Container Orchestration

Instructor: Asst. Prof. Dr. Praphan Pavarangkoon
Office: Room no. 418-5, 4th Floor
Email: praphan@it.kmitl.ac.th
Office hours: Thursday at 9:00 – 11:00 or
as an advance appointment

Agenda

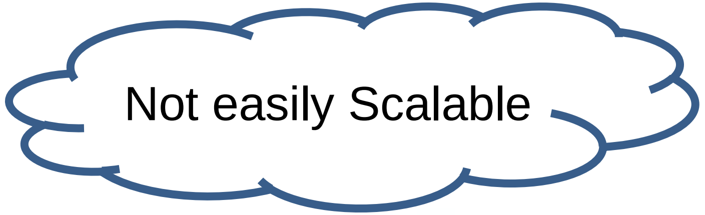
- Why do We Need Container Orchestration?
- What is Container Orchestration?
- Container Orchestration Tools
- What is Kubernetes and How It Works
- Kubernetes Use Case and Case Studies
- Kubernetes Architecture and Advantages

Containers are Good…

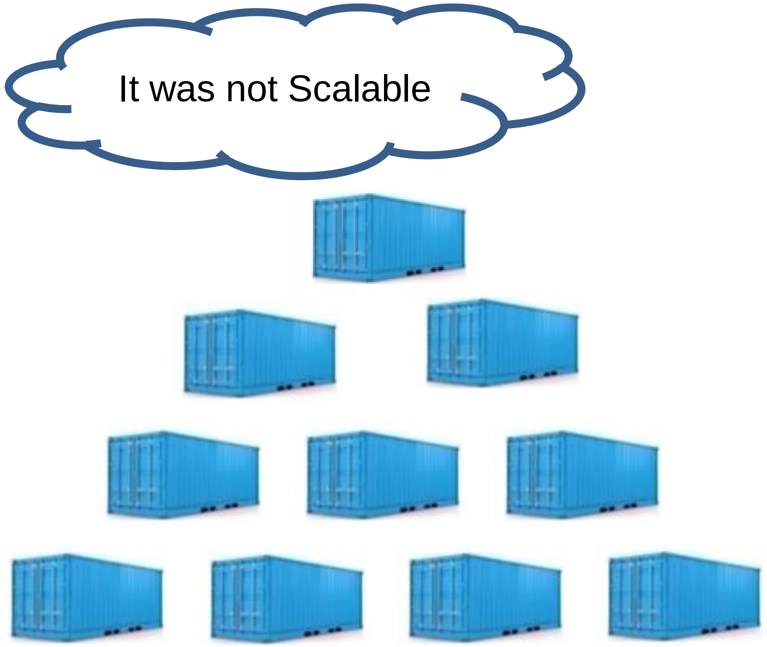
- Both Linux Containers and Docker Containers
  - ➢ Isolate the application from the host

FASTER, RELIABLE, EFFICIENT, LIGHTWEIGHT, AND SCALABLE

Containers Problems!

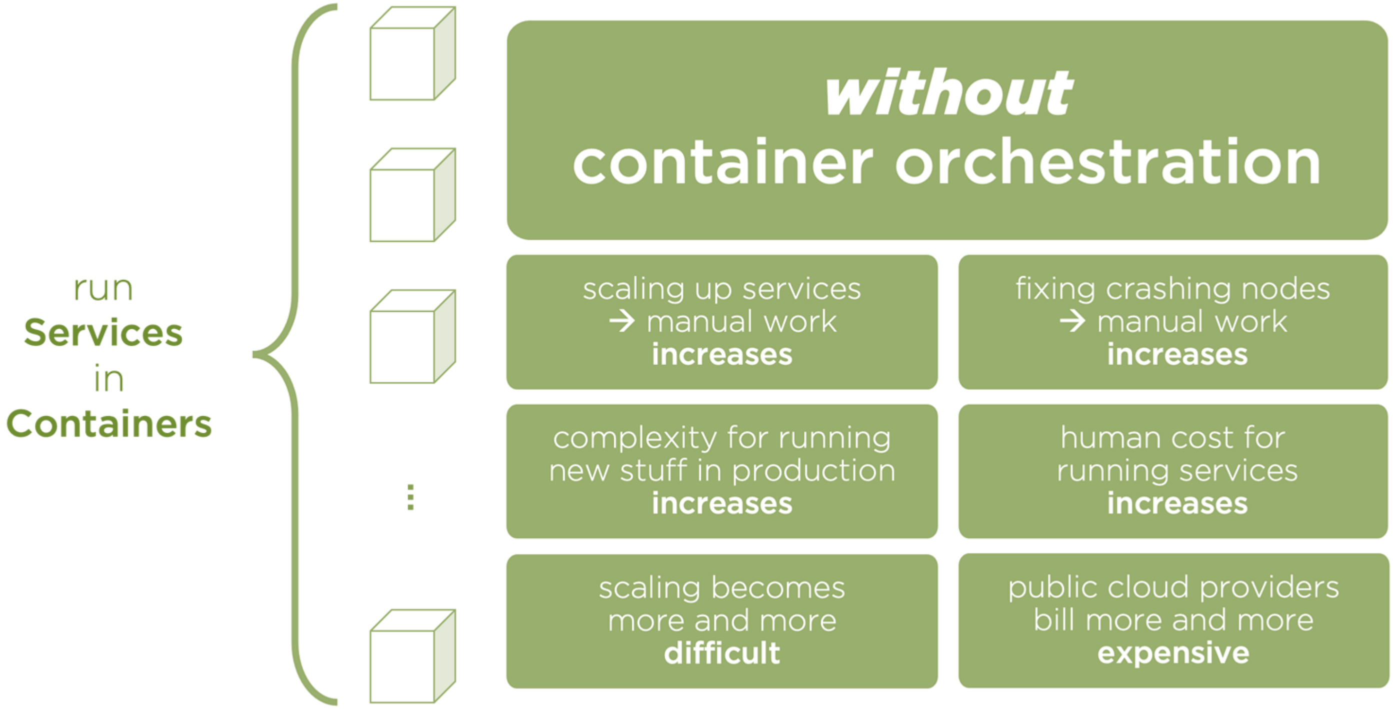


Problems with Scaling Up Containers



1. Containers could communicate with each other
2. Containers had to be deployed appropriately
3. Containers had to be managed carefully
4. Auto scaling was not possible
5. Distributing traffic was still challenging

Containers without Orchestration



run **Services** in **Containers**

without **container orchestration**

scaling up services → manual work **increases**

fixing crashing nodes → manual work **increases**

complexity for running new stuff in production **increases**

human cost for running services **increases**

scaling becomes more and more **difficult**

public cloud providers bill more and more **expensive**

Why do We Need Container Orchestration?

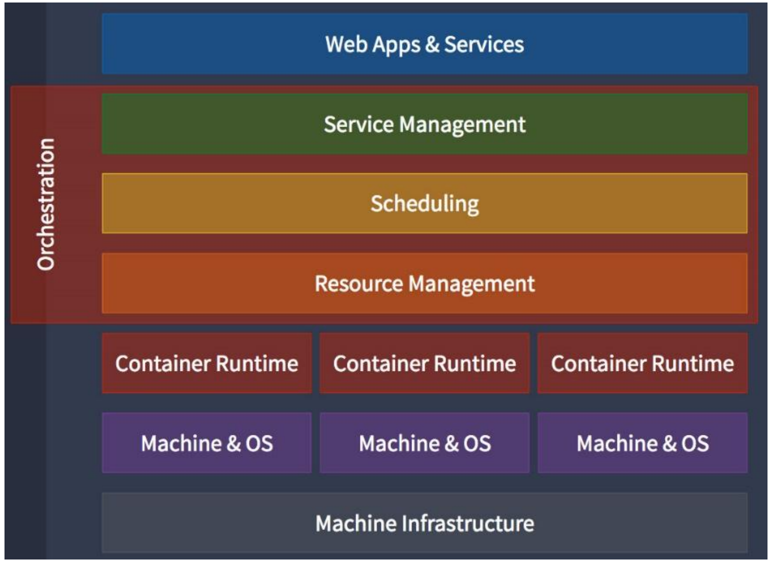
- Because containers are lightweight and ephemeral by nature, running them in production can quickly become a massive effort.
- Particularly when paired with microservices—which typically each run in their own containers—a containerized application might translate into operating hundreds or thousands of containers, especially when building and operating any large-scale system.
- This can introduce significant complexity if managed manually.
- Container orchestration is what makes that operational complexity manageable for development and operations—or DevOps—because it provides a declarative way of automating much of the work.

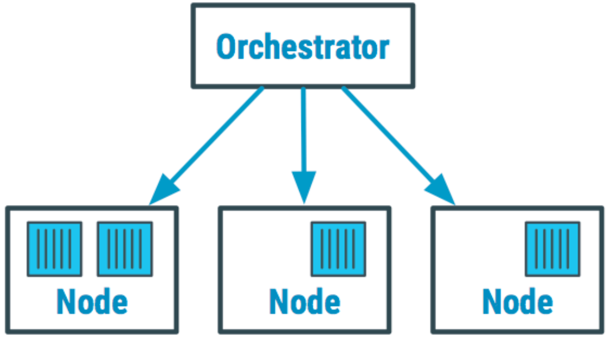
What is Container Orchestration?

- 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.

