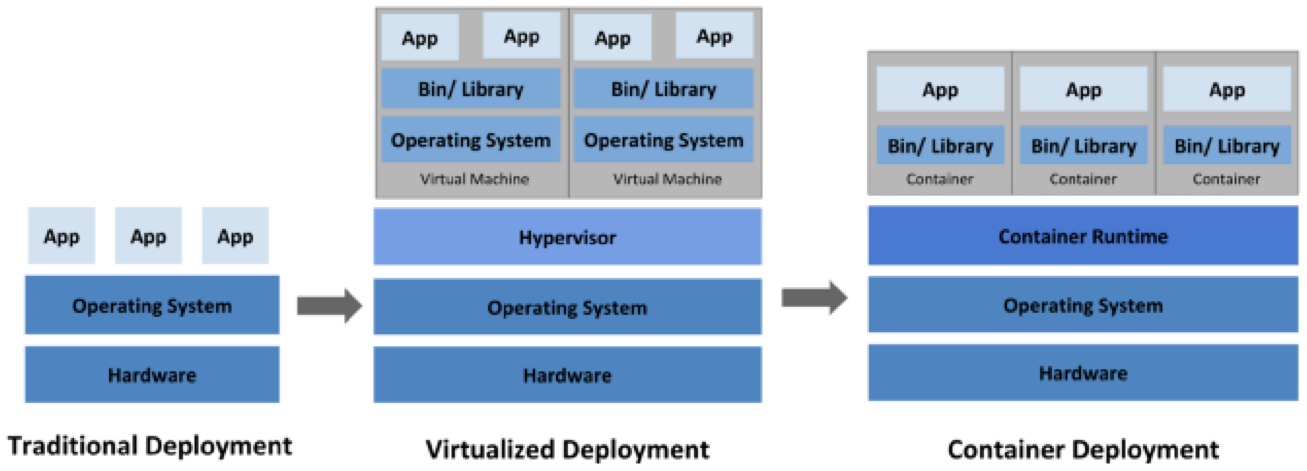
Kubernetes — Deep dive

Siddharth Patnaik Follow
Oct 17, 2019 · 6 min read

Introduction to Kubernetes

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. Its the first project to get graduated at https://www.cncf.io/. It was originally developed at Google.

A brief history of containerization



Traditional deployment

Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues

Virtualized Deployment

It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

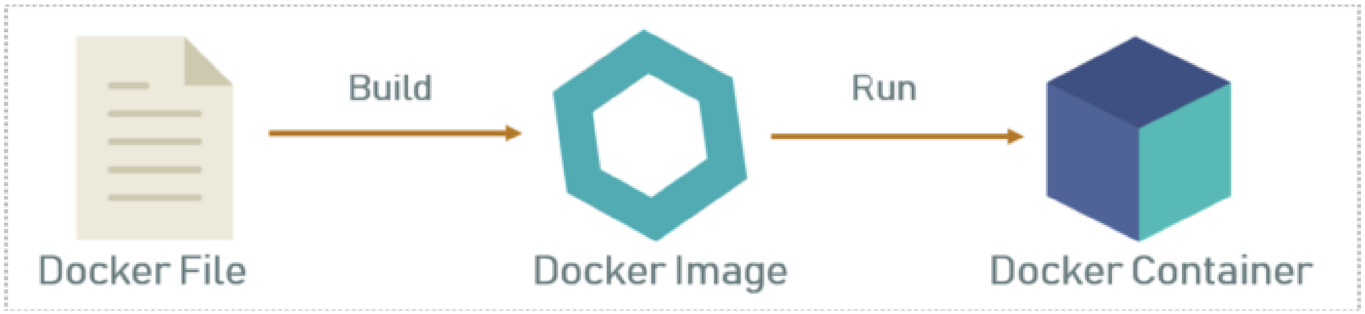
Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.

Container deployment

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, CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.

What is Docker?

Docker is a platform which packages an application and all its dependencies together in the form of containers. This containerization aspect of Docker ensures that the application works in any environment. Docker engine virtualises at OS level. Kubernetes uses Docker as the underlying container platform.



