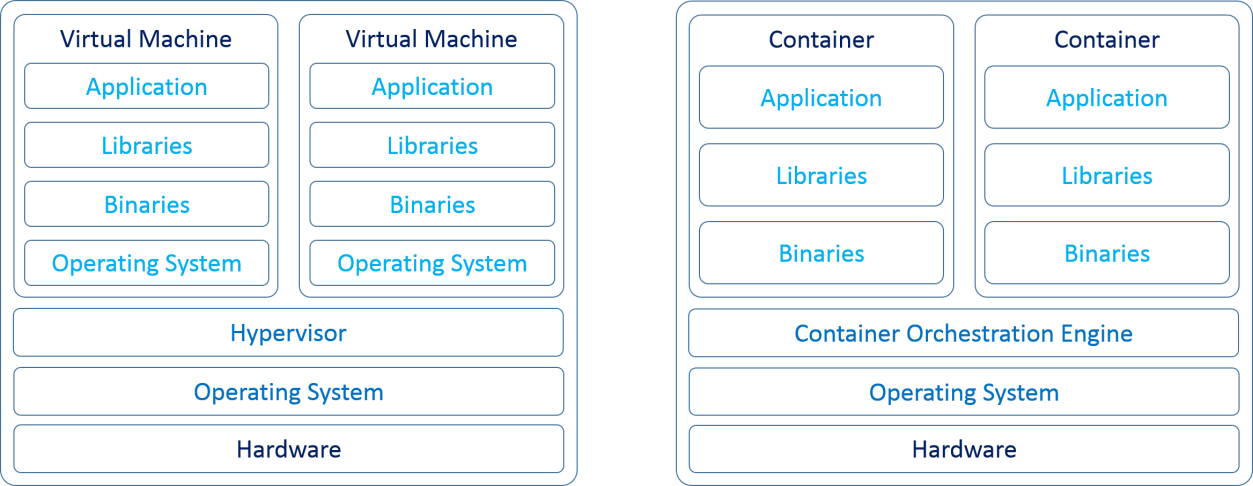
**Module No. 02**

**Write Up Name:** Container Orchestration using Kubernetes

# Theory:

# What is difference in traditional, virtualized and container deployment? Ans:

[Virtual machines](https://www.alibabacloud.com/knowledge/what-is-virtualization) are multiple guest operating systems based on hardware, which are emulated by a [hypervisor.](https://www.alibabacloud.com/knowledge/what-is-hypervisor) [Containers](https://www.alibabacloud.com/knowledge/what-is-containerization) are an application level construct, and emulate multiple virtual environments which share a single kernel.



* System Performance
* Virtual Machine :

In the case of traditional virtualization with virtual machines, each virtual machine has its own full operating system, so when running applications built into virtual machines, memory usage can be higher than necessary and virtual machines can begin to use up resources needed by the host.

* Container :

Different from traditional, containerized applications share a operating system environment (kernel) so they use fewer resources than full virtual machines and reduce pressure on the host's memory.

* Container “Weight”
* Virtual Machine :

Traditional virtual machines can take up a lot of disk space: they contain a full operating system and associated tools, in addition to whatever application the VM is hosting.

* Container :

Containers are relatively light: they contain only those libraries and tools needed to make the containerized application run, so they are more compact than virtual machines and start more quickly as well.

* Maintenance & Updates
* Virtual Machine:

When it comes to updating or patching the operating system, traditional machines must be updated one-by-one: each guest OS must be patched separately.

* Container :

With containers, only the operating system of the container host (the machine hosting the containers) must be updated. This simplifies maintenance significantly.

# What is container orchestration? Ans :

* Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a wide range of things software teams need to manage a container’s lifecycle, including provisioning, deployment, scaling (up and down), networking, load balancing and more.
* [Containers](https://www.ibm.com/cloud/learn/containers) are lightweight, executable application components that combine application source code with all the operating system (OS) libraries and dependencies required to run the code in any environment.
* [Containers](https://www.vmware.com/topics/glossary/content/containers) are a method of building, packaging and deploying software. They are similar to but not the same thing as [virtual machines](https://www.vmware.com/topics/glossary/content/virtual-machine) (VMs). One of the primary differences is that containers are isolated or abstracted away from the underlying operating system and infrastructure that they run on. In the simplest terms, a container includes both an application’s code and everything that code needs to run properly.
* Because of this, containers offer many benefits, including:
  + Portability.
  + Application development.
  + Resource utilization and optimization.

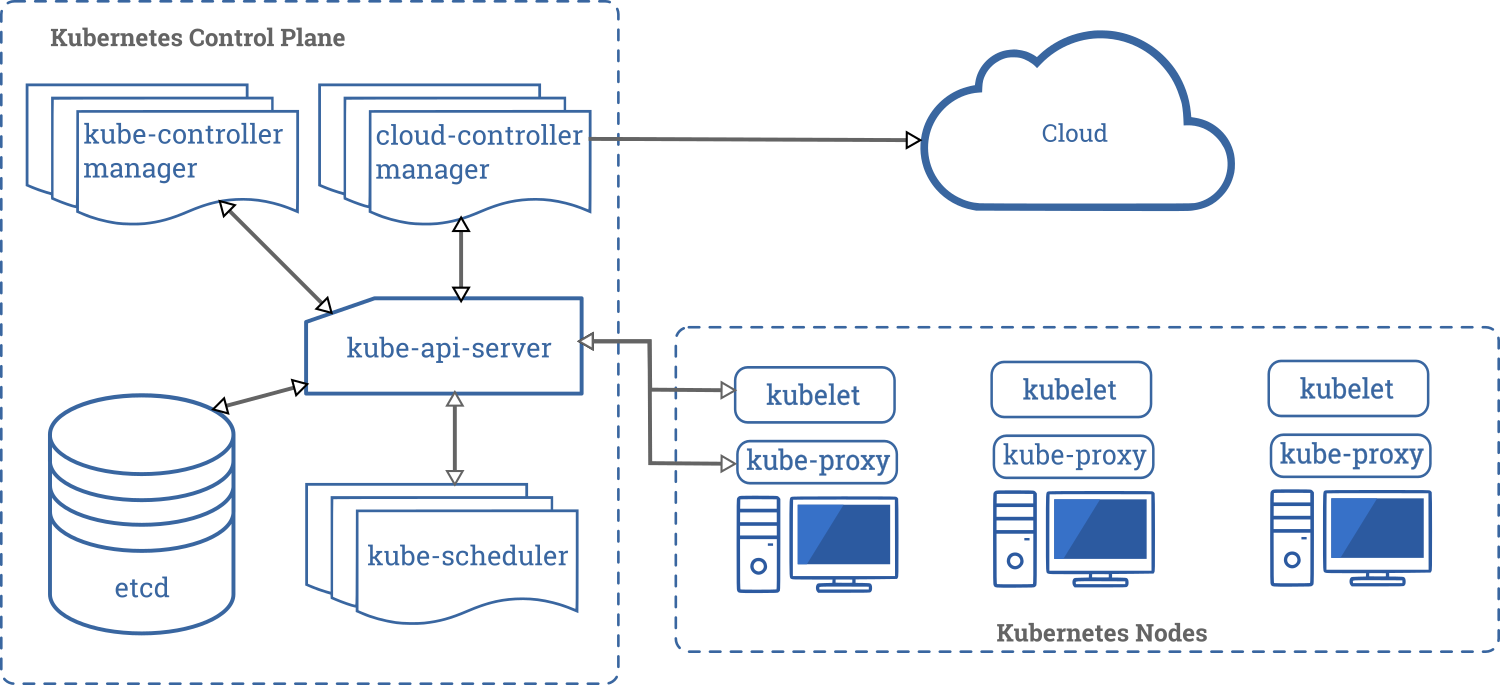
# Identify the platforms through which you could deploy containers as well orchestration tools required?

**Ans :**

* Container orchestration solves the problem by automating the scheduling, deployment, scalability, load balancing, availability, and networking of containers. Container orchestration is the automation and management of the lifecycle of containers and services.
* It is a process of managing and organizing multiple containers and microservices architecture at scale.
* There are many container orchestration tools available in the market.

# Kubernetes

[Kubernetes](https://kubernetes.io/) is an open-source platform that was originally designed by Google and now maintained by the Cloud Native Computing Foundation. Kubernetes supports both declarative configuration and automation. It can help to automate deployment, scaling, and management of containerized workload and services.



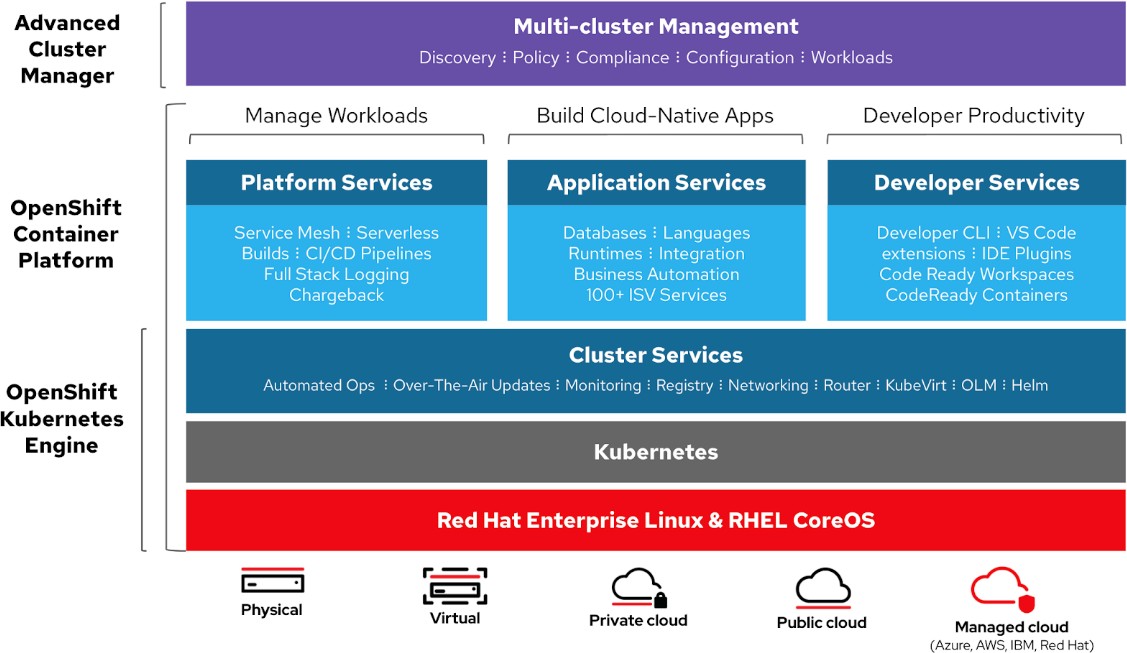
Kubernetes API helps to establish communication between users, cluster components, and external third-party components. Kubernetes control plane, and Nodes run on a group of nodes that together form the cluster. Application workload consists of one or more Pods which runs on Worker node(s). The control plane manages Pods and worker nodes.

Companies like Babylon, Booking.com, AppDirect extensively use [Kubernetes](https://geekflare.com/kubernetes-introduction/). Features

* + Service discovery and load balancing
  + Storage orchestration
  + Automated rollouts and rollbacks
  + Horizontal scaling
  + Secret and configuration management
  + Self-healing
  + Batch execution
  + IPv4/IPv6 dual-stack
  + Automatic bin packing

# OpenShift

Redhat offers [OpenShift](https://www.openshift.com/) Container Platform as a Service (PaaS). It helps in the automation of applications on secure and scalable resources in hybrid cloud environments. It provides enterprise-grade platforms for building, deployment, and managing containerized applications.



It’s built on Redhat enterprise Linux and Kubernetes engine. Openshift has various functionalities to manage clusters via UI and CLI. Redhat provides Openshift in two more variants,

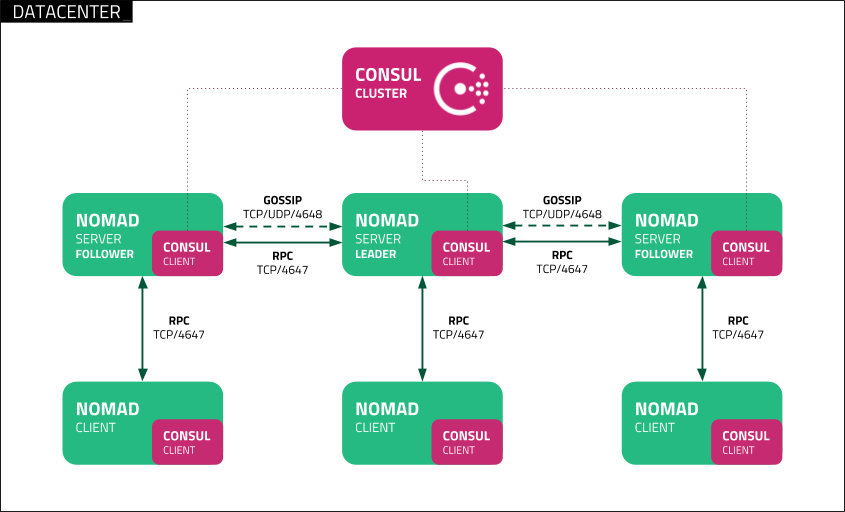
* + Openshift Online – offered as software as a service(SaaS)
  + OpenShift Dedicated – offered as managed services

Openshift Origin (Origin Community Distribution) is an open-source upstream community project which is used in OpenShift Container Platform, Openshift Online, and OpenShift Dedicated.

# Nomad

[Nomad](https://www.nomadproject.io/) is a simple, flexible, and easy to use workload orchestrator to deploy and manage containers and non-containerized applications across on-prem and clouds at scale. Nomad runs as a single binary with a small resource footprint (35MB) and supported on macOS, Windows, Linux.

Developers use declarative infrastructure-as-code ([IaC](https://geekflare.com/infrastructure-as-code-intro/)) for deploying their applications and define how an application should be deployed. Nomad automatically recovers applications from failures.



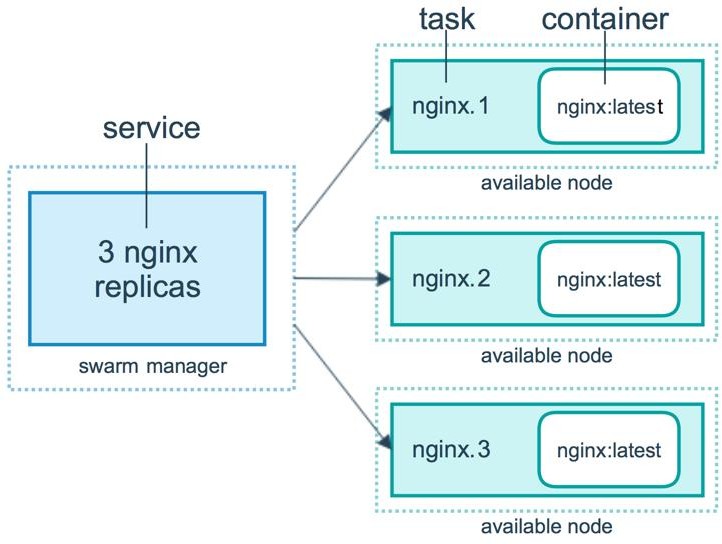
Nomad Orchestrate applications of any type (not just containers). It provides First-class support for Docker, Windows, Java, VMs, and more.

Features

* + Simple & Reliable
  + Modernize Legacy Applications without Rewrite
  + Easy Federation at Scale
  + Proven Scalability
  + Multi-Cloud with Ease
  + Native Integrations with Terraform, Consul, and Vault

# Docker Swarm

[Docker Swarm](https://docs.docker.com/engine/swarm/) uses a declarative model. You can define the desired state of the service, and Docker will maintain that state. Docker Enterprise Edition has integrated Kubernetes with Swarm. Docker is now providing flexibility in the choice of orchestration engine. Docker engine CLI is used to create a swarm of docker engines where application services can be deployed.



Features

* + Cluster management integrated with Docker Engine
  + Decentralized design
  + Declarative service model
  + Scaling
  + Desired state reconciliation
  + Multi-host networking
  + Service discovery
  + Load balancing
  + Secure by default
  + Rolling updates

# Docker Compose

[Docker Compose](https://docs.docker.com/compose/) is for defining and running multi-container applications that work together. Docker-compose describes groups of interconnected services that share software dependencies and are orchestrated and scaled together.

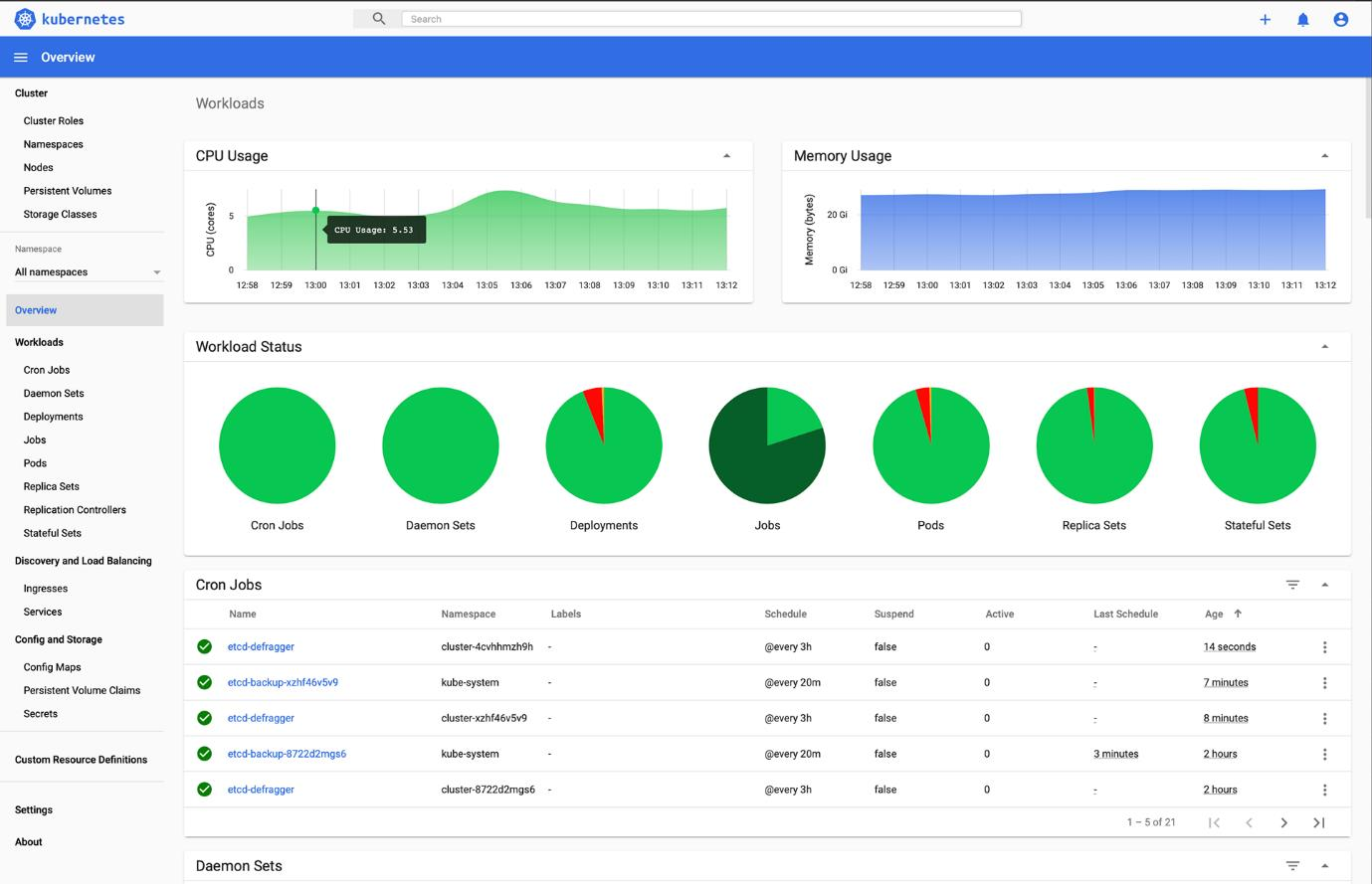
You can use a YAML file (*[dockerfile](https://geekflare.com/dockerfile-tutorial/)*) to configure your application’s services. Then, with a docker-compose up command, you create and start all the services from your configuration.

Features

* + Multiple isolated environments on a single host
  + Preserve volume data when containers are created
  + Only recreate containers that have changed
  + Variables and moving a composition between environments

# MiniKube

[Minikube](https://minikube.sigs.k8s.io/docs/) allows users to run Kubernetes locally. With Minikube, you can test applications locally inside a single-node Kubernetes cluster on your personal computer. Minikube has integrated support for the [Kubernetes Dashboard](https://geekflare.com/kubernetes-dashboard/).



Features

* + Load Balancing
  + Multi-cluster
  + Persistent Volumes
  + NodePorts
  + ConfigMaps and Secrets
  + Container Runtime: Docker, CRI-O, and containered
  + Enabling CNI (Container Network Interface)

# Are docker and Kubernetes dependent on each other!! Compare and contrast the ideas.

**Ans:**

* Kubernetes can run without Docker and Docker can function without Kubernetes. But Kubernetes can benefit greatly from Docker and vice-versa.
* Docker is a standalone software that can be installed on any computer to run containerized applications. Containerization is an approach of running applications on an Os such that application is isolated from the rest of the system. You create an illusion for our application that it is getting its very own os instance, although there may be other containers running

on same system. Docker is what enables us to run, create and manage containers on a single operating system.

* If you have Docker installed on a bunch of hosts, you can leverage Kubernetes. These nodes, or Docker hosts, can be bare-metal servers or virtual machines. Kubernetes can then allow you to automate container provisioning, networking, load-balancing, security and scaling across all these nodes from a single command line or dashboard. A collection of nodes that is managed by a single Kubernetes instance is referred to Kubernetes cluster.
* A fundamental difference between Kubernetes and docker is that Kubernetes is meant to run across a cluster while docker runs on single node. Kubernetes is more extensive than docker swarm and is meant to coordinate cluster of nodes at scale in production in an efficient manner. Kubernetes pods scheduling units that can contain one or more container in Kubernetes ecosystem are distributed among nodes to provide high availability.

# What if containers are damaged or goes down for whatever reasons. Will I be able to retrieve the data back!! How do I ensure that I can retrieve my data? State our views.

**Ans:**

* One problem is the loss of files when a container crashes and solution to this is to use a volume.
* Docker has a concept of volumes, though it is somewhat looser and less managed. A Docker volume is a directory on disk or in another container. Docker provides volume drivers, but the functionality is somewhat limited.
* Kubernetes supports many types of volumes. A Pod can use any number of volume types simultaneously. Ephemeral volume types have a lifetime of a pod, but persistent volumes exist beyond the lifetime of a pod. When a pod ceases to exist, Kubernetes destroys ephemeral volumes; however, Kubernetes does not destroy persistent volumes. For any kind of volume in a given pod, data is preserved across container restarts.
* At its core, a volume is a directory, possibly with some data in it, which is accessible to the containers in a pod.

# Explain what is the importance of ingress in Kuberentes! Ans :

* Kubernetes Ingress is an API object that provides routing rules to manage external users' access to the services in a Kubernetes cluster, typically via HTTPS/HTTP. With Ingress, you can easily set up rules for routing traffic without creating a bunch of Load Balancers or exposing each service on the node. This makes it the best option to use in production environments.
* In production environments, you typically need content-based routing, support for multiple protocols, and authentication. Ingress allows you to configure and manage these capabilities inside the cluster.
* Ingress is made up of an Ingress API object and the Ingress Controller. As we have discussed, Kubernetes Ingress is an API object that describes the desired state for exposing services to the outside of the Kubernetes cluster. An Ingress Controller is essential because it is the actual implementation of the Ingress API. An Ingress Controller reads and

processes the Ingress Resource information and usually runs as pods within the Kubernetes cluster.

* An Ingress provides the following:
  + Externally reachable URLs for applications deployed in Kubernetes clusters
  + Name-based virtual host and URI-based routing support
  + Load balancing rules and traffic, as well as SSL termination