Container Orchestration

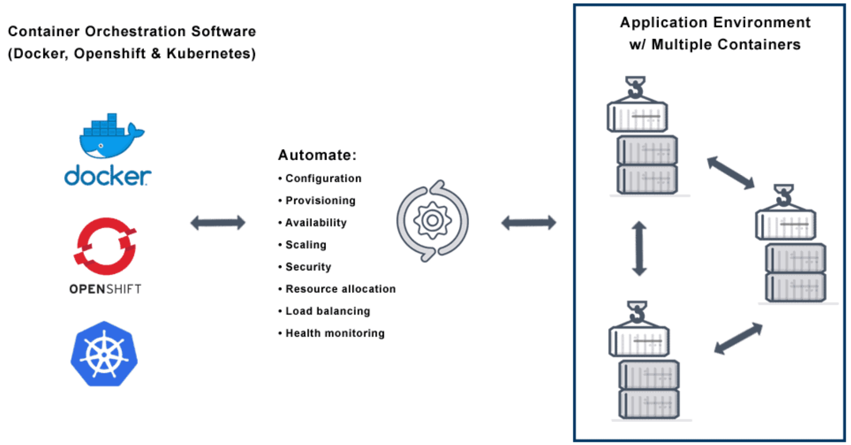
- Container orchestration automates and simplifies provisioning, and deployment and management of containerized applications.





Container Orchestration (cont.)

- Container orchestration is the automatic process of managing or scheduling the work of individual containers for applications based on microservices within multiple clusters.
- The widely deployed container orchestration platforms are based on open-source versions like Kubernetes, Docker Swarm or the commercial version from Red Hat OpenShift.



Container Orchestration (cont.)

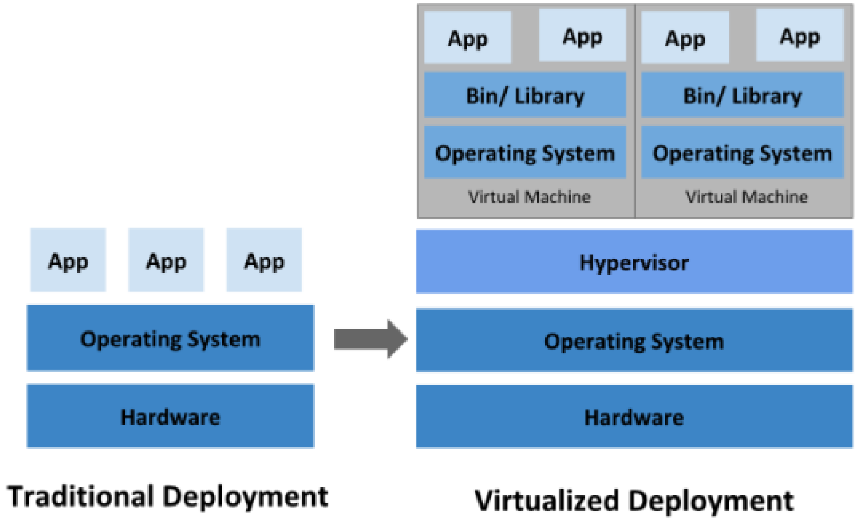
- Fault-tolerance
- On-demand scalability
- Optimal resource usage
- Auto-discovery to automatically discover and communicate with each other
- Accessibility from the outside world
- Seamless updates/rollbacks without any downtime.

Container Orchestration (cont.)

- Container orchestration is used to automate the following tasks at scale:
    - Configuring and scheduling of containers
    - Provisioning and deployments of containers
    - Availability of containers
    - The configuration of applications in terms of the containers that they run in
    - Scaling of containers to equally balance application workloads across infrastructure
    - Allocation of resources between containers
    - Load balancing, traffic routing and service discovery of containers
    - Health monitoring of containers
    - Securing the interactions between containers.

Cloud Orchestration

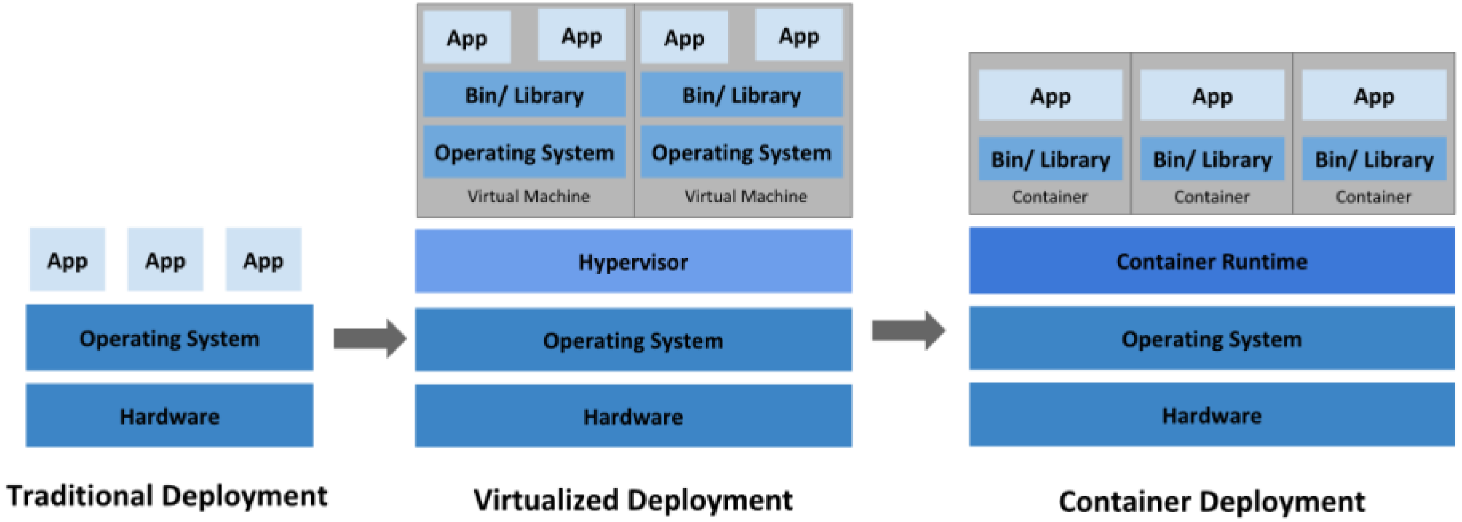
- Infrastructure-as-a-Service (IaaS)
    - ➢ Provisioning virtual machines from a cloud service provider (CSP)



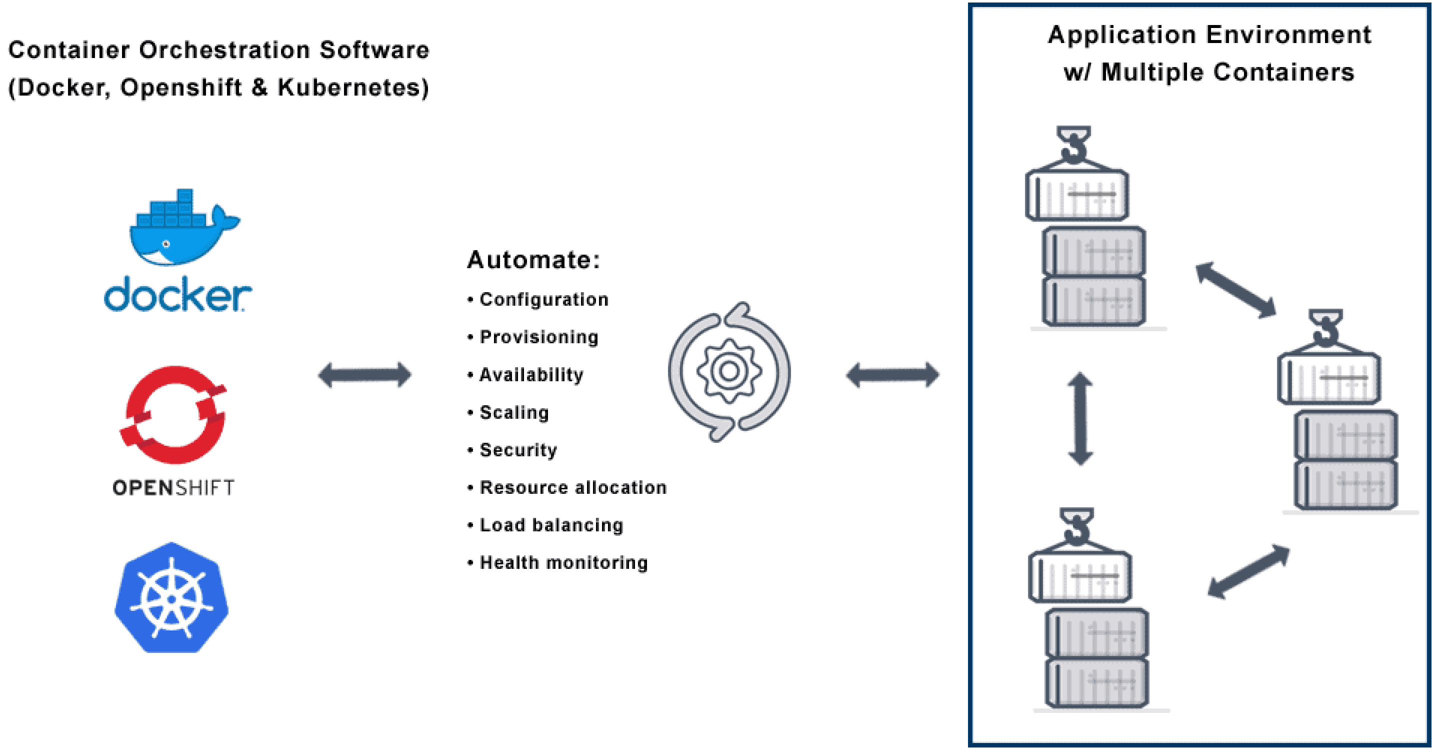
aws Azure Google Cloud openstack.

Container Orchestration (cont.)

- Application containers
  - ➢ Lightweight OS-virtualization
  - ➢ Application packaging for portable, reusable software



Container Orchestration (cont.)