Container Orchestration Engine

Container Orchestration Engine automates deploying, scaling, and managing containerized applications on a group of servers. They offer the following features:

> *Clustering — A cluster consists of a set of master nodes and a set of worker nodes where the application is run*
>
> *Scheduling — COE takes care of scheduling the containers to appropriate worker nodes*
>
> *Scalability — They support scalability both at the container level and node level*
>
> *Load Balancing — Multiple instances of an application are run as containers. load balancer takes care of routing the traffic across these containers over multiple worker nodes*
>
> *Fault tolerance — Monitors the health of worker nodes and containers and make sure that the defined number of containers are always up and running*
>
> *Deployment — Supports multiple deployment strategies like rolling updates, Blue-Green, Canary etc*

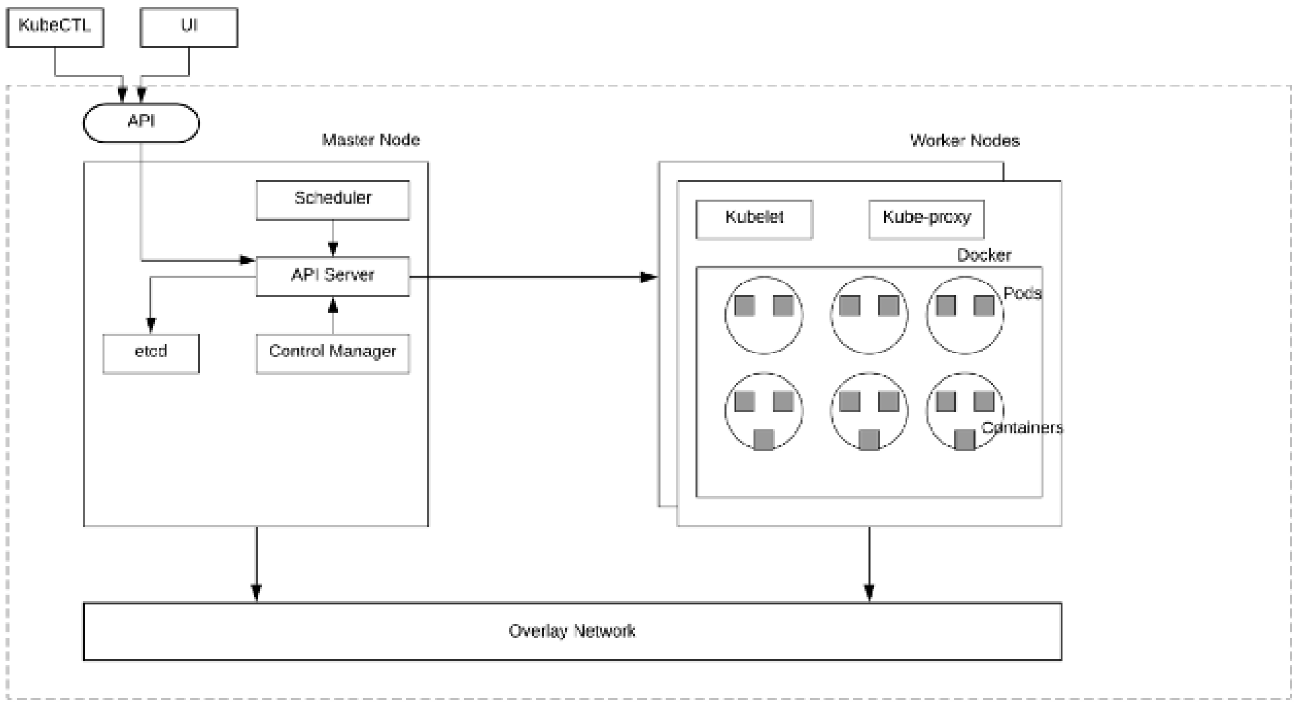
Following are the popular container orchestration engines:

- Kubernetes
- Docker swarm
- Apache Mesos

Why Kubernetes?

- Was originally designed at Google. Have been running at Google for years now. Battle-tested with applications like Gmail, youtube, etc
- The first project to be graduated at cncf. Very active development ecosystem. Backed by industry leaders like Google, Microsoft, Amazon, IBM, Oracle, etc
- Matured & stable. Used by most of the big companies today

Kubernetes Architecture



Kubernetes Architecture

A cluster contains 1 or more master nodes and many worker nodes. Worker nodes are the work hostess where the applications are run.

