

K8S Essential



THAO LUONG
03/2022



Content

- Introduction
- Container Overview
 - What is container?
 - Container vs virtual machine
 - Container use cases
 - Advantages & disadvantages of container
 - Docker
- Kubernetes Overview
 - What is container orchestrator?
 - What is kubernetes?
 - Microservices
- K8s core concepts
 - Workload Object
 - Network Object
 - Storage Object
 - Configuration Object
 - Authentication & identity Object



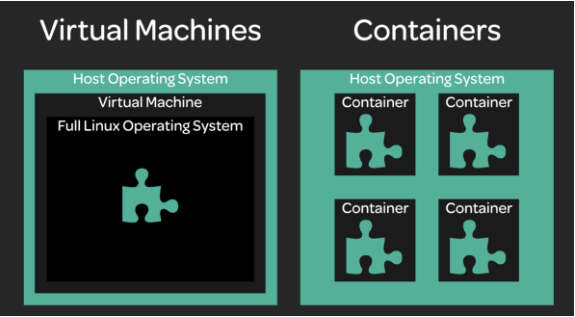
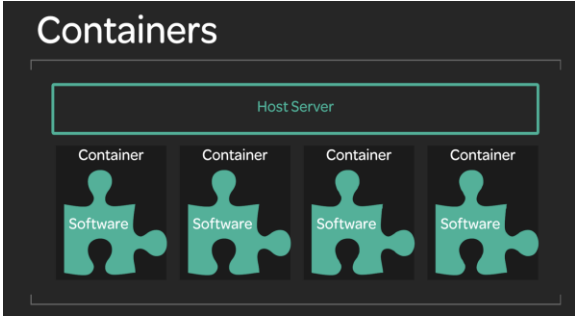
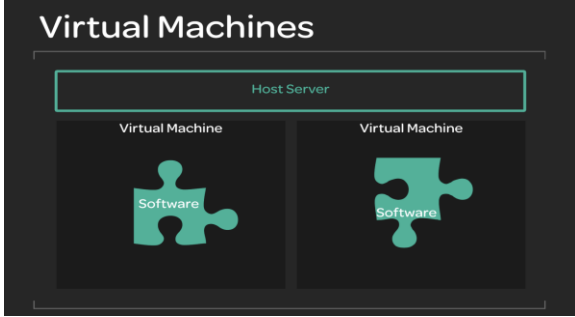
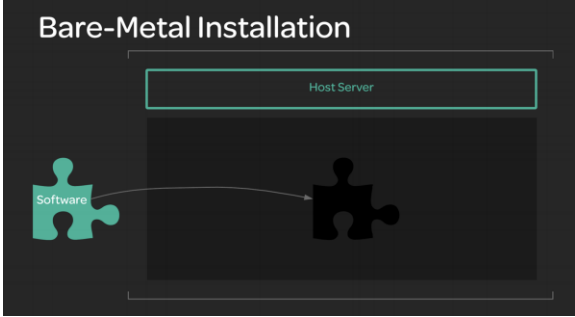
Container Overview

What are Containers?

- Containers are all about portable software.
- They are a technology that allows you to run software on a variety of systems, including a developer's laptop, all the way to a production system.
- This speeds up deployment, simplifies automation, and ensures your code can run consistently in production, as well as everywhere else!
- Like virtual machines, containers wrap your software in a standardized environment that allows it to run consistently on varied machines.
- But containers are smaller, use fewer resources, and are easier to automate than virtual machines.



Container vs Virtual Machine vs Bare-Metal





Container Use cases

What are Containers Used For?


- Software Portability – Running software consistently on different machines.
- Isolation – Keeping individual pieces of software separate from one another.
- Scaling – Increasing or decreasing resources allocated to software as needed.
- Automation – Automating processes to save time and money.
- Efficient Resource Usage – Containers use resources efficiently, which saves money.



Pros

&

Cons

- 
- + The isolation and portability of VMs.
 - + More lightweight than VMs - Less resource usage.
 - + Faster than VMs – Containers can start up in seconds, not minutes.
 - + Smaller than VMs – Container images can be measured in megabytes, not gigabytes.
 - + All of these add up to faster and simpler automation!

- Less flexibility than VMs – You can't run a Windows container on a Linux machine (yet).
- Introduces new challenges around orchestration and automation.



Docker

What is Docker

- Docker is primarily a container runtime.
- This means that Docker is a piece of software that is designed to implement and support containers.
- It is currently the industry leader in container runtimes.





Other container runtimes?

Docker is not the only option for doing containers!

- rkt - Created by CoreOS, “designed with composability and security in mind.”
- Containerd - Emphasizes “simplicity, robustness, and portability.”
- LXC/LXD





Kubernetes Overview

What is Orchestration?

- Container Orchestration simply refers to processes used to manage containers and to automate the management of containers.
- For example:
 - I want to start up a set of five containers in production.
 - I could spin up each container manually.
 - Or, I could tell an orchestration tool like Kubernetes that I want five containers, and let the tool do it.