Container Orchestration Software
(Docker, Openshift & Kubernetes)
docker
OPENSHIFT
Application Environment
w/ Multiple Containers

Automate:

- Configuration
- Provisioning
- Availability
- Scaling
- Security
- Resource allocation
- Load balancing
- Health monitoring

What are The Benefits of Container Orchestration?

- Container orchestration is key to working with containers, and it allows organizations to unlock their full benefits. It also offers its own benefits for a containerized environment, including:
- Simplified operations
  - ➢ Most important benefit of container orchestration and the main reason for its adoption.
  - ➢ Manages the complexity of Containers
- Resilience
  - ➢ Automatically restart or scale a container or cluster, boosting resilience.
- Added security
  - ➢ Keeping containerized applications secure by reducing or eliminating the chance of human error.

Container Orchestration Tools

kubernetes

HashiCorp Nomad

OPENSHIFT

minikube

CLOUDIFY

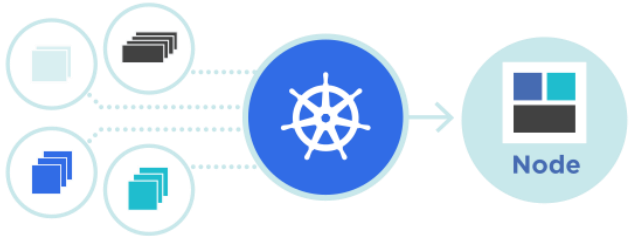
What is Kubernetes and How It Works

Kubernetes is an open-source container management tool which automates container deployment, container (de)scaling and container load balancing

Benefits (Works with all cloud vendors (Public, Private (on-premises), and Hybrid))

More about Kubernetes

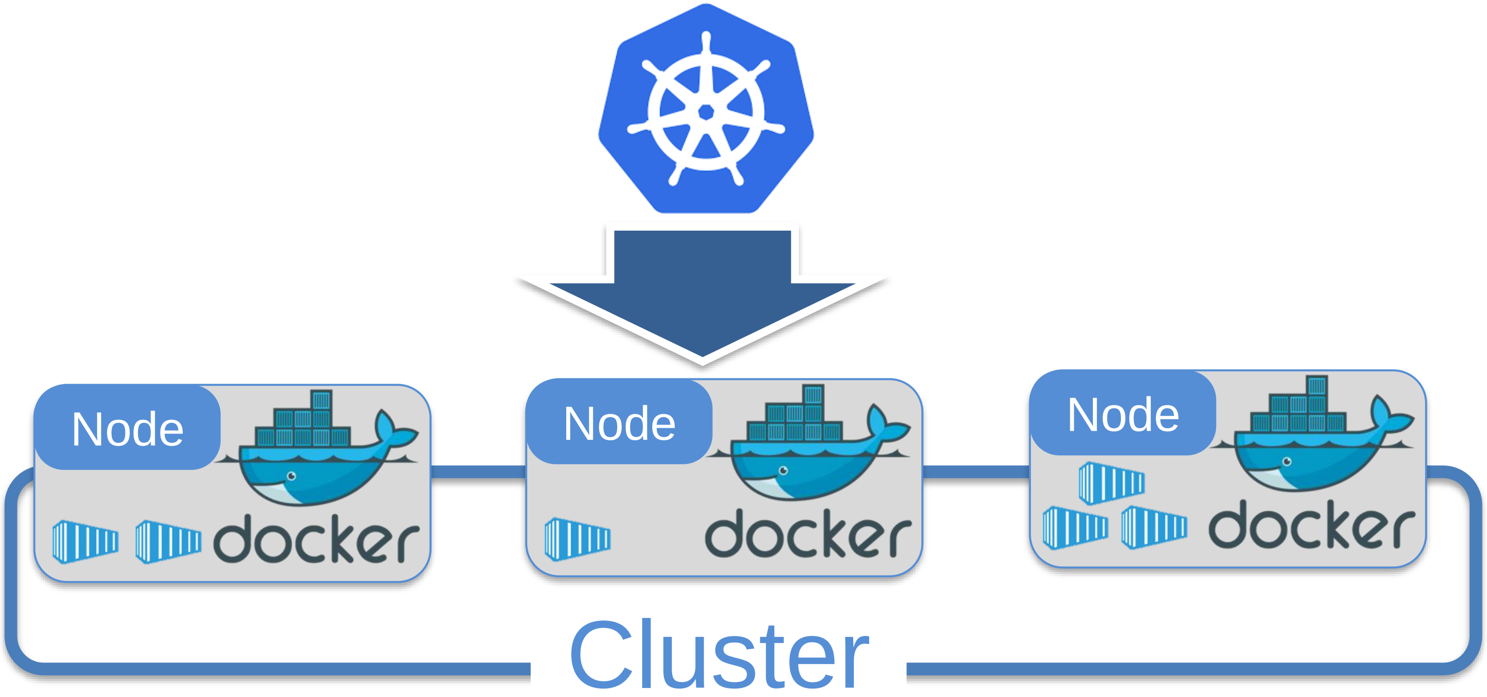
- Developed by Google and written in Golang with a huge community
- Can group ‘n’ containers into one logical unit for managing and deploying them



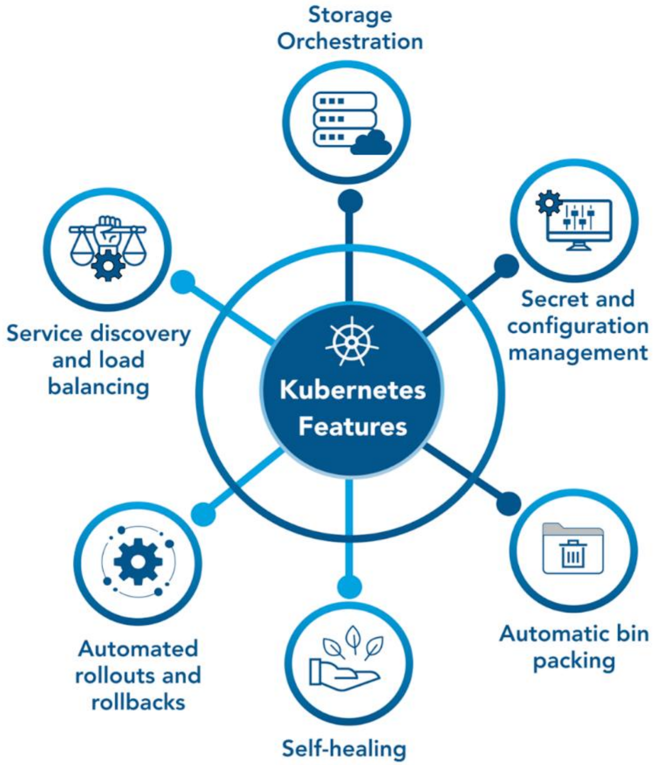
Kubernetes

- Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized application.
- Originally an open source project launched by Google and now part of the Cloud Native Computing Foundation (CNCF).
- Kubernetes is **highly extensible** and **portable**
- Kubernetes is considered **highly declarative**
- Kubernetes initial release **(7 June 2014)**
- Releases every 3 months

Kubernetes (cont.)



Kubernetes Features

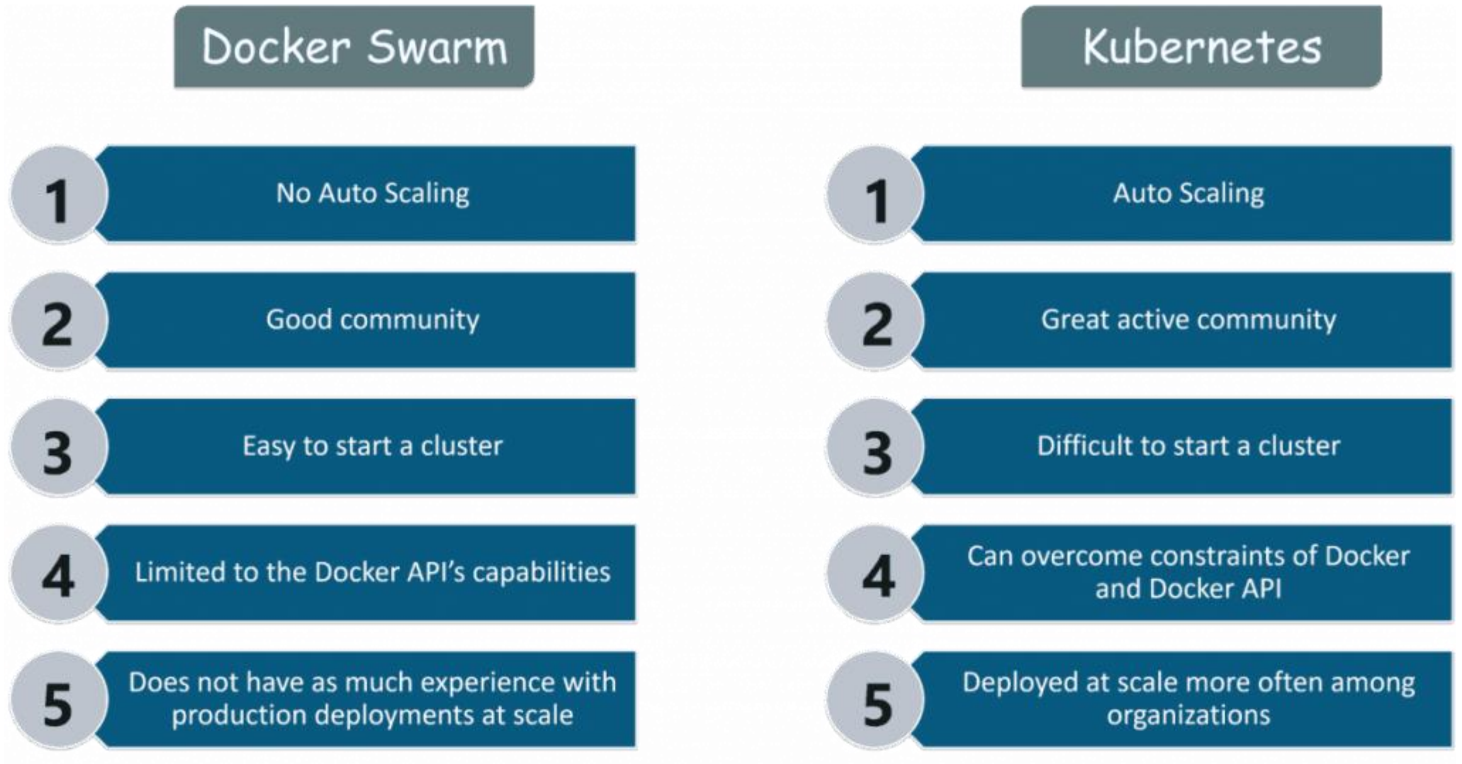

Storage Orchestration
Secret and configuration management
Service discovery and load balancing
Kubernetes Features
Automatic bin packing
Automated rollouts and rollbacks
Self-healing

