. Saare resources ek single organization ke control mein hote hain.
- Example: Company ka internal cloud infrastructure.

1. **Public Cloud**:

- Yeh cloud infrastructure general public ke liye available hota hai aur multiple organizations resources share karte hain.
- Example: AWS, Google Cloud, Microsoft Azure.

1. **Hybrid Cloud**:

- Yeh private aur public cloud ka combination hota hai. Kuch resources private cloud pe aur kuch public cloud pe host kiye jaate hain.
- Example: Aapke critical applications private cloud pe ho sakte hain, aur baaki general workloads public cloud pe run ho rahe hote hain.

1. **Community Cloud**:

- Yeh cloud infrastructure ek specific community ke liye hota hai, jaise koi particular industry ya interest group.
- Example: Government institutions ka shared cloud infrastructure.

4. Cloud Service Providers

Cloud Service Providers woh companies hoti hain jo cloud-based services offer karte hain. Kuch well-known cloud service providers hain:

1. **Amazon Web Services (AWS)**
2. **Microsoft Azure**
3. **Google Cloud Platform (GCP)**
4. **IBM Cloud**
5. **Oracle Cloud**

5. SAAS, PAAS, IAAS

Cloud computing mein 3 main service models hote hain:

1. **SAAS (Software as a Service)**:

- Yeh fully managed software hota hai jo internet ke through access kiya jaata hai. Users ko installation, maintenance, aur updates ka tension nahi hota.
- Example: Gmail, Microsoft 365.

1. **PAAS (Platform as a Service)**:

- Yeh platform provide karta hai jisme developers apne applications ko build, test, aur deploy kar sakte hain bina infrastructure ke bare mein soche.
- Example: Google App Engine, Heroku.

1. **IAAS (Infrastructure as a Service)**:

- Yeh virtualized computing resources provide karta hai, jaise virtual machines, storage, aur networking. Users ko apna operating system, applications, aur data manage karna hota hai.
- Example: AWS EC2, Google Compute Engine.

Session 3-5:

1. Create Amazon EC2 Instance

Amazon EC2 (Elastic Compute Cloud) ek service hai jo aapko scalable virtual machines (instances) provide karti hai. EC2 instance create karne ke liye:

1. **AWS Management Console** pe login karke EC2 service choose karo.
2. **Launch Instance** option select karo.
3. Apne desired operating system (like Linux or Windows) ka selection karo.
4. Instance type choose karo (small, medium, large depending on your requirements).
5. **Security groups** configure karo (firewall rules for your instance).
6. **Key pair** generate karo (for secure SSH/RDP login).
7. Launch the instance.

2. Create AWS S3 Bucket

AWS S3 (Simple Storage Service) ek object storage service hai jisme aap data store kar sakte hain.

1. **AWS Management Console** pe login karo.
2. **S3** service pe jao aur **Create Bucket** option select karo.
3. Bucket ka unique name choose karo aur region select karo.
4. Configure additional settings if needed (like versioning, encryption).
5. **Create** pe click karke bucket create kar lo.

3. Create AWS Lambda

AWS Lambda ek compute service hai jo aapko serverless applications run karne ka option deti hai.

1. **AWS Management Console** pe login karo.
2. **Lambda** service ko search karo aur **Create Function** pe click karo.
3. Function ko name do aur runtime choose karo (like Python, Node.js, etc.).
4. Function ka code likho ya upload karo.
5. **Configure triggers** (like S3 event, DynamoDB event).
6. Function ko **Test** karo aur execute karo.

4. Create AWS VPC

AWS VPC (Virtual Private Cloud) ek isolated network environment hai jo aapke AWS resources ko securely host karne ke liye use hota hai.

1. **AWS Management Console** pe login karo.
2. **VPC** service ko search karo aur **Create VPC** option pe click karo.
3. CIDR block define karo (IP range jo aap use karna chahte hain).
4. Subnets create karo (private aur public subnets).
5. **Internet Gateway** attach karo agar aapko public internet access chahiye ho.
6. **Route tables** configure karo.
7. **Security groups** aur **network ACLs** set karo for security.
8. **Create VPC** pe click karke VPC create kar lo.