> ***Kubernetes Master***

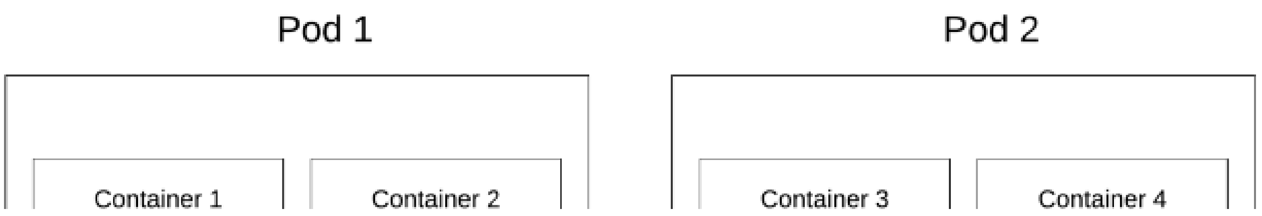
Responsible for managing the cluster. It coordinates with all the activities inside the cluster and interacts with the workers. Following are the major components of the master:

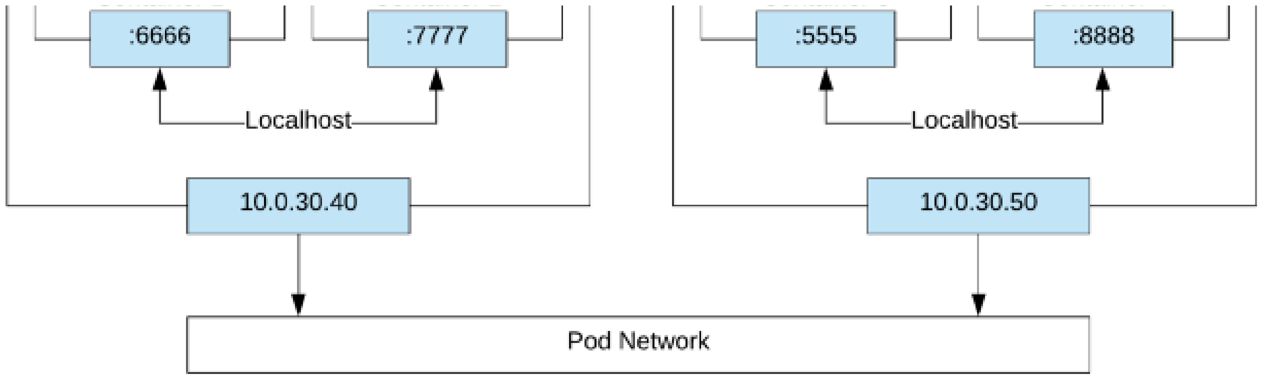
> *API Server — Exposes various APIs for working with the cluster*
>
> *Scheduler — Responsible for scheduling Pods across the worker nodes*
>
> *Control Manager — Works internally with various controllers like node controller, replication controller to manage the overall health of the cluster*
>
> *etcd — A distributed key-value data store where the cluster state information is stored*

Kubernetes Worker

Every worker node has a container runtime. Both Docker and rocket are supported. In addition to this, there are the following components which run on each worker nodes

> *Kubelet — An agent that runs on each node in the cluster. It makes sure that containers are running in a pod*
>
> *Kubeproxy — A network proxy that runs on each node in your cluster, implementing a part of the Kubernetes Service concept. It maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.*
>
> *Containers — They are the application runtimes. Designed to run microservices. Both docker and rocket-based containers are supported*
>
> *Pods — They are the atomic unit of scheduling in Kubernetes. Each pod contains one or more containers.*





Pod networking

Pod networking

- Every pod has an IP address
- Every container inside the pod inherits the ip and has a port
- The container inside a pod communicate with each other inside a pod using localhost (Intra pod communication)
- Containers across the pods communicate with each other using their IP address (inter pod communication)

ReplicaSet

Ensures that a specified number of pods are running at any time a. If there are excess Pods, they get killed and vice versa b. New Pods are launched when they get fail, get deleted or terminated

Deployment

A *Deployment* provides declarative updates for Pods and ReplicaSets. You describe the *desired state* in a Deployment, and the Deployment controller changes the actual state to the desired state at a controlled rate. You can define Deployments to create new ReplicaSets or to remove existing Deployments and adopt all their resources with new Deployments.