Zero-Downtime Deployments

In the old days, deployments went like this:

1. Take the server down for maintenance (it is unavailable to customers).
2. Perform the deployment.
3. Bring the server back up.

A zero-downtime deployment (with containers) goes like this:

1. Spin up containers running the new code.
2. When they are fully up, direct user traffic to the new containers.
3. Remove the old containers running the old code.

No downtime for users! How do you coordinate those steps quickly and efficiently?

Orchestration tools can do it for you!



Kubernetes - k8s

- Kubernetes is a container orchestration tool.
- It allows you to easily build and manage your container infrastructure and automation.
- Do you want things like self-healing applications, automated scaling, and easy automated deployments? Then you should definitely look into Kubernetes!
- Check out the official Kubernetes site for documentation and additional info!

<https://kubernetes.io>

<https://github.com/kubernetes/kubernetes>





Other container orchestrator

- Docker Swarm
Docker's native container orchestration solution.
- Marathon
Based on Apache Mesos, offers APIs for integrating with other tools.
<https://mesosphere.github.io/marathon/>
- Nomad
Open source and built by HashiCorp, designed to be simple and lightweight.
<https://www.nomadproject.io/>
- Openshift
An enterprise-ready Kubernetes container platform with full-stack automated operations to manage hybrid cloud and multi-cloud



Cloud Orchestration Solutions

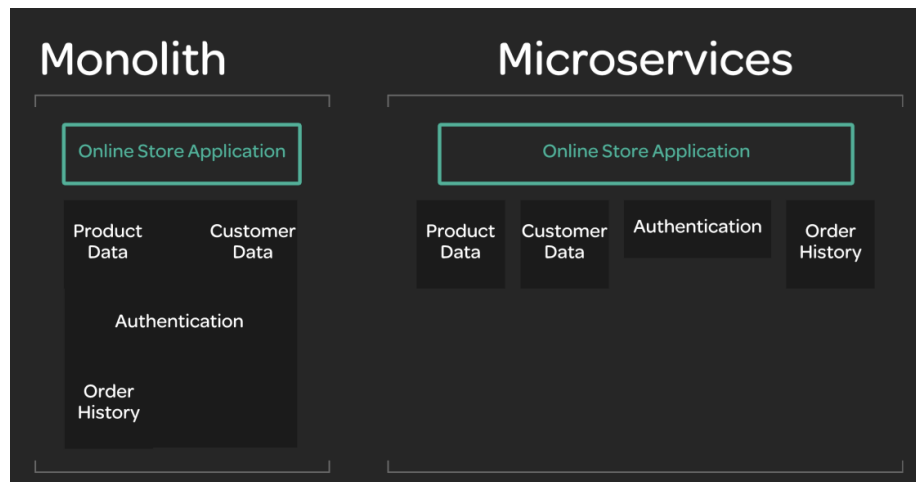
Cloud providers such as Amazon Web Services, Microsoft Azure, and Google Cloud Platform also offer built-in container orchestration solutions, including cloud-native Kubernetes implementations!

- Amazon ECS for Kubernetes - EKS
 - Azure Kubernetes Service - AKS
 - Google Kubernetes Engine - GKE
 - DigitalOcean Kubernetes - DOKS
 - IBM Cloud Kubernetes Service
- etc.



Microservices

- Microservices are a type of application architecture that involves splitting the application into a series of small, independent services.
- Microservices can be built, modified, and scaled separately, with relatively little impact on one another.





Containers and Microservices

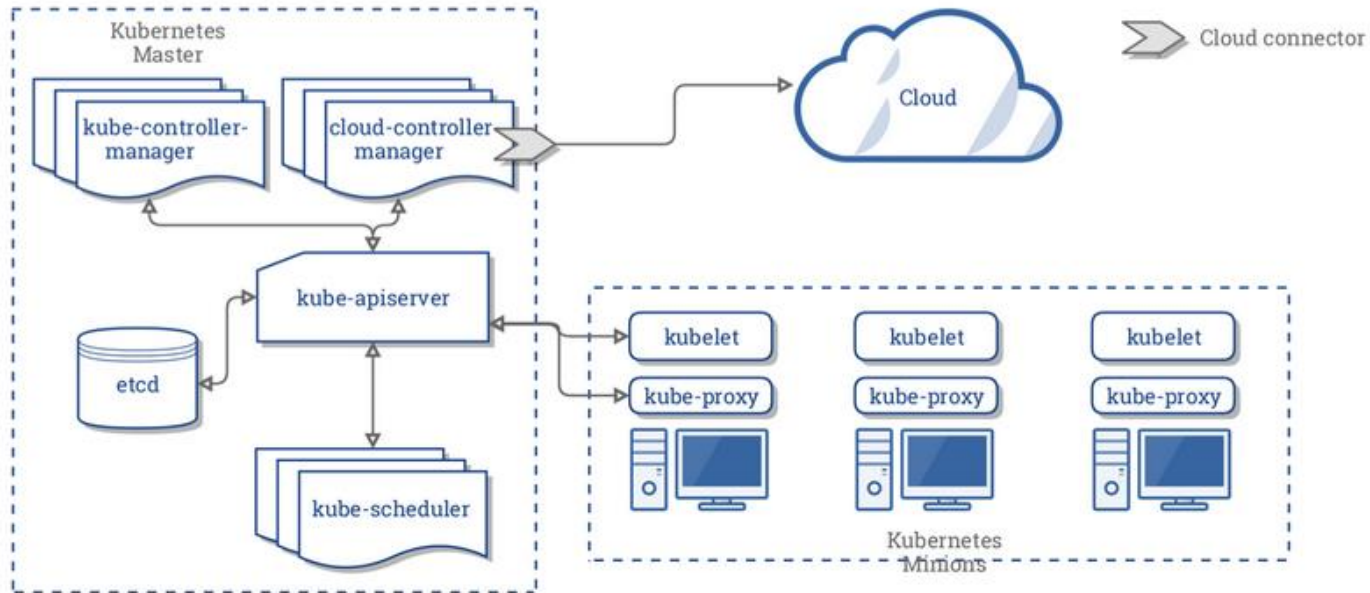
Containers excel when it comes to managing a large number of small, independent workloads.

