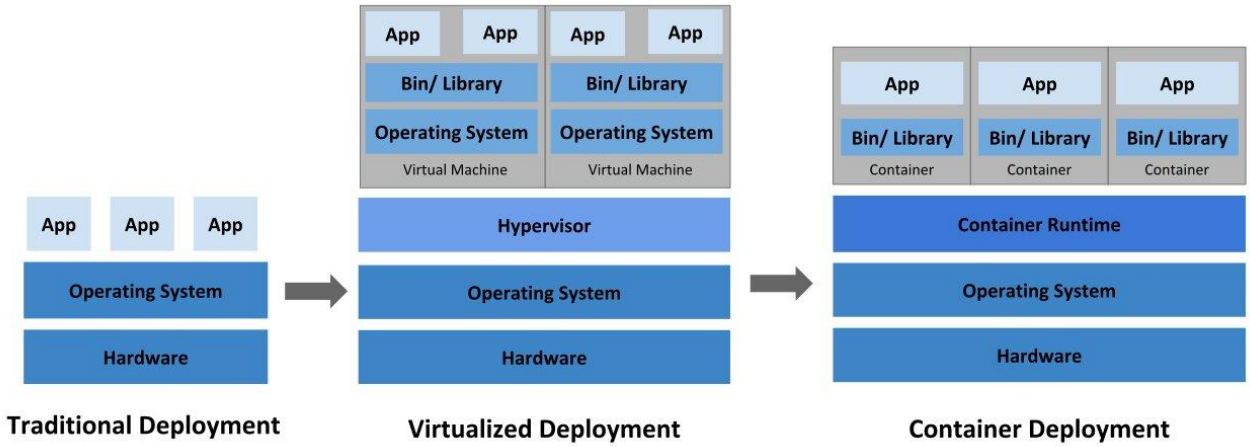
ASSIGNMENT 5

1. What exactly is Kubernetes?

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.



Container deployment era: Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, share of CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

Containers have become popular because they provide extra benefits, such as:

Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.

Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and efficient rollbacks (due to image immutability).

Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.

Observability: not only surfaces OS-level information and metrics, but also application health and other signals.

Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.

Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.

Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.

Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.

Resource isolation: predictable application performance.

Resource utilization: high efficiency and density

Why you need Kubernetes and what it can do

Service discovery and load balancing Kubernetes can expose a container using the DNS name or using their own IP address. If traffic to a container is high, Kubernetes is able to load balance and distribute the network traffic so that the deployment is stable.

Storage orchestration Kubernetes allows you to automatically mount a storage system of your choice, such as local storages, public cloud providers, and more.

Automated rollouts and rollbacks You can describe the desired state for your deployed containers using Kubernetes, and it can change the actual state to the desired state at a controlled rate. For example, you can automate Kubernetes to create new containers for your deployment, remove existing containers and adopt all their resources to the new container.

Automatic bin packing You provide Kubernetes with a cluster of nodes that it can use to run containerized tasks. You tell Kubernetes how much CPU and memory (RAM) each container needs. Kubernetes can fit containers onto your nodes to make the best use of your resources.

Self-healing Kubernetes restarts containers that fail, replaces containers, kills containers that don't respond to your user-defined health check, and doesn't advertise them to clients until they are ready to serve.

Secret and configuration management Kubernetes lets you store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys. You can deploy and update secrets and application configuration without rebuilding your container images, and without exposing secrets in your stack configuration

2. What do you mean by Kubernetes load balancing?

Kubernetes load balancing makes the most sense in the context of how Kubernetes organizes containers. Kubernetes does not view single containers or individual instances of a service, but rather sees containers in terms of the specific services or sets of services they perform or provide.

In Kubernetes, a pod is a set of containers that are related by function. A set of related pods that have the same set of functions is a service. This allows Kubernetes to create and destroy pods automatically based on need, without additional input from the user, and pods are designed not to be persistent.

IP addresses for Kubernetes pods are not persistent because the system assigns each new pod a new IP address. Typically, therefore, direct communication between pods is impossible. However, services have their own relatively stable IP addresses which field requests from external resources. The service then dispatches the request to an available Kubernetes pod.

The Kubernetes pod, a set of containers, along with their shared volumes, is a basic, functional unit. Containers are typically closely related in terms of services and functions they provide.

Services are sets of Kubernetes pods that have the same set of functions. These Kubernetes services stand in for the individual pods as users access applications, and the Kubernetes scheduler ensures that you have the optimal number of pods running at all times by creating and deleting pods as needed. In other words, Kubernetes services are themselves the crudest form of load balancing traffic.

In Kubernetes the most basic type of load balancing is **load distribution**. Kubernetes uses two methods of load distribution. Both of them are easy to implement at the dispatch level and operate through the kube-proxy feature. Kube-proxy manages virtual IPs for services.

The default kube-proxy mode for rule-based IP management is iptables, and the iptables mode native method for load distribution is random selection. Previously, kube-proxy default mode was userspace, with its native method for load distribution being round-robin.

There are several cases when you might access services using the Kubernetes proxy:

Allowing internal traffic

Connecting directly to them directly from a computer

Debugging services

Displaying internal dashboards

However, you should not use this method for production services or to expose your service to the internet. This is because the kube proxy requires you to run kubectl as an authenticated user.

In any case, for true load balancing, **Ingress offers the most popular method**. Ingress operates using a controller with an Ingress resource and a daemon. The Ingress resource is a set of rules governing traffic. The daemon applies the rules inside a specialized Kubernetes pod. The Ingress controller has its own sophisticated capabilities and built-in features for load balancing and can be customized for specific vendors or systems.