Let us look into a sample deployment called nginx-deployment.yaml :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

kubectl apply -f ~/Documents/nginx-deployment.yaml

Kubectl get pods -o wide

```
NAME                                READY     STATUS    RESTARTS   
AGE       IP             NODE

nginx-deployment-76bf4969df-6hpqr   1/1       Running   0          1m
10.244.5.14   aks-agentpool-22051123-1

nginx-deployment-76bf4969df-8qv2z   1/1       Running   0          1m
10.244.4.11   aks-agentpool-22051123-0

nginx-deployment-76bf4969df-sg787   1/1       Running   0          1m
10.244.3.12   aks-agentpool-22051123-3
```

DaemonSet

A DaemonSet ensures that all nodes run a single instance of a pod. This is useful for the following scenarios

- Node monitoring daemons like collectD

- Log collection daemons: Ex: fluentd

When a new node joins a cluster, DaemonSet creates the pod and when any node leaves a cluster, it destroys the pod.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-ds
spec:
template:
metadata:
      labels:
        name: fluentd
spec:
      containers:
      - name: fluentd
image: gcr.io/google-containers/fluentd-elasticsearch:1.20
selector:
    matchLabels:
      name: fluentd
```

Jobs

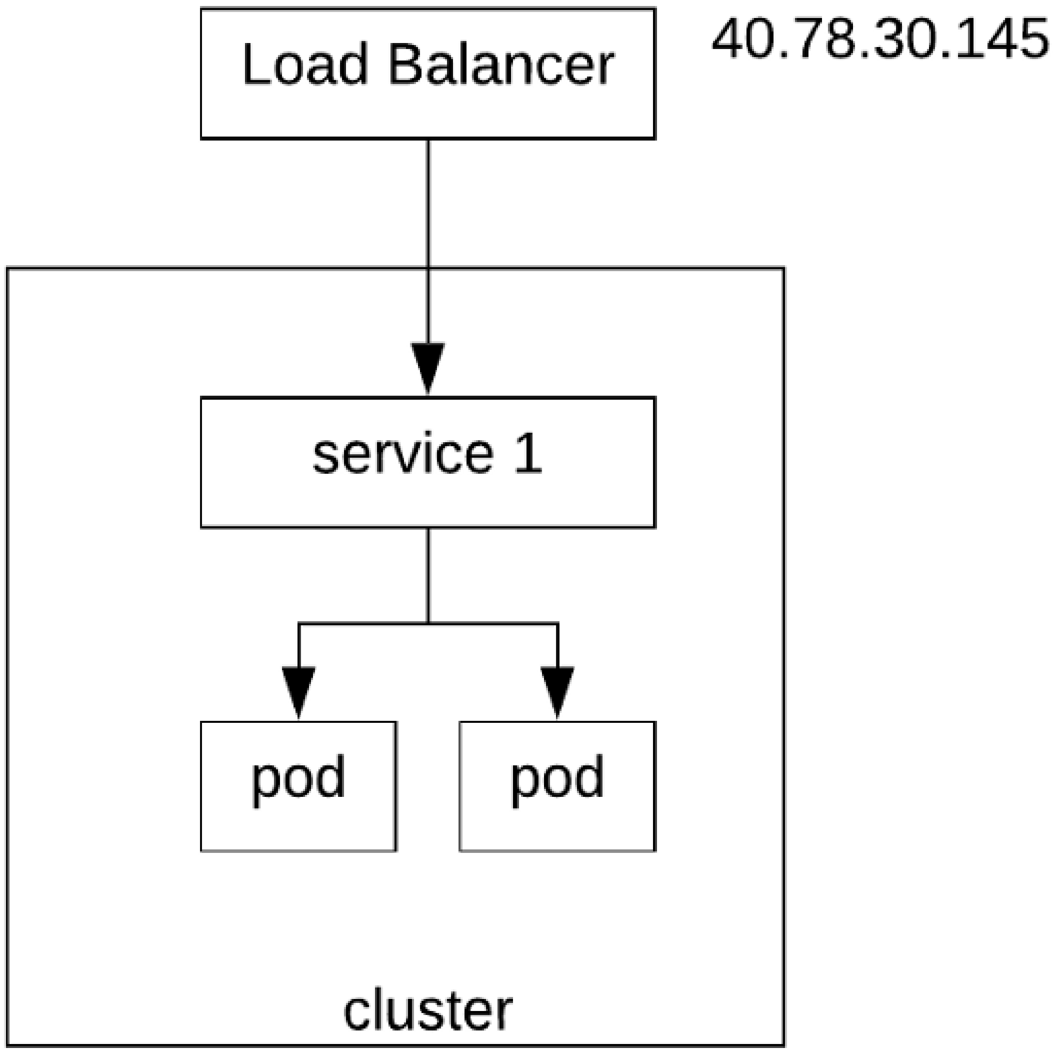
Jobs can be used for the following types of workloads

- A pod should kickoff, do its job and terminate
- A pod which should run on a defined schedule