Containers and orchestration make it easier to manage and automate the process of deploying, scaling, and connecting lots of microservice instances.

For example, I may have one microservice that needs additional resources. With containers, all I need to do is create more containers for that service to handle the load. With orchestration, that can even be done automatically and in real time!



Kubernetes Architect Overview





Master Components

The **Kubernetes Master** is a collection of the following processes that run on a single node in your cluster, which is designated as the master node.

- Kube-apiserver
- Etcd
- Kube-controller-manager
- Cloud-controller-manager
- Kube-scheduler



Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.

- Kubelet

An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod

- Kube-proxy

A network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service

- Container runtime engine



Kubernetes core concept

- Workload Object
- Network Object
- Storage Object
- Configuration Object
- Authentication & identity Object



Kubernetes workload Object

- **Pod** - A pod is the smallest unit of work or management resource within Kubernetes. It is comprised of one or more containers that share their storage, network, and context (namespace, cgroups etc).
- **Deployment** - A declarative method of managing stateless Pods and ReplicaSets. Provides rollback functionality in addition to more granular update control mechanisms.
- **StatefulSet** - A controller tailored to managing Pods that must persist or maintain state. Pod identity including hostname, network, and storage will be persisted.
- **DaemonSet** - Ensures that all nodes matching certain criteria will run an instance of a supplied Pod. Ideal for cluster wide services such as log forwarding, or health monitoring.



Kubernetes workload Object

- **Job** - The job controller ensures one or more pods are executed and successfully terminates. It will do this until it satisfies the completion and/or parallelism condition.
- **CronJob** - An extension of the Job Controller, it provides a method of executing jobs on a cron-like schedule.



Kubernetes Network Object

- **Service** - Services provide a method of exposing and consuming L4 Pod network accessible resources. They use label selectors to map groups of pods and ports to a cluster-unique virtual IP.
- **Ingress** - An ingress controller is the primary method of exposing a cluster service (usually http) to the outside world. These are load balancers or routers that usually offer SSL termination, name-based virtual hosting etc.



Kubernetes Storage Object

- **Volume** - Storage that is tied to the Pod Lifecycle, consumable by one or more containers within the pod.
- **PersistentVolume** - A PersistentVolume (PV) represents a storage resource. PVs are commonly linked to a backing storage resource, NFS, GCEPersistentDisk, RBD etc. and are provisioned ahead of time. Their lifecycle is handled independently from a pod.
- **PersistentVolumeClaim** - A PersistentVolumeClaim (PVC) is a request for storage that satisfies a set of requirements instead of mapping to a storage resource directly.
- **StorageClass** - Storage classes are an abstraction on top of an external storage resource. These will include a provisioner, provisioner configuration parameters as well as a PV reclaimPolicy.



Kubernetes Configuration Object

- **ConfigMap** - Externalized data stored within kubernetes that can be referenced as a command line argument, environment variable, or injected as a file into a volume mount. Ideal for separating containerized application from configuration.
- **Secret** - Functionally identical to ConfigMaps, but stored encoded as base64, and encrypted at rest (if configured).



Auth and Identity (RBAC)

- **[Cluster]Role** - Roles contain rules that act as a set of permissions that apply verbs like “get”, “list”, “watch” etc over resources that are scoped to apiGroups. Roles are scoped to namespaces, and ClusterRoles are applied cluster-wide.
- **[Cluster]RoleBinding** - Grant the permissions as defined in a [Cluster]Role to one or more “subjects” which can be a user, group, or service account.
- **ServiceAccount**- ServiceAccounts provide a consumable identity for pods or external services that interact with the cluster directly and are scoped to namespaces.