Kubernetes Myths

- Kubernetes is not:
  - To be compared with Docker
  - For containerizing apps
  - For apps with simple architecture

- Kubernetes is actually:
  - Robust and reliable
  - A container orchestration platform
  - A solution for scaling up Containers
  - Backed by huge community

Kubernetes vs Docker Swarm



	Docker Swarm		Kubernetes
1	No Auto Scaling	1	Auto Scaling
2	Good community	2	Great active community
3	Easy to start a cluster	3	Difficult to start a cluster
4	Limited to the Docker API's capabilities	4	Can overcome constraints of Docker and Docker API
5	Does not have as much experience with production deployments at scale	5	Deployed at scale more often among organizations

Kubernetes vs Docker Swarm (cont.)

Features	Docker Swarm	Kubernetes
Installation and Cluster Configuration	Easy and Fast	Complicated and Time-consuming
GUI	Not Available	Available
Scalability	Scaling up is faster than K8s; but cluster strength is not as robust	Scaling up is slower but guarantees stronger cluster state
Load Balancing	Built-in load balancing technique	Requires manual service configuration
Updates and Rollback	Progressive updates and service health monitoring	Process scheduling to maintain services while updating
Data Volumes	Can be shared with any container	Only shared with containers in the same Pod
Logging and Monitoring	Only 3rd party logging and monitoring tools	Built-in logging and monitoring tools

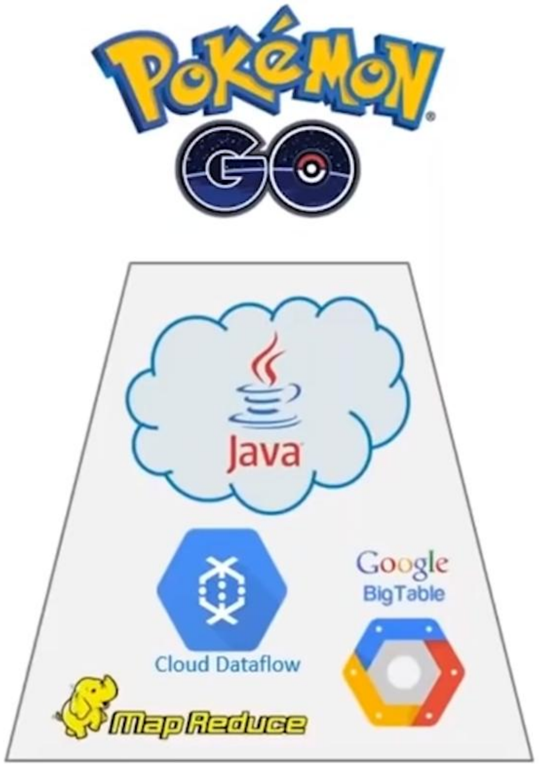
Kubernetes Use Case and Case Studies

Pokemon Go is an augmented reality game developed by **Niantic** for Android and iOS devices

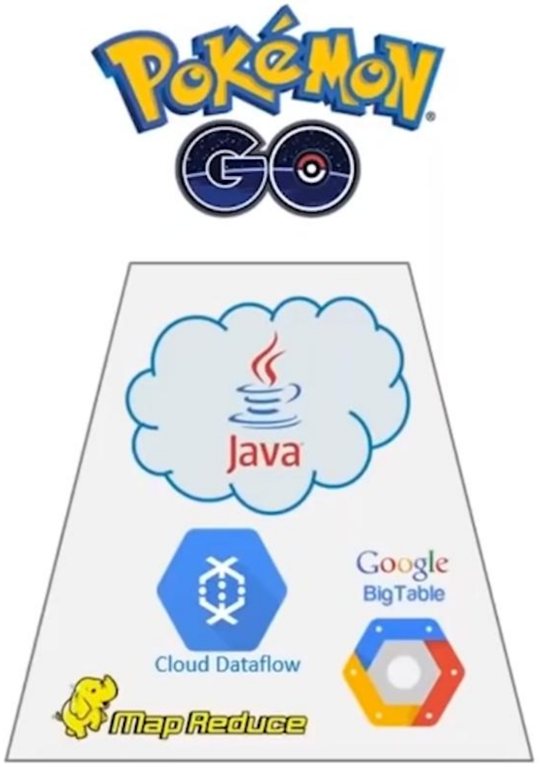
Key Stats:

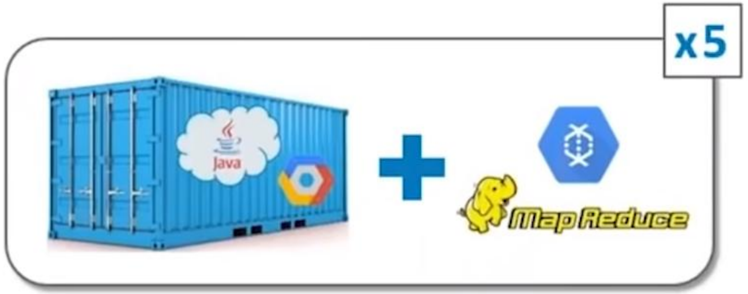
- 500+ million downloads, 20+ million daily active users
- Initially only launched in NA, Australia and New Zeeland
- Inspired users to walk over 5.4 billion miles in a year
- Surpassed engineering expectations by 50 times

Backend Architecture

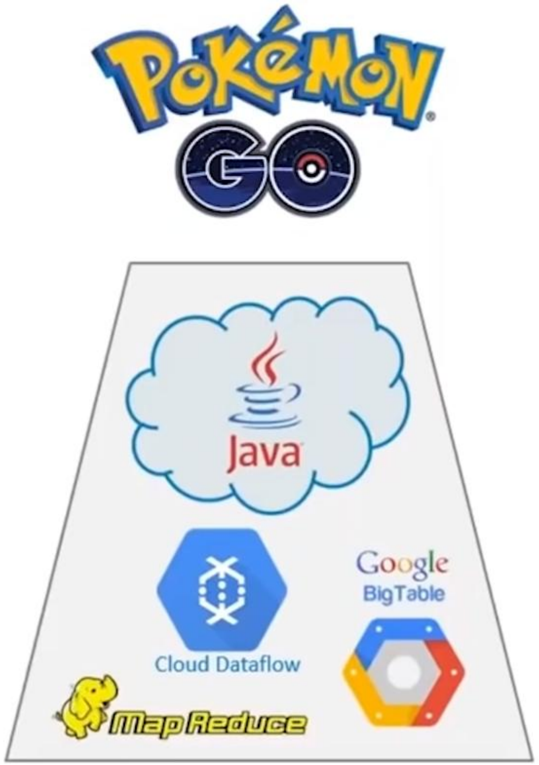


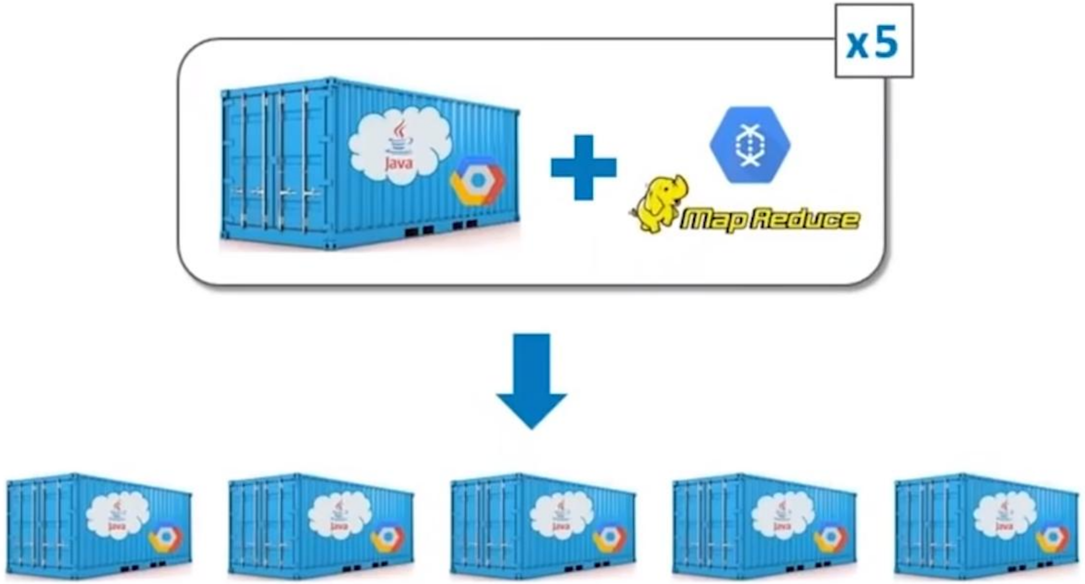
MapReduce & Cloud Dataflow for Scaling Up



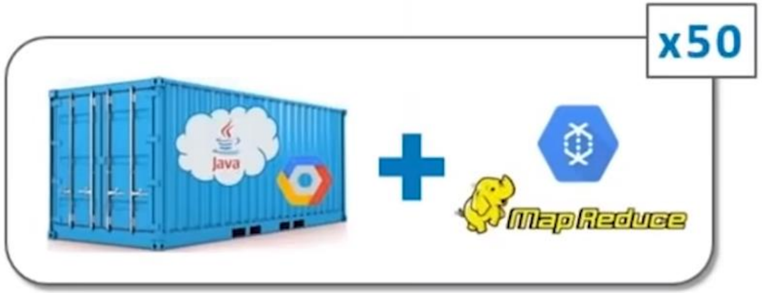


Easy Scaling of Containers Using Kubernetes





Easy Scaling of Containers Using Kubernetes (cont.)