Summary:

1. **Session 1**: Virtualization, Hypervisors, Cloud Computing aur Virtualization Provisioning ko samjha.
2. **Session 2**: Cloud computing, cloud service models (SAAS, PAAS, IAAS) aur cloud providers ke bare mein jaana.
3. **Session 3-5**: AWS services (EC2, S3, Lambda, VPC) ko create karna seekha.

Session 6-7: Overview of Process Automation (DevOps)

1. Overview of Process Automation (DevOps)

DevOps ek software development aur IT operations ka culture aur practice hai, jisme development (Dev) aur operations (Ops) teams ko closely collaborate karne ki approach di jaati hai. Iska goal hota hai fast aur efficient software development aur deployment process ko automate karna, jisse quality aur delivery speed improve ho sake.

Process Automation ka matlab hai repetitive tasks ko automate karna, taaki manual intervention ki zarurat na pade aur processes efficient ho sake. DevOps mein process automation bahut important hai, aur iske through aap:

- **Continuous Integration (CI)**: Automatically code changes ko integrate karte ho aur testing bhi automate karte ho, jisse development aur testing ka process fast ho jata hai.
- **Continuous Deployment (CD)**: Automatically code ko deploy karte ho production environment mein bina manual intervention ke.
- **Monitoring and Logging**: Automatic monitoring aur logs generate karte ho, taaki aap issues ko jaldi identify kar sakein.

DevOps mein **Automation** kaafi important hota hai, kyunki yeh speed aur quality ko balance karta hai, aur errors ko reduce karta hai.

Example: Agar aap code changes kar rahe hain, toh automatically testing aur deployment ho sake, bina kisi manual process ke. Yeh DevOps ka ek example hai.

Session 7: Version Control with Git

1. What is Git?

Git ek **distributed version control system** hai jo software development mein use hota hai. Git ka use aap apne code changes ko track karne ke liye karte hain, jisse aap easily kisi bhi code version pe revert kar sakein, aur multiple developers bhi ek hi project pe efficiently kaam kar sakein.

Key Features of Git:

- **Distributed**: Git ka har developer ke paas apna local copy hota hai, jisme wo apne changes kar sakta hai aur baad mein unhe sync kar sakta hai.
- **Track Changes**: Git har change ko track karta hai, jisse aap kisi bhi time pe kisi bhi file ko purane version pe le jaa sakte hain.
- **Branching and Merging**: Git allows you to create branches for different features or fixes and later merge them back into the main codebase.

2. Branching and Merging in Git

Branching aur **merging** Git ke important concepts hain, jo aapko multiple features pe kaam karte waqt code ko manage karne mein madad karte hain.

Branching:

- **Branch** ek separate working environment hota hai, jisme aap apne code changes karte hain bina main codebase ko affect kiye.
- **Main Branch**: By default, Git mein ek **main** ya **master** branch hoti hai, jo stable version ko represent karti hai.
- Jab aapko koi new feature ya fix karna hota hai, toh aap ek **new branch** create karte hain.
- **Command Example**:
- `git branch <branch-name>`: Create a new branch.
- `git checkout <branch-name>`: Switch to the created branch.
- `git branch`: Show all branches.

Merging:

- Jab aapka kaam ek branch pe complete ho jata hai, toh aap **merge** karte ho us branch ko main branch ke saath, taaki aapke changes main project mein include ho sakein.
- **Merge conflict** tab hoti hai jab dono branches mein same file ko different tarike se modify kiya gaya ho, aur Git ko decide karne mein dikkat ho.
- **Command Example**:
- `git merge <branch-name>`: Merge a branch into the current branch.
- Agar conflict hota hai, toh Git aapko notify karega, aur aapko manually conflict resolve karna padta hai.

Example:

- Suppose aap feature1 branch par kaam kar rahe ho aur main branch mein bhi kaafi updates aaye hain. Jab aap feature1 ko main branch mein merge karenge, toh agar dono branches mein same file ko modify kiya gaya ho, toh Git aapko **merge conflict** batayega, jise aap manually resolve karenge.