A cloud service-based Kubernetes **external load balancer** may serve as an alternative to Ingress, although the capabilities of these tools are typically provider-dependent. External network load balancers may also lack granular access at the pod level.

There are many varieties of Ingress controllers, with various features, and a range of plugins for Ingress controllers, such as cert-managers that provision SSL certificates automatically.

3. What exactly do you mean by "operator," and why do we need them?

In Kubernetes, an operator is an application-specific controller that can help you package, deploy, and manage a Kubernetes application.

Ordinarily, you run and manage Kubernetes applications via the Kubernetes application programming interface (API) and kubectl tooling. Operators lets you extend the functionality of the Kubernetes API, enabling it to configure, create, and manage instances of applications automatically using a structured process.

Operators use the basic capabilities of Kubernetes controllers and resources, but add application-specific or domain knowledge to automate the entire lifecycle of the application it manages.

- Operators Extend Kubernetes Functionality
- Improve Management of Hybrid Environments
- Making Kubernetes Automation Seamless

Kubernetes operator’s workflow:

User makes changes to a custom resource definitions(CRD)

The operator tracks the CRD and identifies change events

The operator reconciles the CRD state with the desired state

The operator adjusts cluster state to the desired state

Most common operators used by Kubernetes administrators

RBAC Manager Operator

HPA Kubernetes Operator

Istio Operator

Elastic Cloud on Kubernetes (Elastic Kubernetes Operator)

Grafana Operator

Starboard Operator

4. What is the best way to operate Kubernetes locally?

Minikube has the primary goals of being the best tool for local Kubernetes application development, and to support all Kubernetes features that fit.

kind runs local Kubernetes clusters using Docker container "nodes."

CodeReady Containers (CRC) manages a local OpenShift 4.x cluster optimized for testing and development purposes.

Minishift helps you run OpenShift 3.x clusters locally by running a single-node OpenShift cluster inside a virtual machine (VM).

5. How can the Kubernetes Cluster be monitored

Kubernetes Monitoring Tools

The following are some of the more popular tools used to monitor Kubernetes clusters

- Prometheus
- Grafana
- ELK Elastic stack
- Fluentd
- cAdvisor

Best Practices To Monitor Your Cluster

- Use DaemonSets
- Tags And Labels
- Use Service Discovery

- Kube-System
- Constantly Watch For High Disk Usage

6. What exactly do you mean when you say GKE?

Google Kubernetes Engine (GKE) is a management and orchestration system for Docker container and container clusters that run within Google's public cloud services. Google Kubernetes Engine is based on Kubernetes, Google's open source container management system.

Organizations typically use Google Kubernetes Engine to:

- Create or resize Docker container clusters
- Create container pods, replication controllers, jobs, services or load balancers
- Resize application controllers
- Update and upgrade container clusters
- Debug container clusters

7. What is the difference between Kubernetes and Docker Swarm?

Docker Swarm

Docker Swarm is an open-source container orchestration platform built and maintained by Docker. Under the hood, Docker Swarm converts multiple Docker instances into a single virtual host. A Docker Swarm cluster generally contains three items:

- Nodes
- Services and tasks
- Load balancers

Nodes are individual instances of the Docker engine that control your cluster and manage the containers used to run your services and tasks. Docker Swarm clusters also include load balancing to route requests across nodes.

Advantages of Docker Swarm

Docker Swam is straightforward to install, especially for those just jumping into the container orchestration world. It is lightweight and easy to use. Also, Docker Swarm takes less time to understand than more complex orchestration tools. It provides automated load balancing within the Docker containers, whereas other container orchestration tools require manual efforts.

Docker Swarm works with the Docker CLI, so there is no need to run or install the entire new CLI. Plus, it works seamlessly with existing Docker tools such as Docker Compose.

Docker Swarm does not require configuration changes if your system is already running inside Docker.

Disadvantages of Docker Swarm

Despite its benefits, there are a few disadvantages of using Docker Swarm that you should be aware of.

First, it is lightweight and tied to the Docker API, which limits functionality in Docker Swarm, compared to Kubernetes. Likewise, Docker Swarm's automation capabilities are not as robust as those offered by Kubernetes.

Kubernetes

Kubernetes is an open source container orchestration platform that was initially designed by Google to manage their containers. Kubernetes has a more complex cluster structure than Docker Swarm. It usually has a builder and worker nodes architecture divided further into pods, namespaces, config maps, and so on.

Advantages of Kubernetes

Kubernetes offers a wide range of benefits to teams looking for a robust container orchestration tool:

It has a large open-source community, and Google backs it.

It supports every operating system.

It can sustain and manage large architectures and complex workloads.

It is automated and has a self-healing capacity that supports automatic scaling.

It has built-in monitoring and a wide range of available integrations.

It is offered by all three key cloud providers: Google, Azure, and AWS.

Because of its broad community support and ability to handle even the most complex deployment scenarios, Kubernetes is often the number one choice for enterprise development teams managing microservice-based applications.

Disadvantages of Kubernetes

Despite its comprehensive feature set, Kubernetes also has a few drawbacks:

It has a complex installation process and a steep learning curve.

It requires you to install separate CLI tools and learn each of them.

The transition from Docker Swarm to Kubernetes can be complicated and difficult to manage.

In some situations, Kubernetes can be overly complicated and lead to a loss of productivity.