```
apiVersion: batch/v1
kind: Job
metadata:
name: countdown
spec:
template:
metadata:
name: countdown
spec:
containers:
- name: counter
image: centos:7
command:
- "bin/bash"
- "-c"
```

```
  - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
  restartPolicy: Never
```

Load Balancer



Load balancer

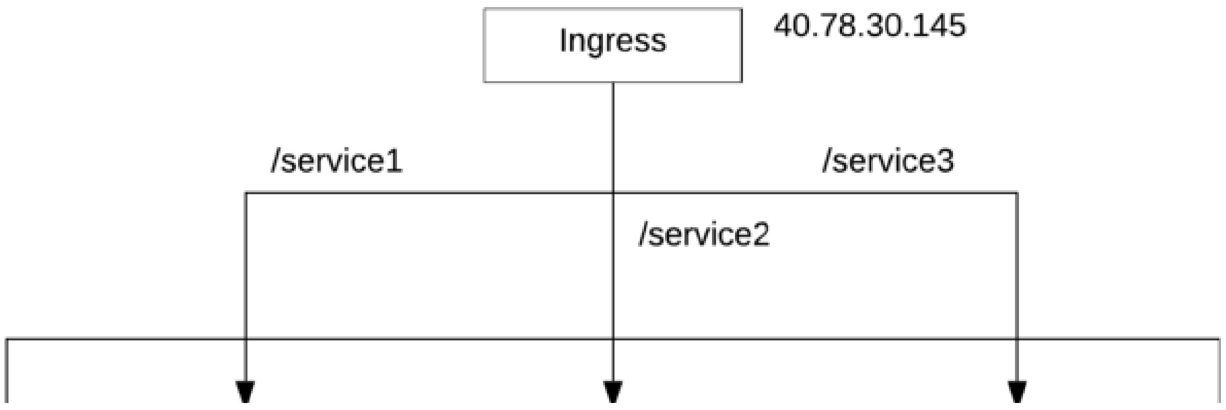
A LoadBalancer service is the standard way to expose a service to the internet. Based on the cloud provider, it will generate the respective load balancer and give you a single IP address that will forward all traffic to your service.

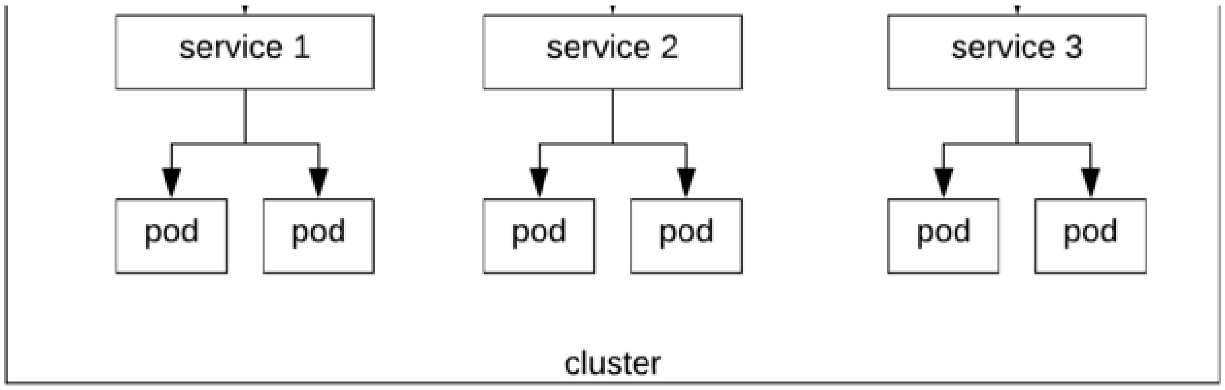
```
apiVersion: v1
kind: Service
metadata:
  name: helloindia
spec:
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    targetPort: 8080
    protocol: TCP
  selector:
    app: helloindia
```

Once deployed, get the service details

```
kubectl get svc

NAME         TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
AGE

helloindia   LoadBalancer   10.0.26.251   40.78.145.151
80:32076/TCP   15m

kubernetes   ClusterIP      10.0.0.1      <none>          443/TCP
24m
```

Ingress Controller





When you have multiple services deployed on a cluster and you want to expose one single entry point, you can use Ingress. Later you can define rules based on URL path to route the traffic to appropriate services

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /service1
        backend:
          serviceName: service1
          servicePort: 80
      - path: /service2
        backend:
          serviceName: service2
          servicePort: 80
      - path: /service3
        backend:
          serviceName: service3
          servicePort: 80
```

Kubernetes Docker Azure