3. Git Workflows

Git workflows ek set of rules aur guidelines hain jo teams ko define karte hain ki kaise unhe code ko collaboratively manage karna hai.

Kuch common **Git workflows** hain:

1. **Centralized Workflow**:

- Is workflow mein ek single central repository hoti hai, jisme sab developers apne changes push karte hain.
- Simple aur small teams ke liye suitable hai.

1. **Feature Branch Workflow**:

- Har feature ke liye ek separate branch banayi jaati hai.
- Yeh workflow aapko features ko isolate karne mein madad karta hai, jisse main codebase stable rehta hai.

1. **Gitflow Workflow**:

- Gitflow ek branching model hai jisme multiple types of branches hote hain (e.g., main, develop, feature, release, hotfix).
- Yeh complex projects ke liye suitable hota hai, jisme development aur release process clearly defined hoti hai.

1. **Forking Workflow**:

- Yeh workflow open-source projects ke liye suitable hai. Developer apne forked version par kaam karta hai aur changes ko upstream repository mein pull request ke through merge karta hai.

Example:

- Aap **feature-branch workflow** use kar rahe ho. Aap ek feature branch create karte ho, usme apne changes karte ho, aur phir merge karte ho jab kaam complete ho jata hai.

Important Git Commands for Workflows:

- `git pull`: Remote repository se latest changes fetch karta hai aur merge karta hai.
- `git push`: Local changes ko remote repository mein push karta hai.
- `git fetch`: Remote changes ko download karta hai bina merge kiye.
- `git rebase`: Ek branch ke changes ko dusri branch ke upar apply karta hai, jisse merge history clean hoti hai.

Summary:

1. **DevOps and Process Automation**:

- DevOps mein software development aur operations ko automate kiya jaata hai. Yeh efficiency, speed, aur quality ko improve karta hai.

1. **Version Control with Git**:

- Git ek distributed version control system hai jo code changes ko track karta hai aur multiple developers ko ek project par kaam karne ka option deta hai.

1. **Branching and Merging in Git**:

- Branching se aap ek separate working environment create kar sakte hain, aur merging se aap changes ko main branch mein integrate karte hain.

1. **Git Workflows**:

- Different workflows (Centralized, Feature Branch, Gitflow, Forking) use kiye jaate hain taaki team ke collaboration ko streamline kiya ja sake.

Session 8-11: Docker Introduction & Key Concepts

1. Introducing Docker, What it Does, and Why to Use It?

Docker ek platform hai jo application ko containerized form mein package karne aur run karne mein madad karta hai. Matlab, aap apne application ko ek isolated environment (container) mein run kar sakte hain, jo har machine par exactly same behavior show karega, chahe woh development machine ho, testing environment ho, ya production server ho.

What Docker Does:

- Docker containers applications ko encapsulate karte hain, jisme saari dependencies, libraries, aur configurations hoti hain jo us application ko run karne ke liye zaroori hoti hain.
- Docker ek lightweight aur portable environment provide karta hai jisme applications ko run kiya jaata hai bina kisi conflicts ke.

Why Use Docker?

1. **Portability**: Docker containers ko kisi bhi machine ya environment mein run kiya ja sakta hai, chahe woh Linux, Windows, ya macOS ho.
2. **Isolation**: Docker containers ek dusre se isolated hote hain, isliye ek container ke andar hone wale changes dusre containers ko affect nahi karte.
3. **Speed**: Docker containers bahut lightweight hote hain, aur applications ko start karne ka time bahut kam hota hai.
4. **Consistency**: Docker ensures ki har environment mein same version of application run ho, jisse deployment aur testing processes consistent rehte hain.
5. **Scalability**: Docker aapko easy scaling provide karta hai. Agar aapko more instances chahiye, toh aap easily aur quickly new containers deploy kar sakte hain.