Challenge

- Biggest challenge fro most applications is horizontal scaling
- For Pokemon Go, vertical scaling was an also a major challenge, because of real-time activity in gaming environment from millions of users
- Niantic were prepared for traffic disasters of up to x5 times

Solution

- Thanks to Kubernetes, Niantic were able to handle x50 times traffic

Case Study: Booking.com

Challenge

- In 2016, Booking.com migrated to an OpenShift platform
  - ➢ which gave product developers faster access to infrastructure.
- Kubernetes was abstracted away from the developers, the infrastructure team became a “knowledge bottleneck” when challenges arose.
  - ➢ Trying to scale that support wasn't sustainable.

Case Study: Booking.com (cont.)

Solution

- After a year operating OpenShift, the platform team decided to build its own vanilla Kubernetes platform—and ask developers to learn some Kubernetes in order to use it.
- “This is not a magical platform,” says Ben Tyler, Principal Developer, B Platform Track.
- “We're not claiming that you can just use it with your eyes closed. Developers need to do some learning, and we're going to do everything we can to make sure they have access to that knowledge.”

Case Study: Booking.com (cont.)

Impact

- Despite the learning curve, there's been a great uptick in adoption of the new Kubernetes platform.
- Before containers, creating a new service could take a couple of days if the developers understood Puppet, or weeks if they didn't.
- On the new platform, it can take as few as 10 minutes. About 500 new services were built on the platform in the first 8 months.

Case Study: Huawei

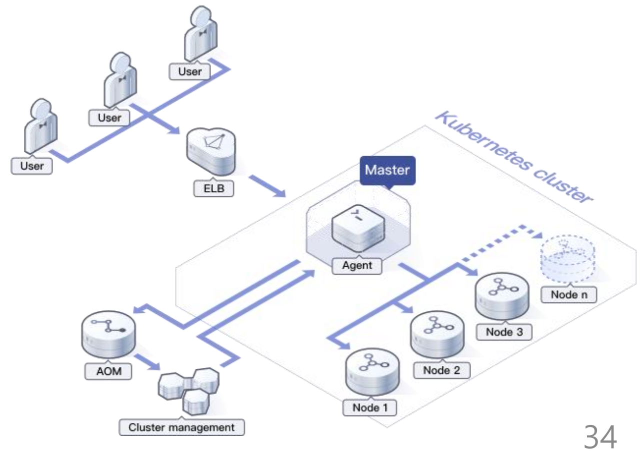
Challenge

- In order to support its fast business development around the globe
  - ➢ Huawei has eight data centers for its internal I.T. department, which have been running 800+ applications in 100K+ VMs to serve these 180,000 users.
- With the rapid increase of new applications
  - ➢ The cost and efficiency of management and deployment of VM-based apps all became critical challenges for business agility.

Case Study: Huawei (cont.)

Solution

- After deciding to use container technology
    - Huawei began moving the internal I.T. department's applications to run on Kubernetes.
    - So far, about 30 percent of these applications have been transferred to cloud native.

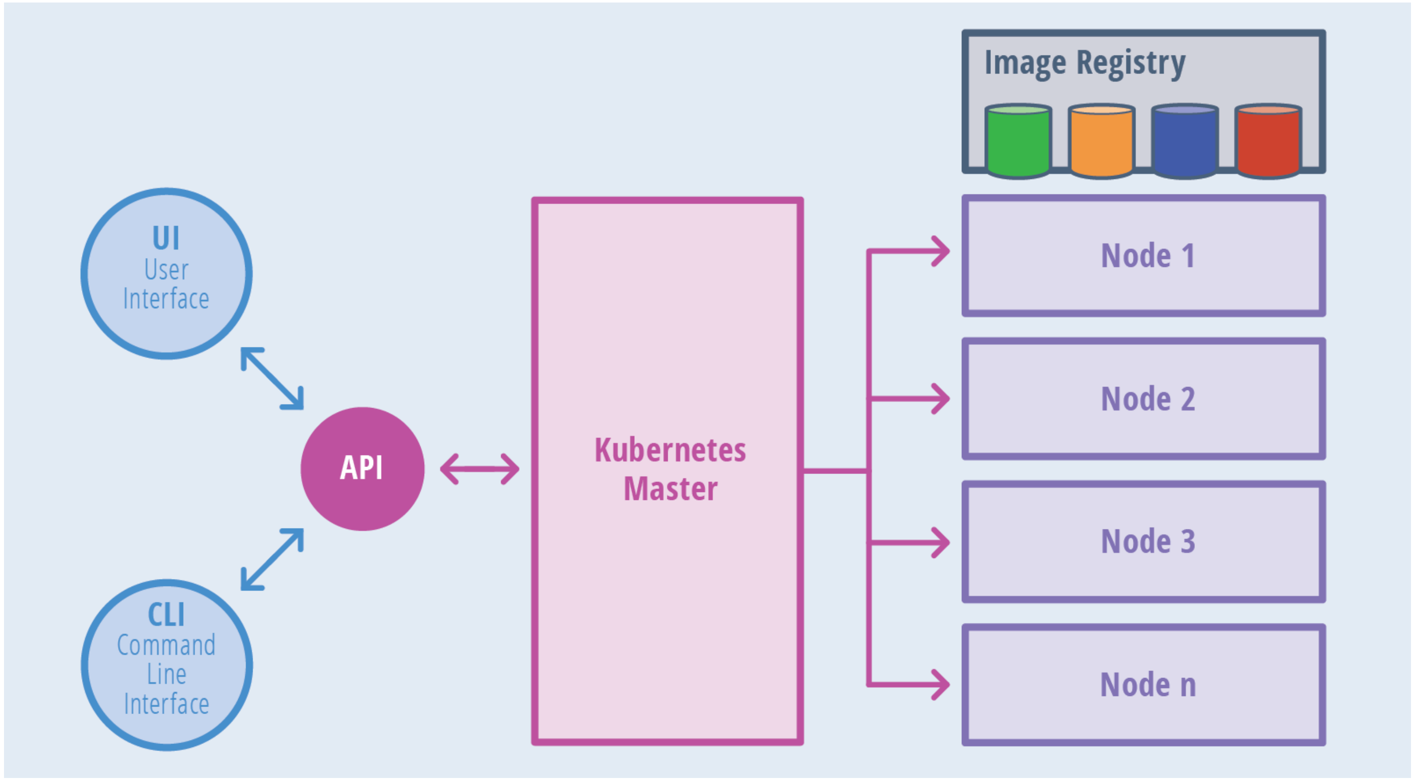


Case Study: Huawei (cont.)

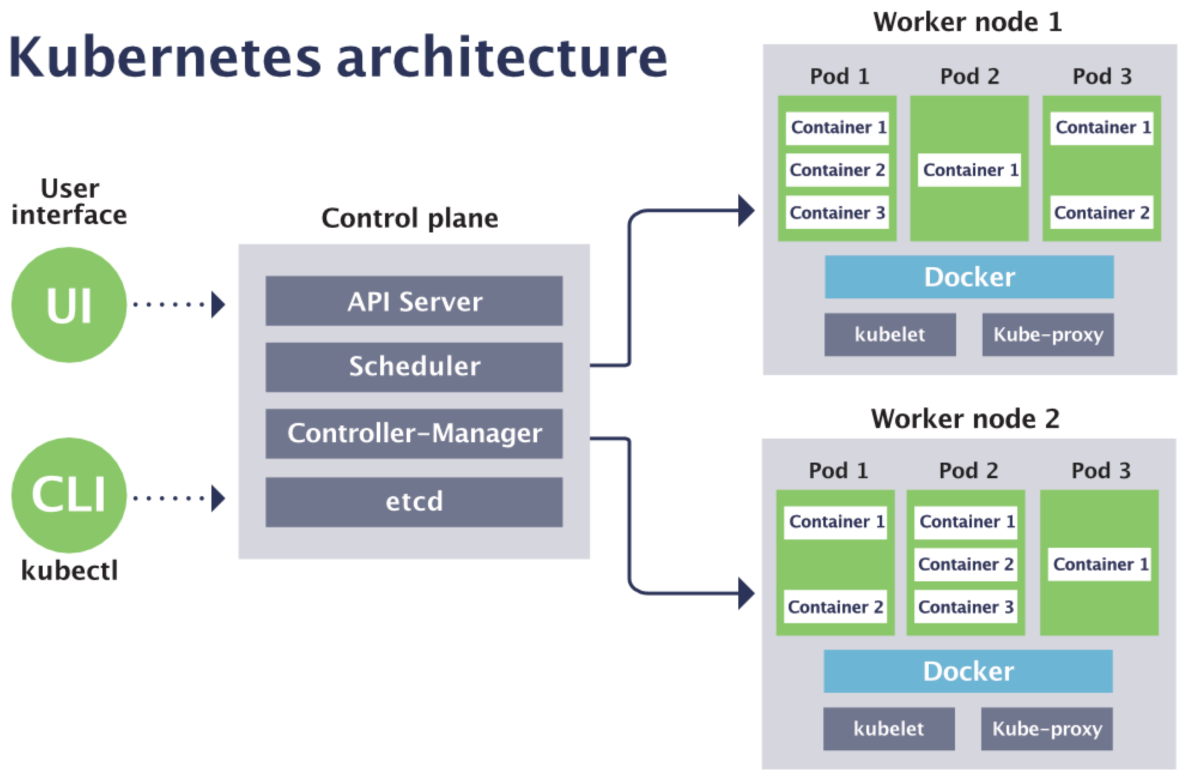
Impact

- By the end of 2016, Huawei's internal I.T. department managed more than 4,000 nodes with tens of thousands containers using a Kubernetes-based Platform as a Service (PaaS) solution
- The global deployment cycles decreased from a week to minutes
- The efficiency of application delivery has been improved 10 fold
- They saw significant operating expense spending cut, in some circumstances 20-30 percent

