# 1.2. Student Handout

# Student Handout: Introduction to Cloud Orchestration - Kubernetes

---

## Cloud Orchestration

**Definition**: Cloud Orchestration is the process of automating and coordinating various tasks and services in the cloud.

**Examples**:

1. Automating the deployment of a web application across multiple cloud servers.
2. Coordinating data backup processes across different cloud storage services.
3. Managing the scaling of resources for an e-commerce platform during peak shopping seasons.

---

## Containers vs. Virtual Machines (VMs)

### Virtual Machines (VMs)

- **Definition**: A Virtual Machine is a software emulation of a physical computer, running its own operating system and applications.

**Examples**:

1. Running a Windows VM on a Linux host to use Windows-specific applications.
2. Using VMs to isolate development, testing, and production environments.
3. Deploying multiple VMs on a single server to maximize hardware utilization.

### Containers

- **Definition**: Containers are lightweight, portable units that package an application and its dependencies, sharing the host OS kernel.

**Examples**:

1. Deploying a microservices architecture using Docker containers.
2. Running a Python application in a container with all its dependencies.
3. Using containers to ensure consistent environments across development, testing, and production.

---

# Why Kubernetes?

**Purpose**: Kubernetes is a tool that helps manage containers by automating deployment, scaling, and operations of application containers.

**Examples**:

1. Automatically scaling a web application based on user demand.
2. Restarting failed containers to ensure application availability.
3. Distributing network traffic evenly across multiple instances of an application.

---

# Kubernetes Architecture

## Control Plane (Master Node)

- **Definition**: The control plane is responsible for managing the Kubernetes cluster, making decisions about scheduling and responding to cluster events.

**Examples**:

1. Scheduling a new pod on an available worker node.
2. Monitoring the health of the cluster and initiating recovery actions.
3. Managing the configuration and state of the cluster using etcd.

## Worker Nodes

- **Definition**: Worker nodes are the machines where application containers run.

**Examples**:

1. Running a set of containers for a web service on a worker node.
2. Hosting a database container on a dedicated worker node for performance.

3. Distributing application components across multiple worker nodes for redundancy.

---

# Key Components of Kubernetes

## Nodes

- **Definition**: A node is a machine (physical or virtual) that runs containers.

**Examples**:

1. A physical server running as a worker node in a data center.
2. A virtual machine in the cloud acting as a Kubernetes node.
3. A local development machine configured as a single-node Kubernetes cluster.

## Pods

- **Definition**: A pod is the smallest deployable unit in Kubernetes, encapsulating one or more containers.

**Examples**:

1. A pod running a single container for a web server.
2. A pod with multiple containers for a web application and its logging service.
3. A pod running a batch processing job with a single container.

## Deployments

- **Definition**: A deployment defines the desired state for pods and manages their lifecycle.

**Examples**:

1. Specifying a deployment to run three replicas of a web application pod.
2. Updating a deployment to roll out a new version of an application.
3. Scaling a deployment to handle increased traffic by adding more pod replicas.

## Services

- **Definition**: A service provides a stable endpoint for accessing a set of pods.

**Examples**:

1. Exposing a web application to external users via a LoadBalancer service.
2. Creating a ClusterIP service for internal communication between microservices.
3. Using a NodePort service to access an application from outside the cluster.

---

# Control Plane and Worker Nodes

## Control Plane Components

- **Kubernetes API Server**: Front-end for the Kubernetes control plane.
- **Scheduler**: Assigns pods to nodes based on resource availability.
- **Controller Manager**: Ensures the cluster's desired state matches the actual state.

## Worker Node Components

- **kubelet**: Communicates with the control plane to manage containers.
- **Container Runtime**: Runs the containers (e.g., Docker).
- **kube-proxy**: Manages network communication between pods and services.

---

# Clusters and Namespaces

## Clusters

- **Definition**: A cluster is a group of nodes that work together to run applications.

**Examples**:

1. A Kubernetes cluster running in a cloud provider's data center.
2. A local development cluster using Minikube.
3. A multi-region cluster for a globally distributed application.

## Namespaces

- **Definition**: Namespaces are used to divide a cluster into virtual sub-clusters.

**Examples**:

1. Creating separate namespaces for development, testing, and production environments.
2. Using namespaces to isolate resources for different teams within an organization.

3. Managing resource quotas and access controls at the namespace level.

---

## Setting Up a Local Kubernetes Cluster

**Tools**:

- **Minikube**: Runs a single-node Kubernetes cluster locally.
- **Docker Desktop**: Provides a local Kubernetes cluster as part of Docker installation.

**Examples**:

1. Using Minikube to test Kubernetes configurations on a local machine.
2. Enabling Kubernetes in Docker Desktop for local development.
3. Running a small-scale Kubernetes cluster on a laptop for learning purposes.

---

## Basic kubectl Commands

**Examples**:

1. `kubectl get nodes`: Lists all nodes in the cluster.
2. `kubectl get pods`: Lists all pods running in the cluster.
3. `kubectl create -f <file.yaml>`: Creates resources from a YAML file.

---

## Conclusion

Kubernetes is a powerful tool for managing containers in the cloud, automating tasks like deployment, scaling, and healing. Understanding its architecture and components enables you to build and manage containerized applications effectively.

---