2. Docker Images and Containers

- **Docker Image**:

- Docker image ek read-only template hoti hai jo application aur uske dependencies ko define karti hai. Image ek blueprint hai, jisme application ke saare instructions aur configurations stored hoti hain.
- **Example**: Agar aap ek Python application run karna chahte hain, toh aapko Python ke required version ke saath ek Docker image create karni padegi.
- **Docker Container**:
- Docker container ek running instance hota hai ek image ka. Jab aap Docker image ko run karte hain, toh wo ek container ban jaata hai.
- **Example**: Agar aap Python image ko run karte hain, toh ek container banega jisme Python application run ho sakta hai.

Key Difference:

- Image ek template hai, aur container us image ka running instance hai.
- Image read-only hoti hai, container writable hota hai (container ke andar changes kiye jaa sakte hain).

Commands:

- `docker images`: List all Docker images.
- `docker run <image-name>`: Run a container from a Docker image.
- `docker ps`: List running containers.

3. Dockerfile

Dockerfile ek text file hoti hai jo Docker image create karne ke liye instructions define karti hai. Is file mein aap specify karte hain ki aapko kis base image se start karna hai, kis application ko install karna hai, aur container ke andar kaunsa environment setup karna hai.

Basic Dockerfile Example:

Session 12-14: Container Orchestration and Kubernetes Basics

1. Container Orchestration

Container Orchestration ek process hai jisme multiple containers ko automate tarike se deploy, manage, scale, aur network kiya jata hai. Jab aapke paas bohot saare containers hote hain, toh unhe efficiently manage karna bahut zaroori ho jata hai. Yaha par **Kubernetes** aur **Docker Swarm** jaise tools ka use kiya jata hai.

Why Container Orchestration?

- **Scaling**: Automatically containers ko scale karna jab load increase ho.
- **Management**: Automatically containers ko manage karna (health checks, auto-restarts).
- **Networking**: Containers ko apas mein communicate karne ke liye ek unified network setup karna.
- **Fault Tolerance**: Agar ek container fail hota hai, toh automatically naya container create karna.

Popular Orchestration Tools:

- **Kubernetes**: Ye most widely used tool hai jo complex containerized applications ko manage karta hai.
- **Docker Swarm**: Docker ka built-in orchestration tool hai, lekin Kubernetes ke comparison mein kam features provide karta hai.

2. Architecture and Fundamentals of Kubernetes

Kubernetes ek open-source container orchestration platform hai jo containers ko manage karne mein madad karta hai. Kubernetes aapko containerized applications ko automate tarike se deploy, scale, aur manage karne ka option deta hai.

Key Concepts:

- **Master Node**: Kubernetes ka control plane hota hai. Isme ek ya zyada master nodes ho sakte hain jo cluster ko manage karte hain. Master node manage karta hai:
- **API Server**: Client requests ko process karta hai aur resource updates ko manage karta hai.
- **Scheduler**: Pods ko nodes par schedule karta hai.
- **Controller Manager**: System ke health ko monitor karta hai aur ensure karta hai ki desired state match ho.
- **etcd**: Configuration data ko store karta hai.
- **Worker Nodes**: Ye nodes wo machines hoti hain jaha par containers run karte hain. Worker node me do main components hote hain:
- **Kubelet**: Har node par running agent hai jo containers ko manage karta hai.
- **Kube Proxy**: Networking ko manage karta hai aur services ke andar traffic route karta hai.

Kubernetes Cluster: Ek cluster mein ek ya zyada master nodes aur worker nodes hote hain. Ye cluster containerized applications ko scale, deploy, aur manage karta hai.