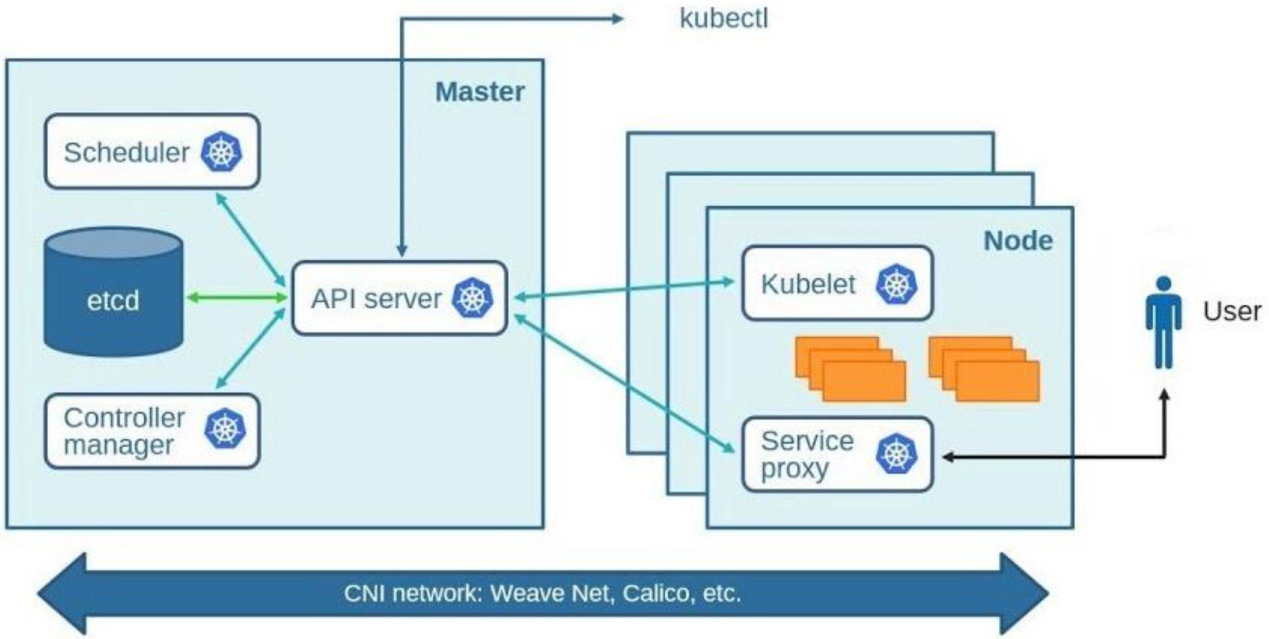
Kubernetes Architecture



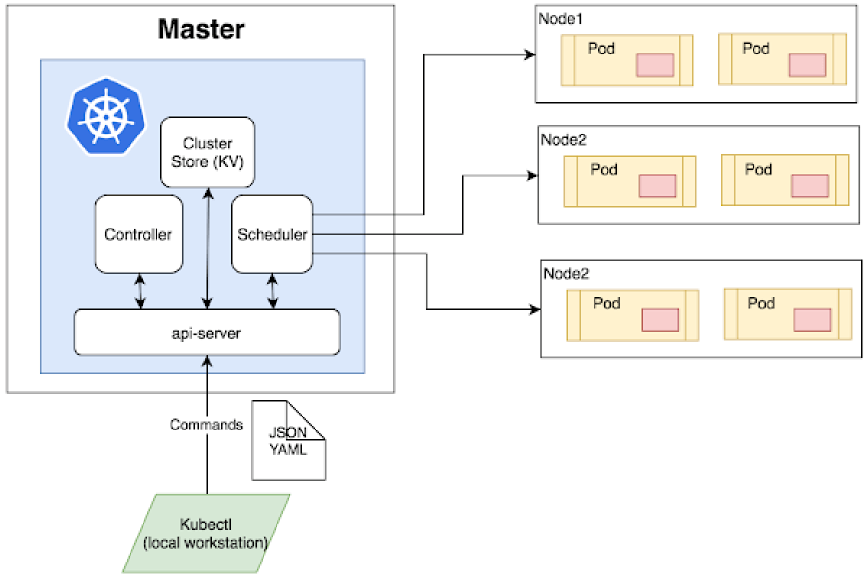
Kubernetes Architecture (cont.)



Kubernetes Architecture (cont.)



Kubernetes Components



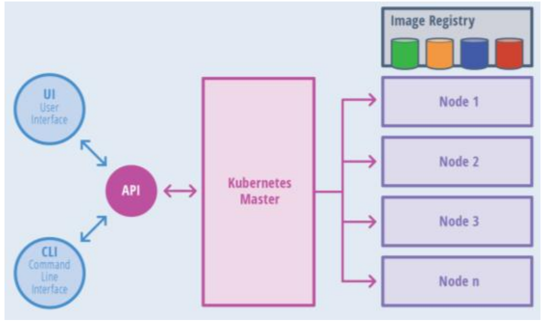
- Cluster:
  - ➢ A collection of hosts(servers) that aggregates their available resources.
- Master:
  - ➢ A collection of components which make up the control panel of Kubernetes.
- Node:
  - ➢ A single host which is capable of running on a physical or virtual machine.
- Namespace:
  - ➢ A logical cluster or environment.

Kubernetes Components (cont.)

- When you deploy Kubernetes, you get a cluster.
- A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications.
    - ➢ Every cluster has at least one worker node;
    - ➢ The worker node(s) host the Pods that are the components of the applications.
- The master manages the worker nodes and the Pods in the cluster.

Kubernetes Architecture (cont.)

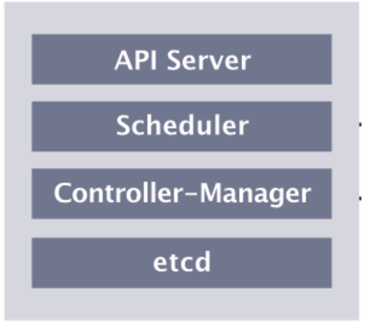
- The master is responsible for:
  - exposing the Kubernetes (REST) API,
  - scheduling the applications,
  - managing the cluster,
  - directing communications across the entire system,
  - monitoring the containers running in each node as well as the health of all the registered nodes.



- The nodes that are responsible for scheduling and running the containerized applications
- Container images, which act as the deployable artifacts, must be available to the Kubernetes cluster through a private or public image registry.

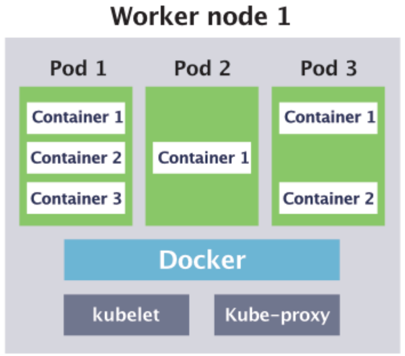
Kubernetes Master

- The Kubernetes master runs the following components that form the control plane:
  - **API server:** the front-end for the Kubernetes control plane that exposes the Kubernetes API
    - kube-apiserver
    - designed to scale horizontally
  - **etcd:** is a persistent, lightweight, distributed key-value data store that maintains the entire state of the cluster at any given point of time
  - **scheduler:** watches for newly created Pods with no assigned node, and selects a node for them to run on
    - kube-scheduler
  - **controller:** control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current cluster state to the desired cluster state



Kubernetes Nodes

- Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment.
- Kubernetes Nodes components:
  - ➢ kubelet: agent that makes sure that containers are running in a Pod.
  - ➢ kube-proxy: implemented as a network proxy and a load balancer.
    - o It routes traffic to the appropriate container based on its IP address and the port number of the incoming request.
    - o Part of the Kubernetes Service concept.
  - ➢ container runtime: the software that is responsible for running containers.
  - ➢ Kubernetes supports several container runtimes:
    - o Docker,
    - o containerd,
    - o CRI-O,
    - o any implementation of the Kubernetes CRI (Container Runtime Interface).

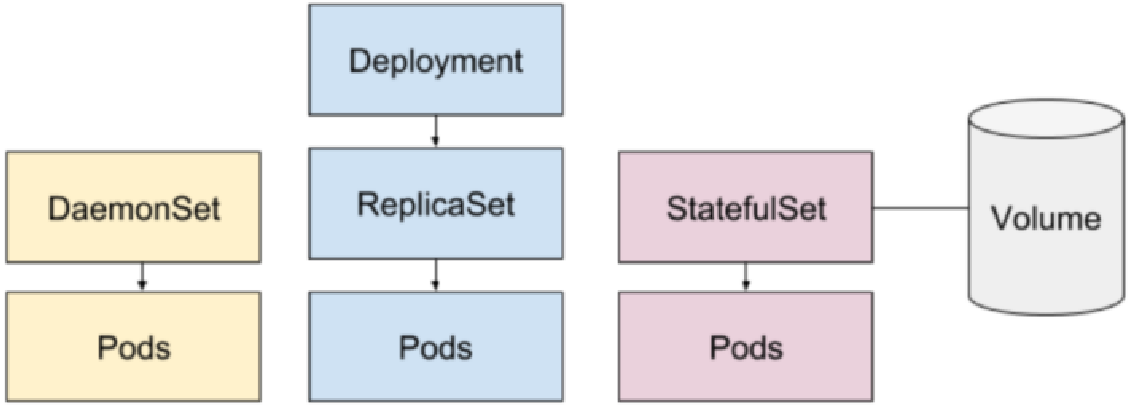


Kubernetes Object

- Kubernetes objects are persistent entities provided by Kubernetes for deploying, maintaining, and scaling applications.
- Kubernetes uses these entities to represent the state of your cluster.
- The objects can describe:
    - ➢ What containerized applications are running (and on which nodes)
    - ➢ The resources available to those applications
    - ➢ The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance
- To work with Kubernetes objects (i.e., to create, modify, or delete them) you'll need to use the Kubernetes API. When you use the kubectl command-line interface, for example, the CLI makes the necessary Kubernetes API calls for you.

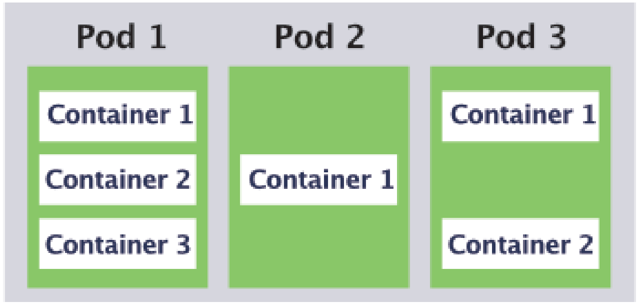
Basic Kubernetes Objects

- Pod
- Deployment, ReplicaSet
- DaemonSet
- StatefulSet
- Service
- Secret



Pods

- Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.
- A Pod is a group of one or more containers, with shared storage/ network resources, and a specification for how to run the containers.
- A Pod's contents are always co-located and co-scheduled.
- To create and manage multiple Pods, Kubernetes defines multiple resource types:
    - ➢ Deployment
    - ➢ StatefulSet
    - ➢ DaemonSet



Deployment

- Deployment: represents a set of multiple, identical Pods with no unique identities.
  - ➢ It runs multiple replicas Pods and automatically replaces any instances that fail or become unresponsive.
  - ➢ Deployments ensure that one or more instances of Pods are available to serve user requests.
- PodTemplates are specifications for creating Pods, and are included in resource objects such as Deployment object.
- A ReplicaSet is the next-generation of ReplicationControllers
  - ➢ It ensures that a specified number of pod replicas are running at any given time.

Pod Management

- StatefulSet: manages deployment and scaling of a set of Pods, with durable storage and persistent identifiers for each pod.
    - ➢ Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its Pods.
- DaemonSet: ensures that all (or some) nodes run a copy of a Pod.
    - ➢ As nodes are added to the cluster, Pods are added to them.
    - ➢ As nodes are removed from the cluster, those Pods are garbage collected.
    - ➢ Deleting a DaemonSet will clean up the Pods it created.

Controller

- Controllers are control loops that watch the state of the cluster, then make or request changes where needed.
- A controller tracks at least one Kubernetes resource object.
- The controller(s) for that resource are responsible for making the current state come closer to that desired state (specified in the spec field).
- Kubernetes comes with a set of controllers that run inside the kubecontroller-manager
- The Deployment controller is an example of controller that come as part of Kubernetes itself (“built-in” controllers).

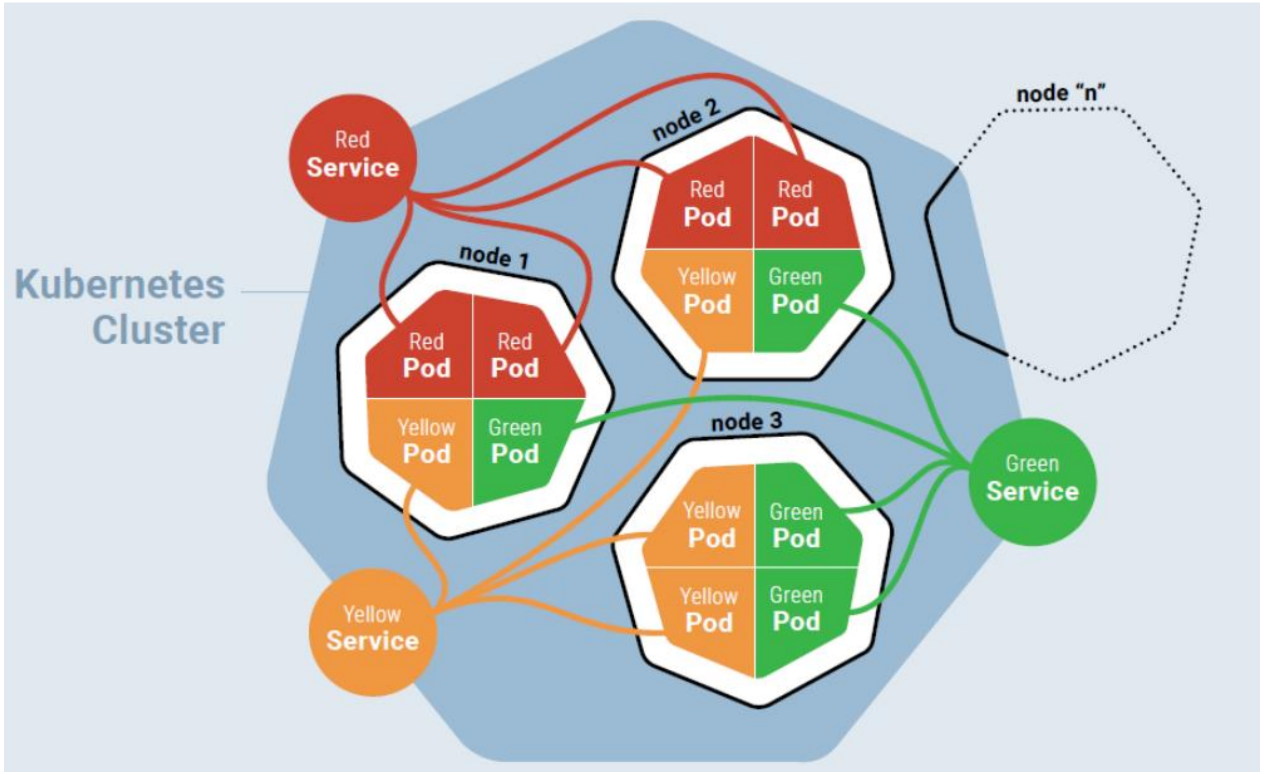
Service

- The services model relies upon the most basic, though most important, aspect of services: **discovery**.
- a Service is an abstraction which defines a logical set of Pods and a policy by which to access them.
- The set of Pods targeted by a Service is usually determined by a **selector**.
- Services ensure that traffic is always routed to the appropriate Pod within the cluster, regardless of the node on which it is scheduled.
- Each service exposes an **IP address**, and may also expose a **DNS endpoint**, both of which will never change.
    - ➢ Internal or external consumers that need to communicate with a set of pods will use the service's IP address, or its more generally known DNS endpoint.

Secret

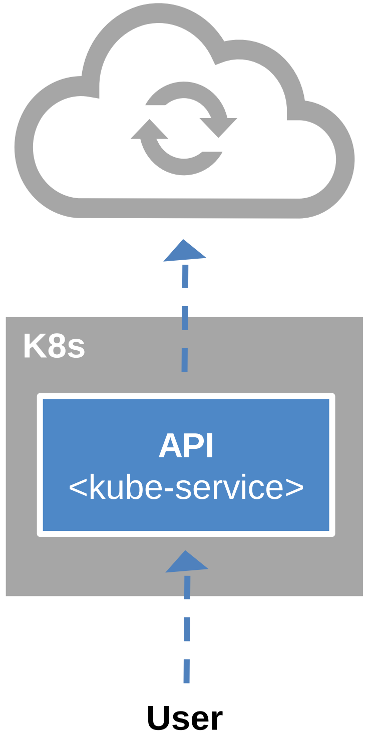
- Secrets are secure objects which store sensitive data, such as passwords, OAuth tokens, and SSH keys, in your clusters.
- Storing sensitive data in Secrets is more secure than plaintext in Pod specifications.
- Using Secrets gives you control over how sensitive data is used, and reduces the risk of exposing the data to unauthorized users.

Kubernetes Cluster



Kubernetes - Advantages

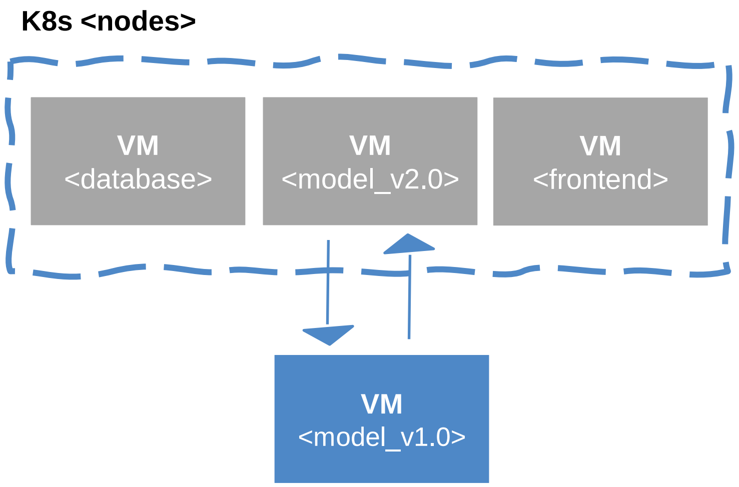
- There are many reasons why people come to use containers and container APIs like Kubernetes:
    1. **Velocity**
    2. **Scaling** (of both software and teams)
    3. **Abstracting the infrastructure**
    4. **Efficiency**
- All these aspects relate to each other to speed up process that can reliably deploy software.



Kubernetes - Advantages - Velocity

Velocity is enabled by:

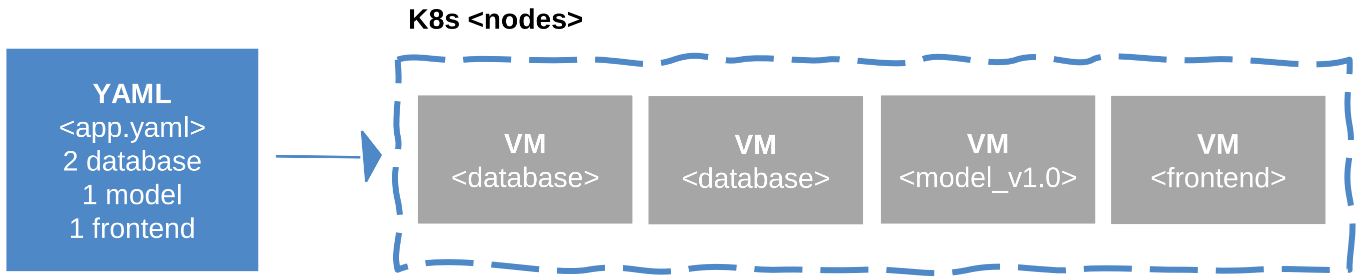
- **Immutable system:** you can't change running container, but you create a new one and replace it in case of failure (allows for keeping track of the history and load older images)



Kubernetes - Advantages - Velocity (cont.)

Velocity is enabled by:

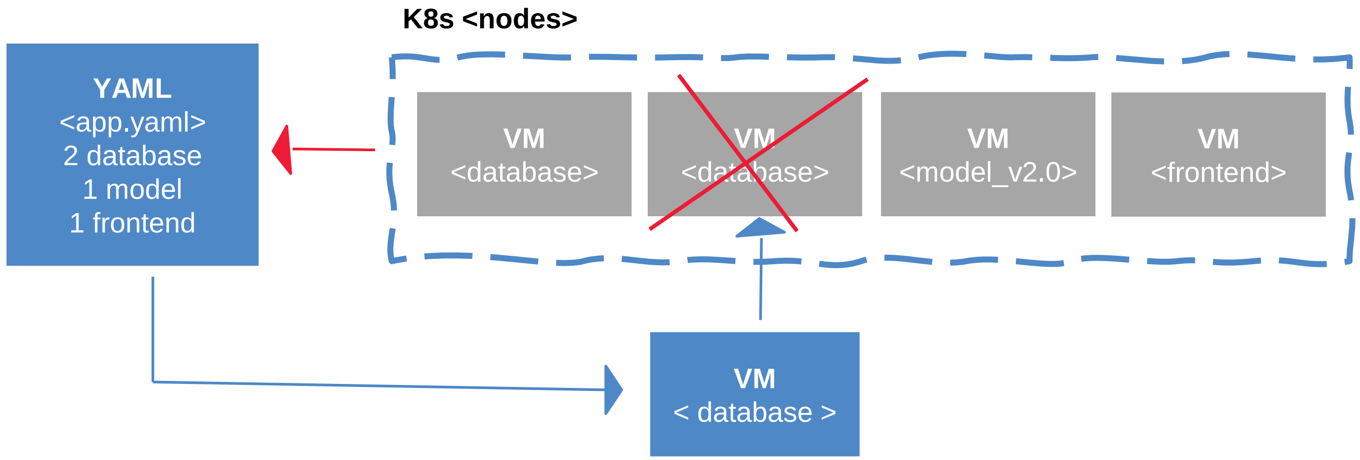
- **Declarative configuration:** you can define the desired state of the system restating the previous declarative state to go back. Imperative configuration are defined by the execution of a series of instructions, but not the other way around.



Kubernetes - Advantages - Velocity (cont.)

Velocity is enabled by:

- **Online self-healing systems:** K8s takes actions to ensure that the current state matches the desired state (as opposed to an operator enacting the repair)



Kubernetes - Advantages - Scaling

Kubernetes provides numerous advantages to address scaling:

- **Decoupled architectures:** each component is separated from other components by defined APIs and service load balancers.
- **Easy scaling for applications and clusters:** simply changing a number in a configuration file, K8s takes care of the rest (part of declarative).
- **Scaling development teams with microservices:** small team is responsible for the design and delivery of a service that is consumed by other small teams.

Kubernetes - Advantages - Scaling (cont.)

Kubernetes provides numerous **abstractions** and APIs that help building these decoupled microservice architectures:

- **Pods** can group together container images developed by different teams into a single deployable unit (similar to docker-compose)
- **Other services to isolate** one microservice from another such (e.g. load balancing, naming, and discovery)
- **Namespaces** control the interaction among services
- **Ingress** combine multiple microservices into a single externalized API (easy-to-use frontend)

K8s provides full spectrum of solutions between doing it “the hard way” and a fully managed service

Kubernetes - Advantages - Abstraction

Kubernetes allows to build, deploy, and manage your application in a way that is portable across a wide variety of environments.

The move to application-oriented container APIs like Kubernetes has two concrete benefits:

- **separation:** developers from specific machines
- **portability:** simply a matter of sending the declarative config to a new cluster

Kubernetes - Advantages - Efficiency

There are concrete economic benefit to the abstraction because tasks from multiple users can be packed tightly onto fewer machines:

- **Consume less energy** (ratio of the useful to the total amount)
- **Limit costs of running a server** (power usage, cooling requirements, datacenter space, and raw compute power)
- **Create quickly a developer’s test environment** as a set of containers
- **Reduce cost of development instances in your stack,** liberating resources to develop others that were cost-prohibitive

Deploying a Kubernetes Cluster

To deploy your cluster you must install Kubernetes. In the exercise you are going to use minikube to deploy a cluster in local mode.

- After installing minikube, use *start* to begin your session creating a virtual machine, *stop* to interrupt it, and *delete* to remove the VM. Below are the commands to execute these tasks:

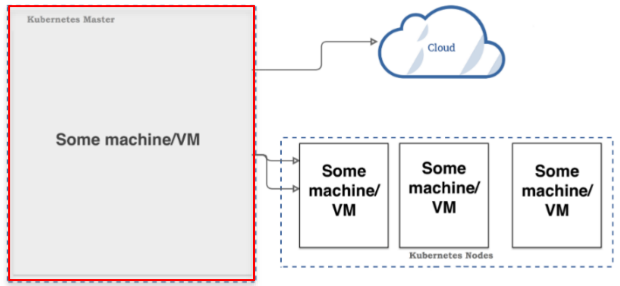
```
$ minikube start
$ minikube stop
$ minikube delete
```

Deploying a Kubernetes Cluster (cont.)

You can easily access the Kubernetes Client using the following command:

- to check your cluster status use:

```
$ kubectl get componentstatuses
```



- and should see output below:

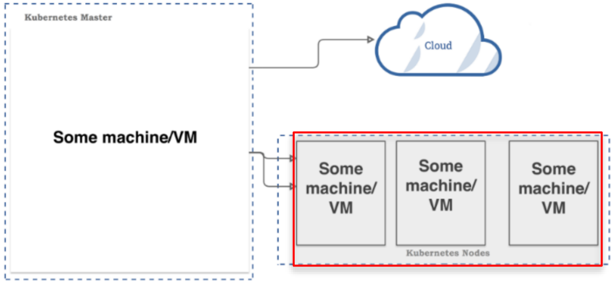
```
NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}
```

Deploying a Kubernetes Cluster (cont.)

You can easily access the Kubernetes Client using the following command:

- to list the nodes in your cluster use:

```
$ kubectl get nodes
```



- and should see output below:

```
NAME          STATUS          AGE        VERSION
kubernetes    Ready,master    45d        v1.12.1
node-1        Ready           45d        v1.12.1
node-2        Ready           45d        v1.12.1
node-3        Ready           45d        v1.12.1
```

Common kubectl Commands

- **Let’s practice Kubernetes!** Useful commands to complete the exercise:

```
$ kubectl create -f app-db-deploymnet.yaml
$ kubectl get deployment
$ kubectl get pods
$ kubectl get pods /
  -o=custom-columns=NAME:.metadata.name,IP:.status.podIP
$ kubectl create -f app-server-deploymnet.yaml
```

Common kubectl Commands (cont.)

- **Let's practice Kubernetes!** Useful commands to complete the exercise:

```
$ kubectl expose deployment /
  app-deployment --type=LoadBalancer --port=8080
$ kubectl get services
$ kubectl delete service app-deployment
$ kubectl delete deployment app-server-deployment
$ kubectl delete deployment app-db-deployment
```

Kubernetes - Advantages

- Easy organization of service with pods
- Largest community among container orchestration tools
- Kubernetes can run on-premises bare metal, OpenStack, public clouds Google, Azure, AWS, etc.
- Avoid vendor lock issues

Q & A