3. Pods, Replicaset, Deployment, Service

- **Pod**:
- **Pod** ek basic unit hai Kubernetes mein. Pod ek ya zyada containers ka group ho sakta hai jo ek common storage aur network share karte hain.
- Pods ko commonly single application run karne ke liye use kiya jata hai.
- **Example**: Agar aapko web server aur database ko ek hi machine pe run karna ho, toh aap unhe ek pod ke andar daal sakte hain.
- **ReplicaSet**:
- **ReplicaSet** ek controller hai jo ensure karta hai ki aapke pods ka desired number hamesha running state mein ho.
- Agar pod fail hota hai, toh ReplicaSet automatically ek naya pod launch kar deta hai.
- **Example**: Agar aapne replicaSet ko 3 pods ke saath configure kiya hai, toh yeh ensure karega ki hamesha 3 pods available ho.
- **Deployment**:
- **Deployment** ek higher-level concept hai jo ReplicaSets ko manage karta hai.
- Deployment ko use karke aap easily applications ko upgrade ya rollback kar sakte hain.
- **Example**: Agar aap apne web application ko update karte hain, toh Deployment automatically new version ke pods ko schedule karega.
- **Service**:
- **Service** ek abstraction hai jo pods ke group ko expose karti hai aur unke beech communication ko manage karti hai.

- Service ek stable IP address aur DNS name provide karta hai, jisse pods ke beech communication easily ho sake.
- **Types of Services**:
- **ClusterIP**: Internal communication ke liye, cluster ke andar hi accessible hota hai.
- **NodePort**: External traffic ko cluster ke ek specific node port se forward karta hai.
- **LoadBalancer**: External load balancer ka use karke traffic distribute karta hai.

4. Kubectl Commands

kubectl Kubernetes ka command-line tool hai jo aapko clusters ko manage karne mein madad karta hai. Yaha kuch commonly used **kubectl commands** diye gaye hain:

- **Get Cluster Information**:
- `kubectl cluster-info`: Cluster ke baare mein information deta hai.
- **Get Pods**:
- `kubectl get pods`: Cluster mein sabhi pods ki list dikhata hai.
- `kubectl get pods --all-namespaces`: Sabhi namespaces ke pods ki list dikhata hai.
- **Create Resources**:
- `kubectl create -f <file>.yaml`: Kubernetes resource create karta hai YAML file ke through.
- **Get Services**:
- `kubectl get services`: Cluster mein available services ki list dikhata hai.
- **Scaling**:
- `kubectl scale deployment <deployment-name> --replicas=<number>`: Deployment ko scale karta hai (e.g., 3 replicas).
- **Describe Resources**:
- `kubectl describe pod <pod-name>`: Pod ke detailed information ko show karta hai.
- **Apply Changes**:
- `kubectl apply -f <file>.yaml`: Changes ko apply karta hai jo aapne YAML file mein define kiye hain.
- **Delete Resources**:
- `kubectl delete -f <file>.yaml`: Resources ko delete karta hai jo aapne YAML file mein define kiye hain.

5. Kubernetes using YAML

YAML (Yet Another Markup Language) ek human-readable data serialization format hai jo Kubernetes mein resource configuration ke liye use hota hai. Aap YAML files likhkar Kubernetes ko batate hain ki kaunse resources create karne hain (pods, services, deployments, etc.).

Example: Pod Configuration in YAML:

Session 15-18: Jenkins Basics, Integration with GitHub, and Docker

1. Introduction to Jenkins

Jenkins ek open-source automation tool hai jo mainly continuous integration (CI) aur continuous delivery (CD) ke liye use hota hai. Jenkins ke through aap apne software development lifecycle ko automate kar sakte hain, jisme code build, test, aur deployment processes shamil hote hain.

Jenkins Features:

- **Continuous Integration (CI)**: Jenkins code ko automatically pull karta hai, build karta hai, aur test karta hai jab bhi koi code changes hoti hain.
- **Continuous Delivery (CD)**: Jenkins automate karta hai deployment process ko, jisse application ko faster aur error-free tarike se production mein deploy kiya ja sakta hai.
- **Plugins**: Jenkins mein bohot saare plugins available hote hain, jo Jenkins ko integrate karne mein help karte hain dusre tools jaise Git, Docker, etc.

Why Jenkins?

- Easy setup and configuration.
- Highly extensible with plugins.
- Supports distributed builds.
- Supports integration with various tools like GitHub, Docker, Maven, etc.

2. Installing Jenkins

Jenkins ko install karna bohot easy hai. Aap Jenkins ko apne system pe manually install kar sakte hain ya phir Docker ke through bhi run kar sakte hain. Main method installation ke liye:

For Ubuntu/Debian:

1. Install Java: