

Kubernetes Basics

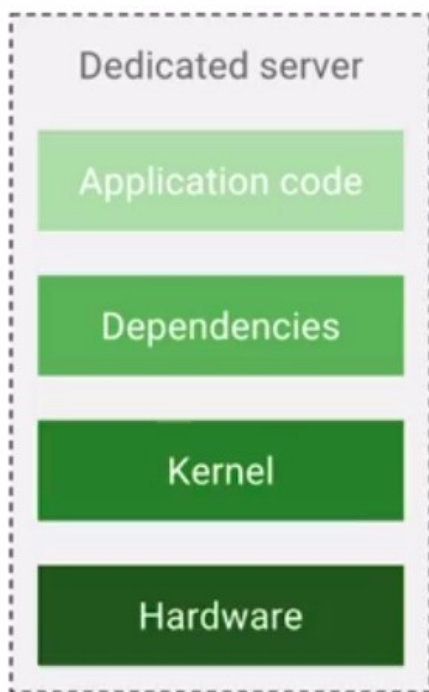
Before managing complete applications with Kubernetes and Container Engine, you'll want to know how containers work and what advantages they provide.

Introduction to containers and docker

What are Containers?

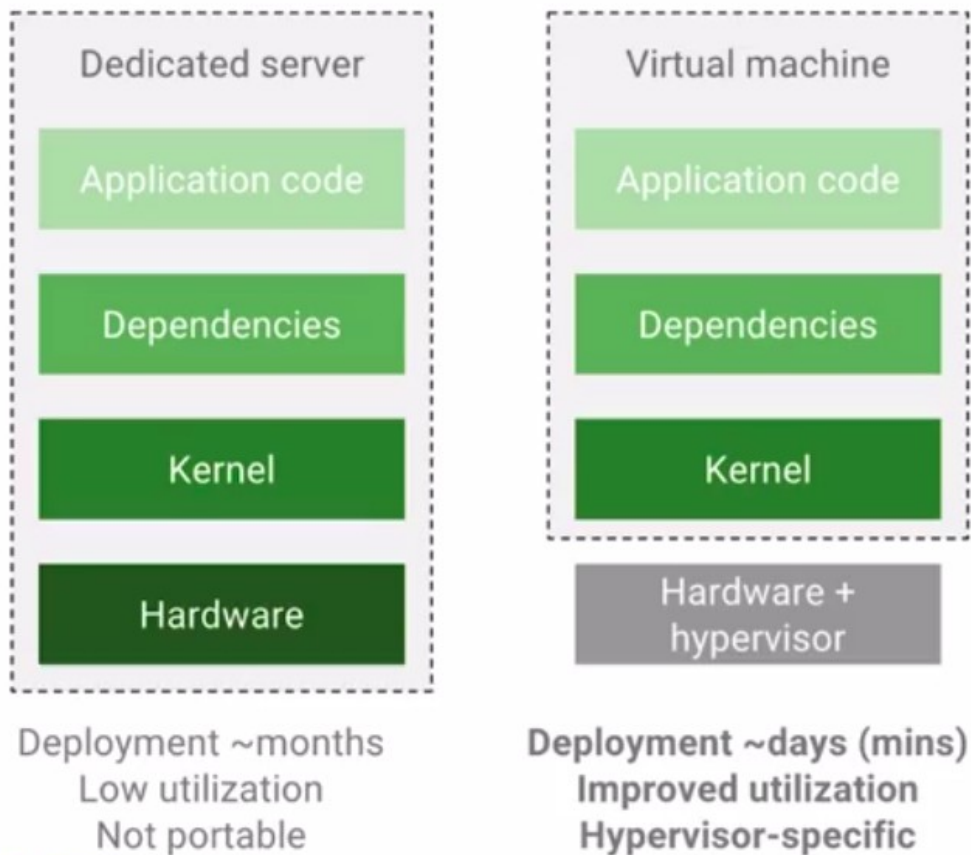
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Looking back, you used to build applications on individual servers

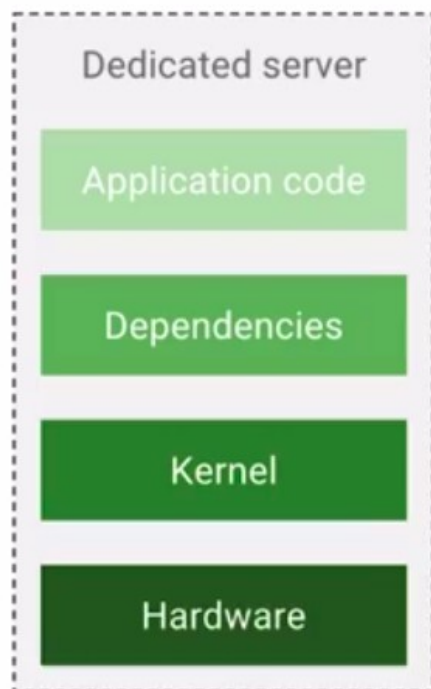


Deployment ~months
Low utilization
Not portable

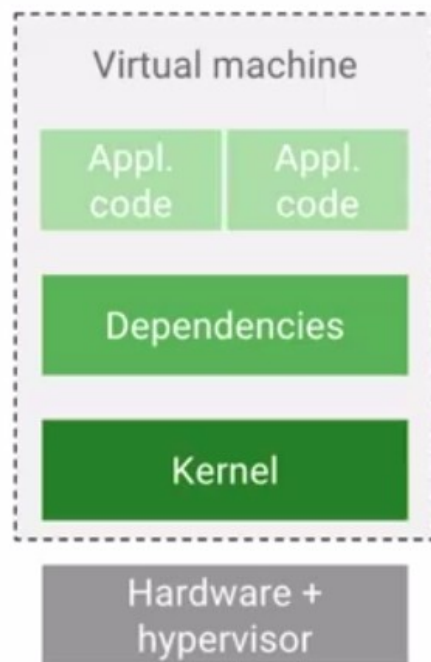
Then VMware popularized running multiple servers and operating systems on the same hardware



But it was difficult to run and maintain multiple applications on a single VM, even with policies

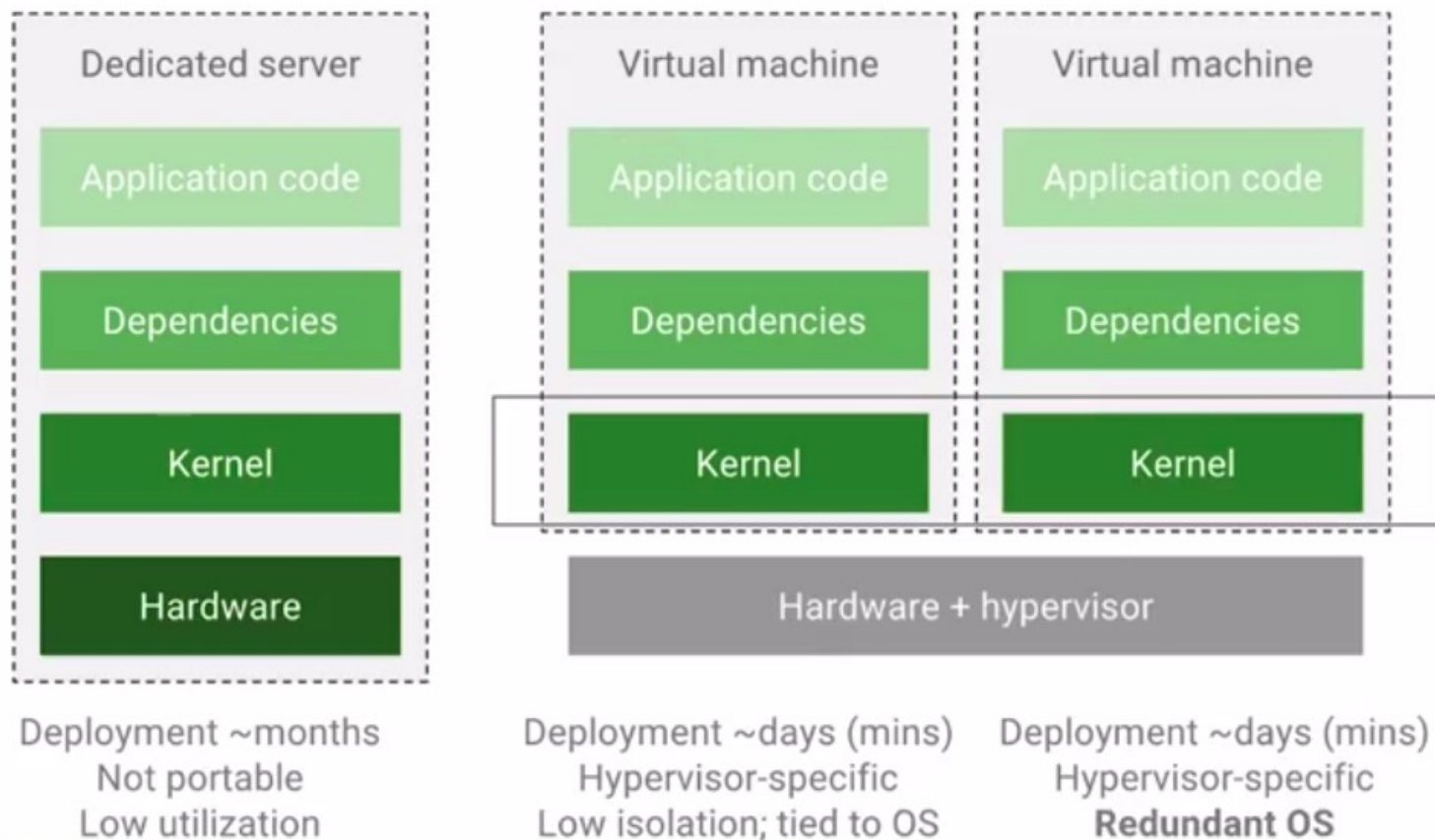


Deployment ~months
Low utilization
Not portable

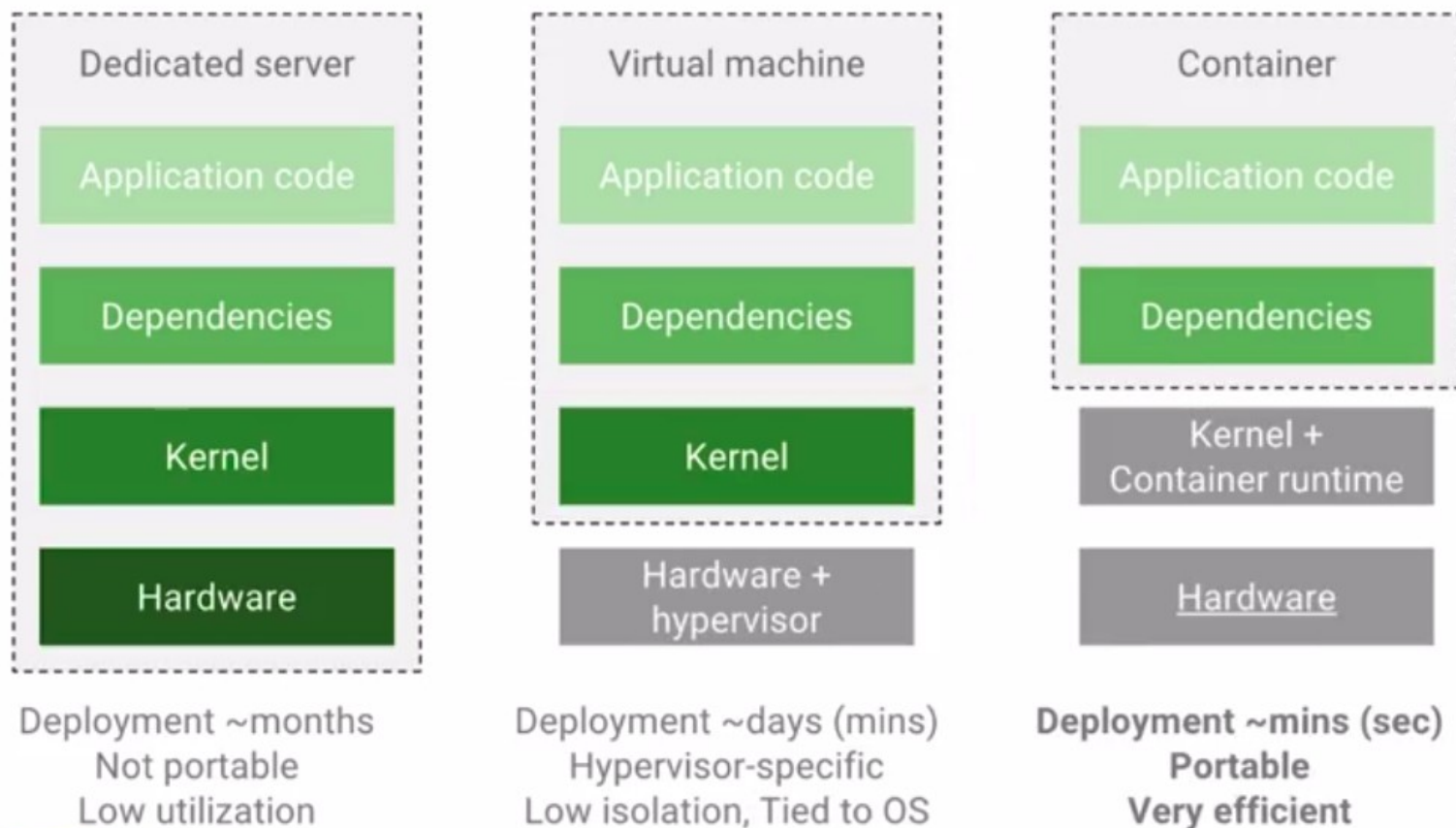


Deployment ~days (mins)
Hypervisor-specific
Low isolation; tied to OS

The VM-centric way to solve this is to run each app on its own server with its own dependencies, but that's wasteful

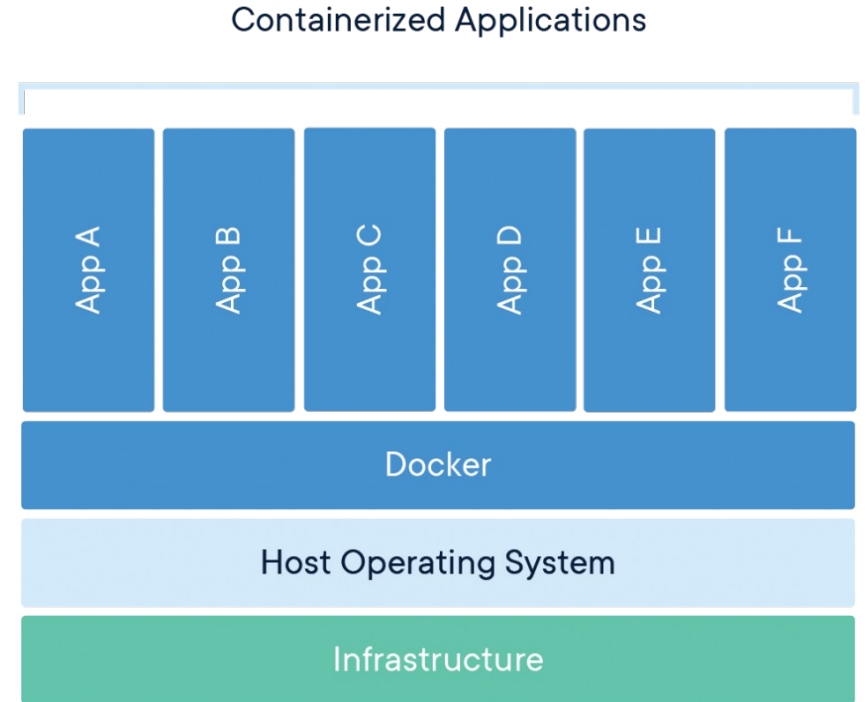


So you raise the abstraction one more level and virtualize the OS

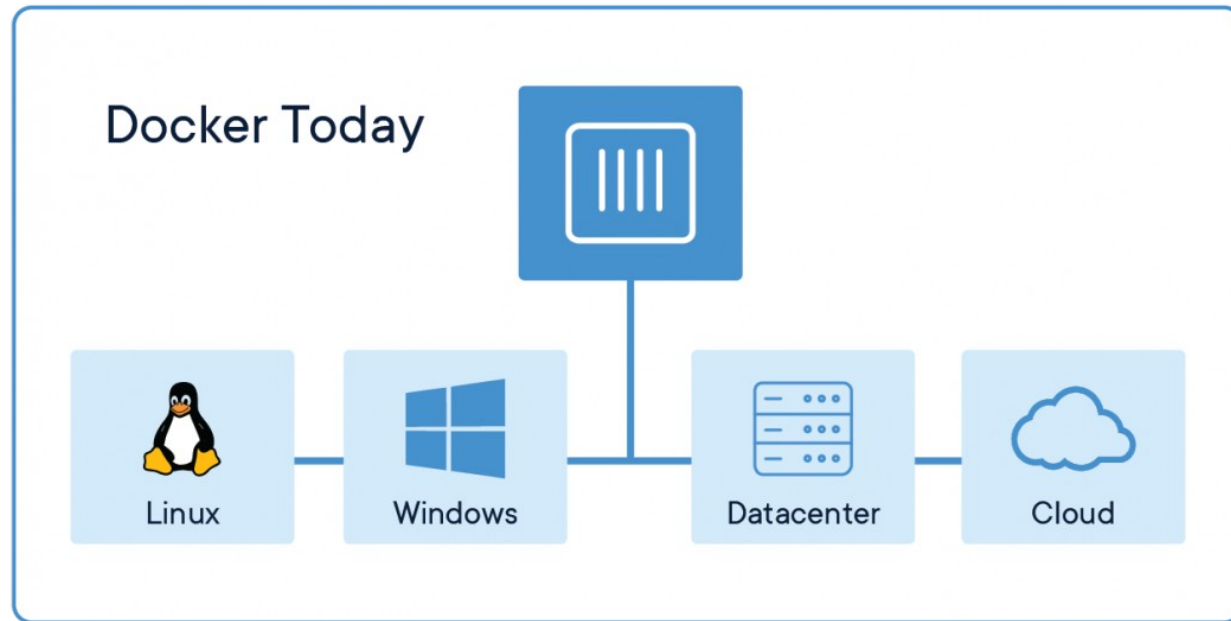


What is Docker?

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

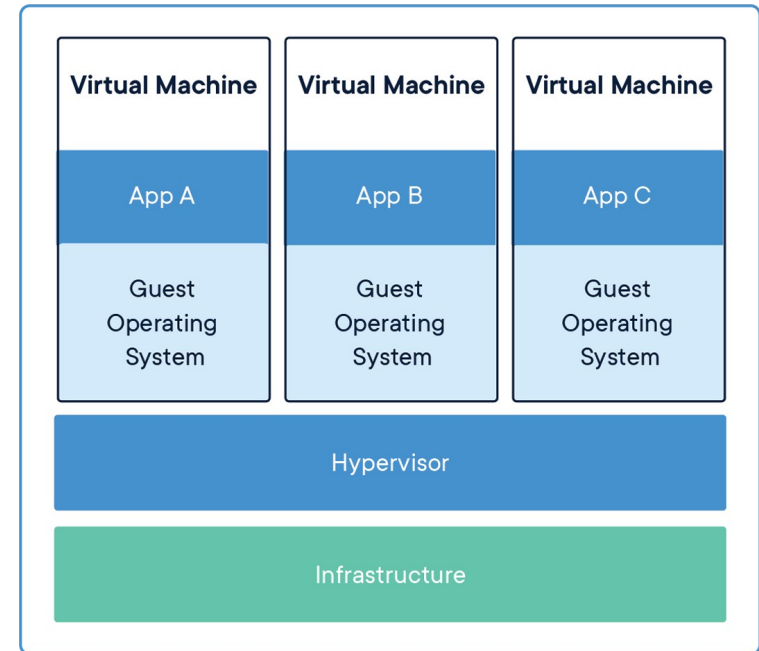
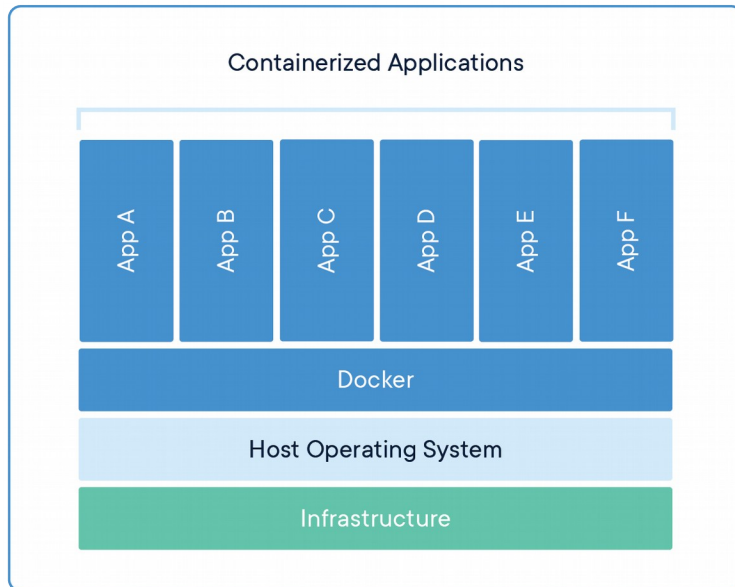


Docker Containers Are Everywhere: Linux, Windows, Data center, Cloud, Serverless, etc.



Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.



Docker is a platform for developers and sysadmins to **develop, deploy, and run** applications with containers. The use of Linux containers to deploy applications is called *containerization*. Containers are not new, but their use for easily deploying applications is.

Containerization is increasingly popular because containers are:

- Flexible: Even the most complex applications can be containerized.

- Lightweight: Containers leverage and share the host kernel.

- Interchangeable: You can deploy updates and upgrades on-the-fly.

- Portable: You can build locally, deploy to the cloud, and run anywhere.

- Scalable: You can increase and automatically distribute container replicas.

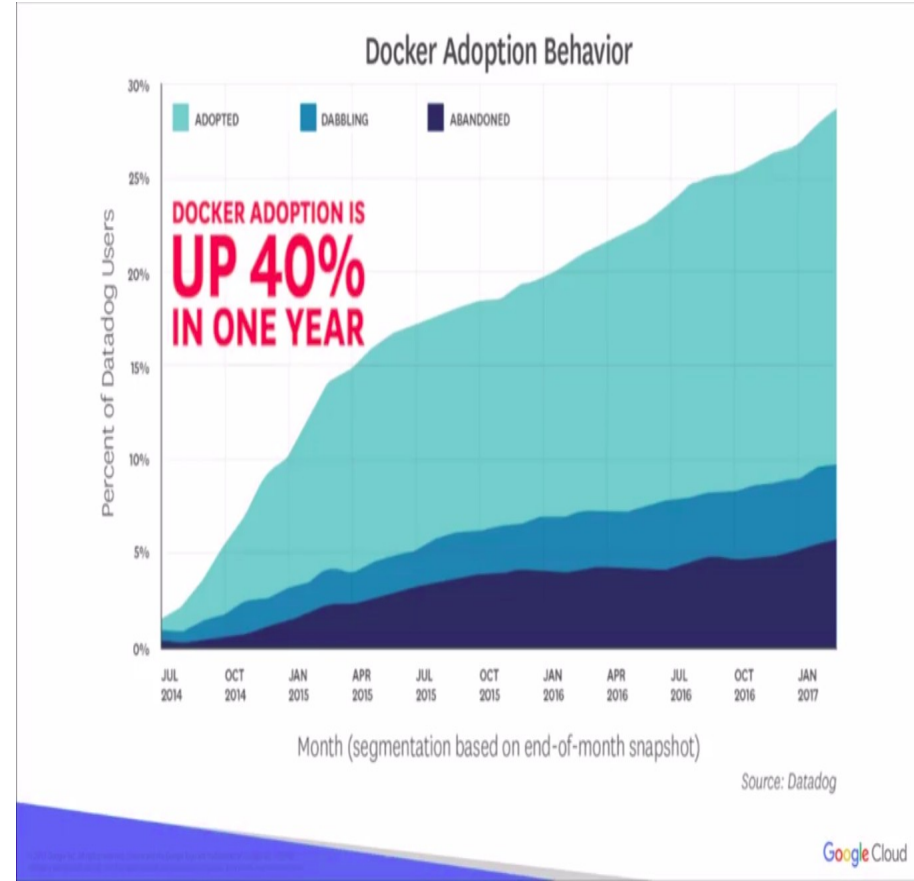
- Stackable: You can stack services vertically and on-the-fly.

Docker History

Released in around 2013. It's a massive adoption in the market today.

It's up to about 40% in one year Dockered adoption itself. Written in: Go

Developed by: Docker, Inc.



Images and containers

A container is launched by running an image. An image is an executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.

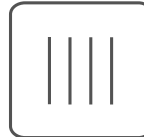
These portable images are defined by something called a Dockerfile

Dockerfiles

Docker file:

```
FROM alpine
RUN apk update && apk add nodejs
COPY . /app
WORKDIR /app
CMD ["node", "index.js"]
```

FROM alpine



alpine

+

RUN apk

apk update
apk add nodejs

+

COPY /app

host/index.js -->
container/app

+

WORKDIR /app
CMD ["node", "index.js"]

\$ cd /app
\$ node index.js

=



hello:v0.1

Docker Engine+Docker hub = Docker Platform

Docker Engine

Docker Daemon
Docker CLI

Docker Daemon

Builds Images
Runs and Manages Containers

Docker CLI

- `docker build` # Build an image from a Dockerfile
 - `docker images` # List all images on a Docker host
 - `docker run` # Run an image
 - `docker ps` # List all running and stopped instances
 - `docker stop` # Stop a running instances
 - `docker rm` # Remove an instance
 - `docker rmi` # Remove an image
- Docker

Docker Architecture

Docker Client

docker pull

docker run

docker ...



Host

Docker Daemon

Container 1

Container 2

Container 3

Container ...

Docker Hub

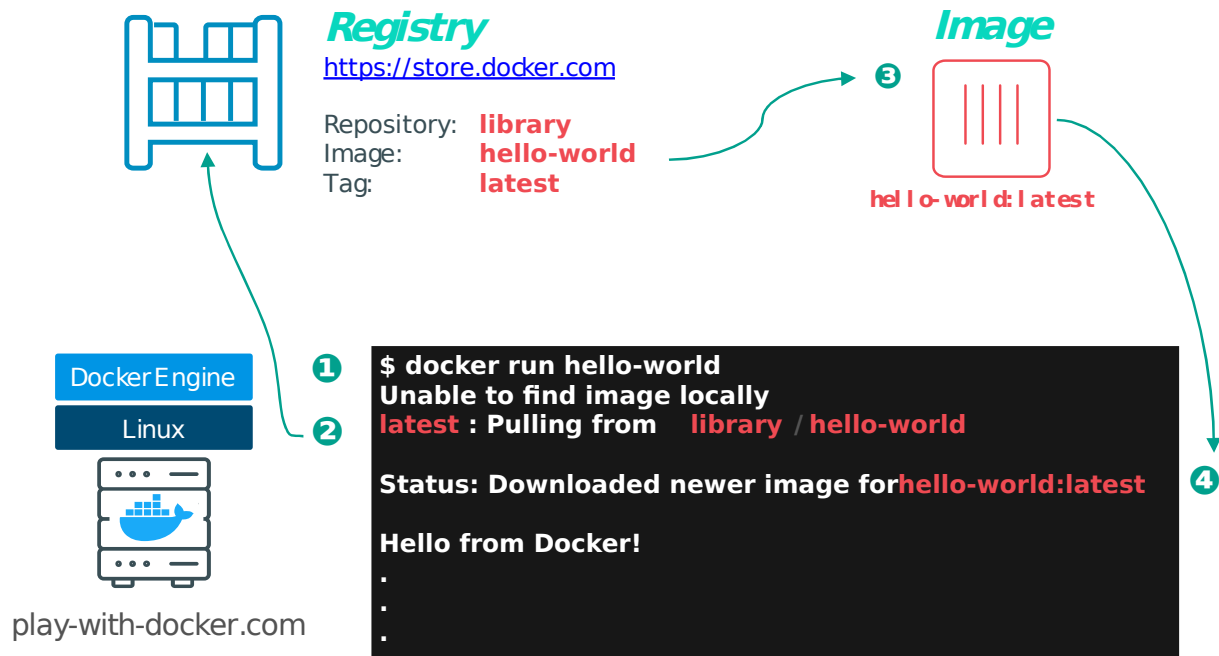
- Provides Docker Services
 - Library of public images
 - Storage for your images
 - free for public images
 - cost for private images
 - Automated builds(link github/bitbucket repo; trigger build on commit)
- Docker Hub

The screenshot displays the Docker Hub web interface. At the top, a blue navigation bar contains the Docker Hub logo, a search bar with the placeholder text "Search for great content (e.g., mysql)", and links for "Explore", "Repositories", "Organizations", "Get Help", and a user profile for "asimdogar". Below the navigation bar, the main content area shows a dropdown menu for the namespace "asimdogar" and a search bar "Search by repository name...". A "Create Repository +" button is located to the right. The repository list shows two entries: "asimdogar / myfirstapp" (Updated 7 days ago, 0 stars, 1 pull, PUBLIC) and "asimdogar / get-started" (Updated 11 days ago, 0 stars, 47 pulls, PUBLIC). A tip box at the bottom of the list suggests switching namespaces. The right sidebar features a "Create an Organization" section with the text "Manage Docker Hub repositories with your team" and an "Explore" section with the Docker logo.

Running Image in Docker

```
## Execute Docker image  
docker run hello-world
```


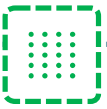
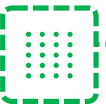

Hello World: What Happened?



Docker Container Instances

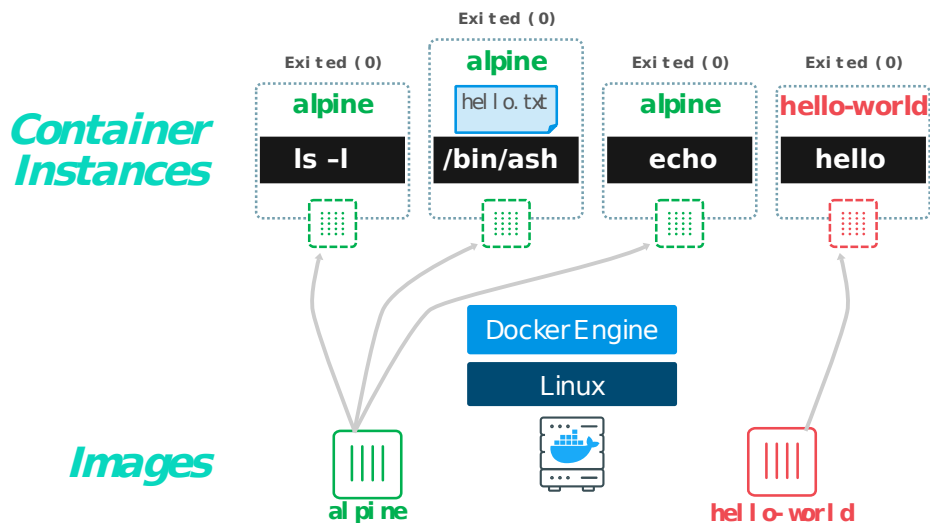
Output of `docker container ls -a`

*Container
Instances*

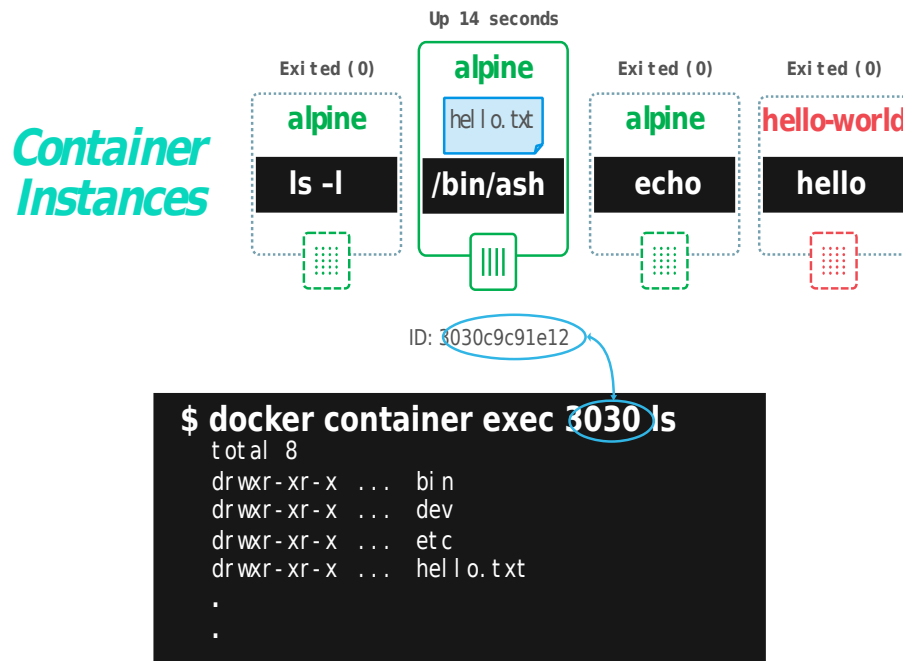
	Exited (0)	Exited (0)	Exited (0)	Exited (0)
				
<i>Container IDs</i>	ff0a5c3750b9	36171a5da744	a6a9d46d0b2f	c317d0a9e3d2
<i>Container Names</i>	elated_ramanujan	fervent_newton	lonely_kilby	stupefied_mcclintock

Container Isolation

Docker Container Isolation



docker container exec



Container Orchestration

Container Orchestration Software
(Docker, Openshift & Kubernetes)

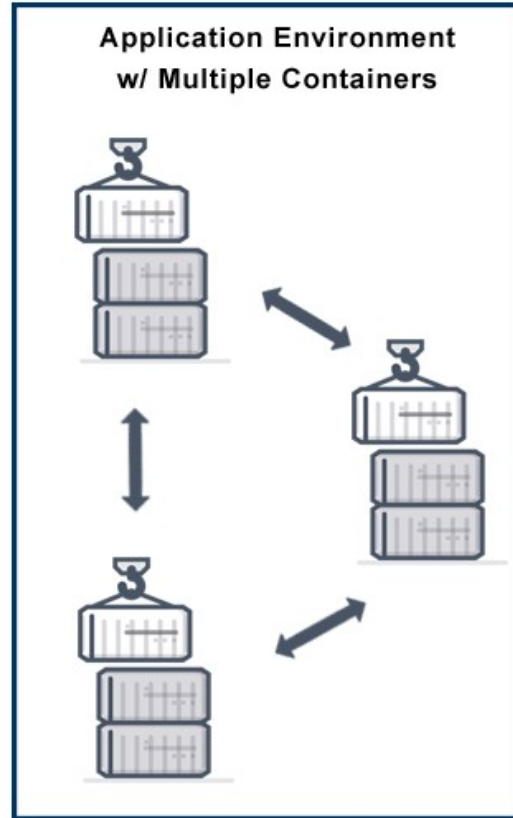


OPENSIFT



Automate:

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



Kubernetes (K8s)

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.

It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation. Initial release: 7 June 2014; Written in: Go



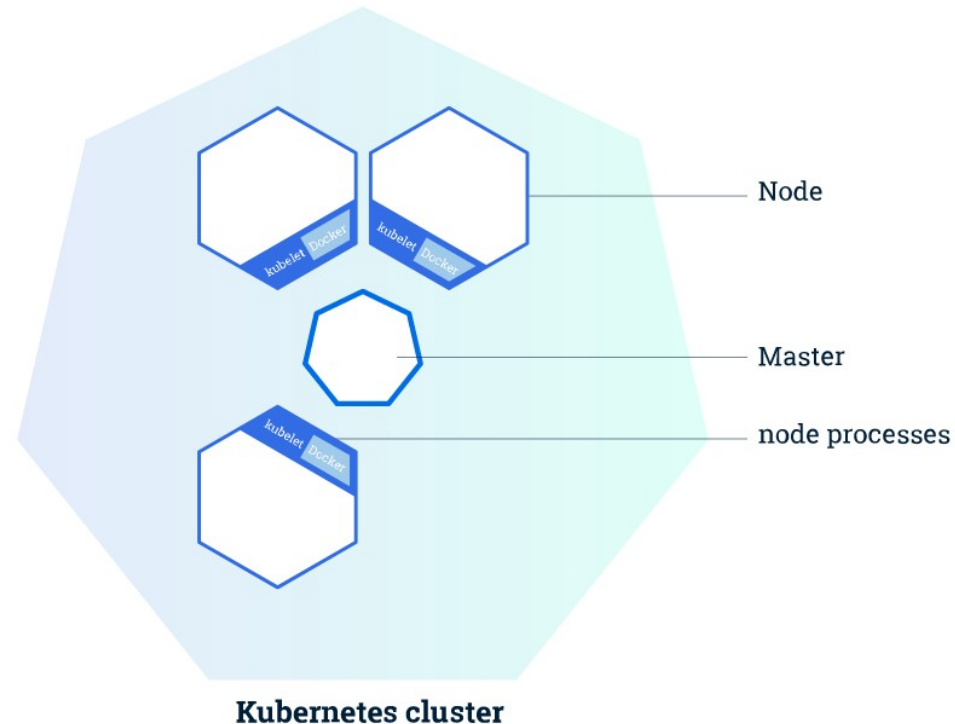
Kubernetes Clusters

A Kubernetes cluster consists of two types of resources:

The Master coordinates the cluster

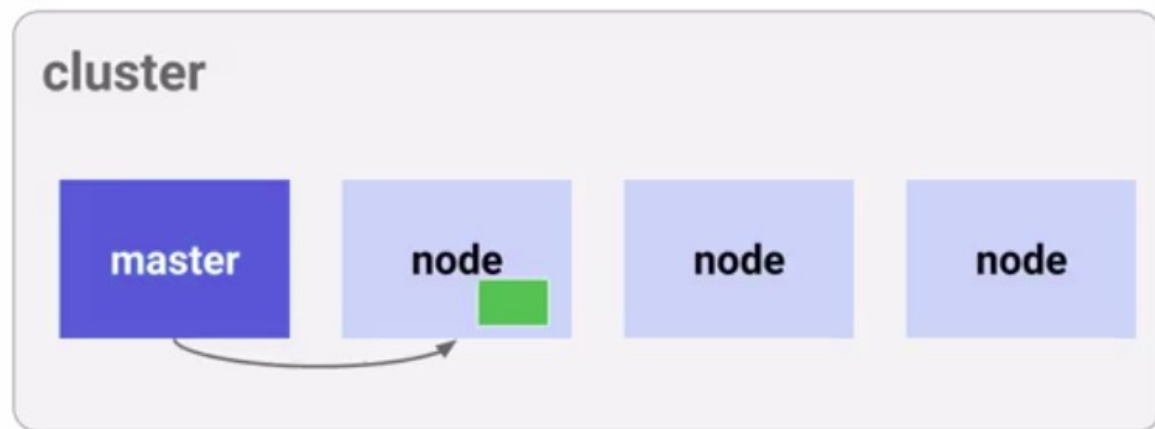
Nodes are the workers that run applications

The nodes communicate with the master using the Kubernetes API, which the master exposes. End users can also use the Kubernetes API directly to interact with the cluster.



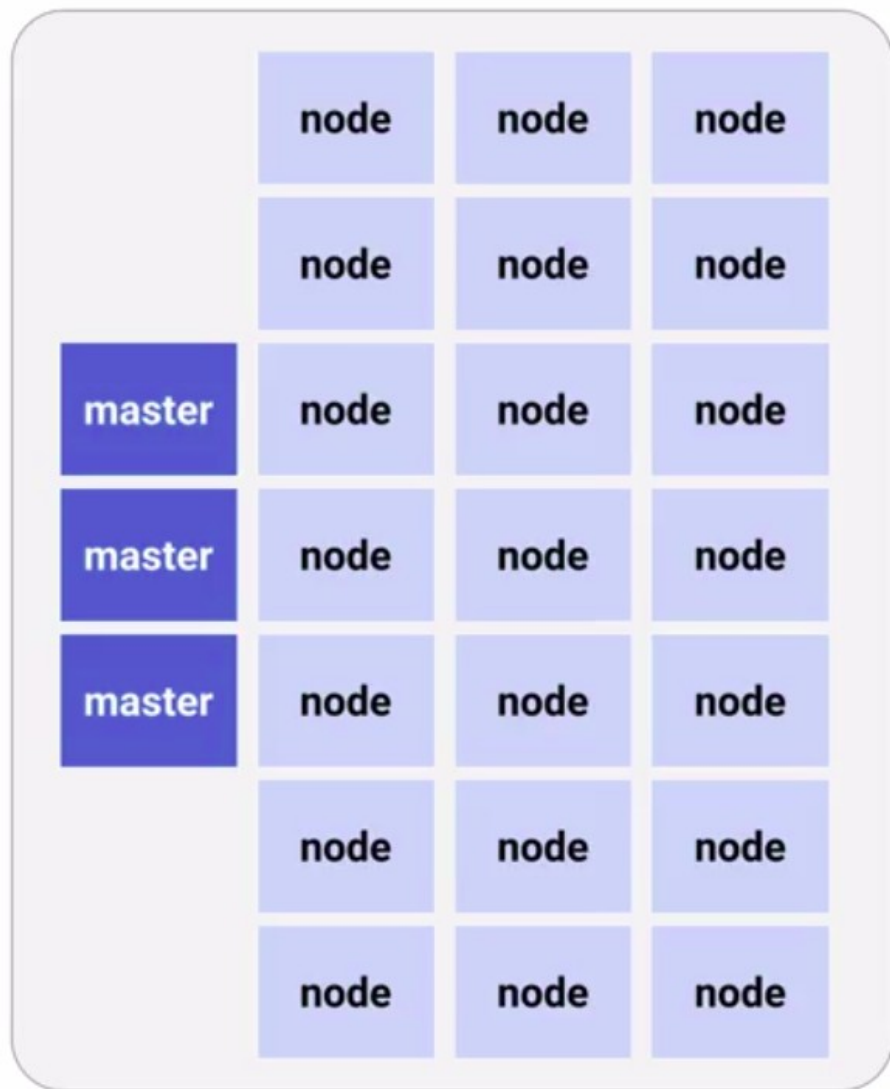
Kubernetes manages jobs

Jobs run containers on nodes



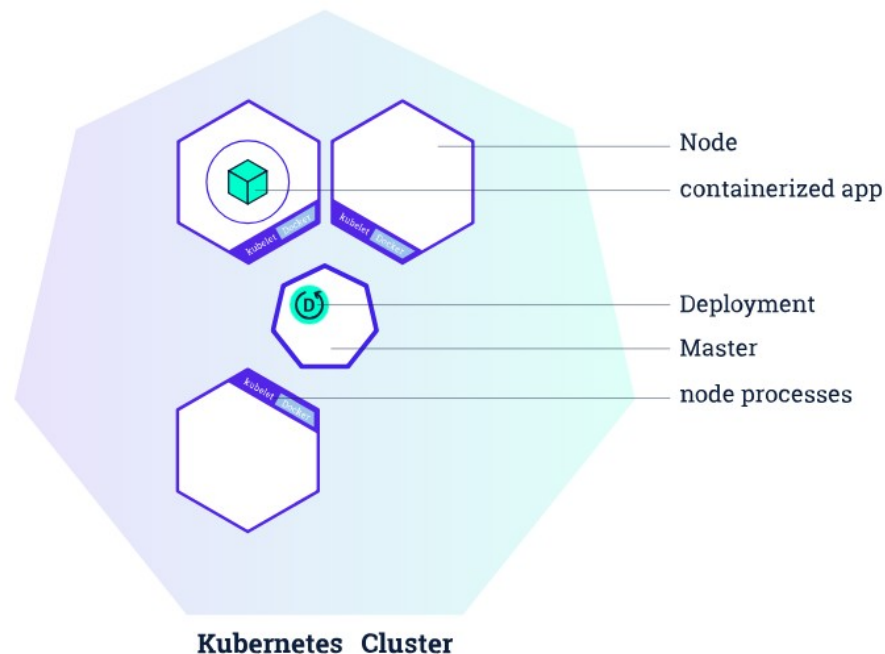
In a real ecosystem, a cluster may have 1000s of nodes and multiple masters.

Regional clusters have masters and nodes spread across 3 zones



Deploying your first app on Kubernetes

You can create and manage a Deployment by using the Kubernetes command line interface, Kubectl. Kubectl uses the Kubernetes API to interact with the cluster.



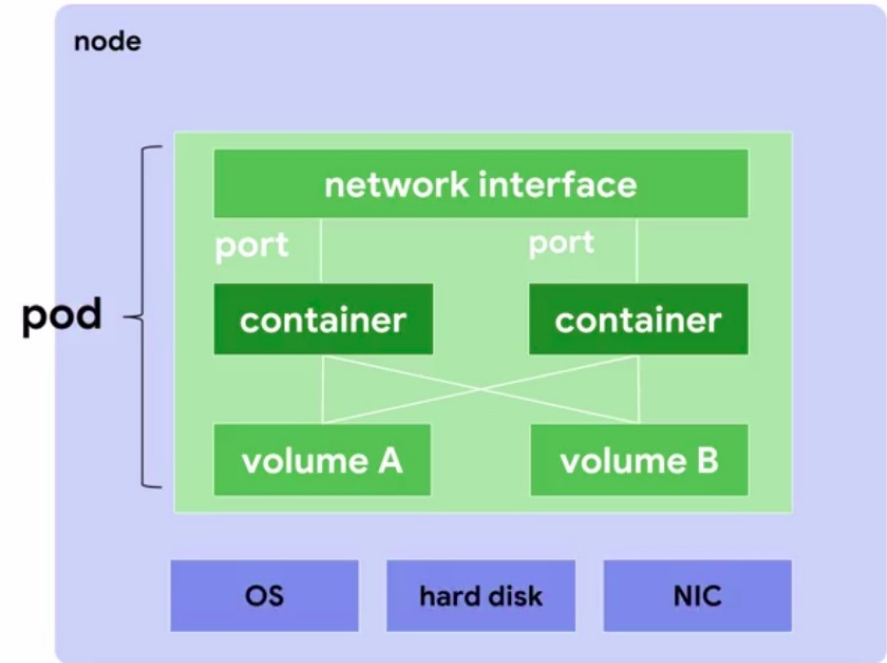
Kubernetes Pods

Shared storage, as Volumes

Networking, as a unique cluster IP address

Information about how to run each container, such as the container image version or specific ports to use

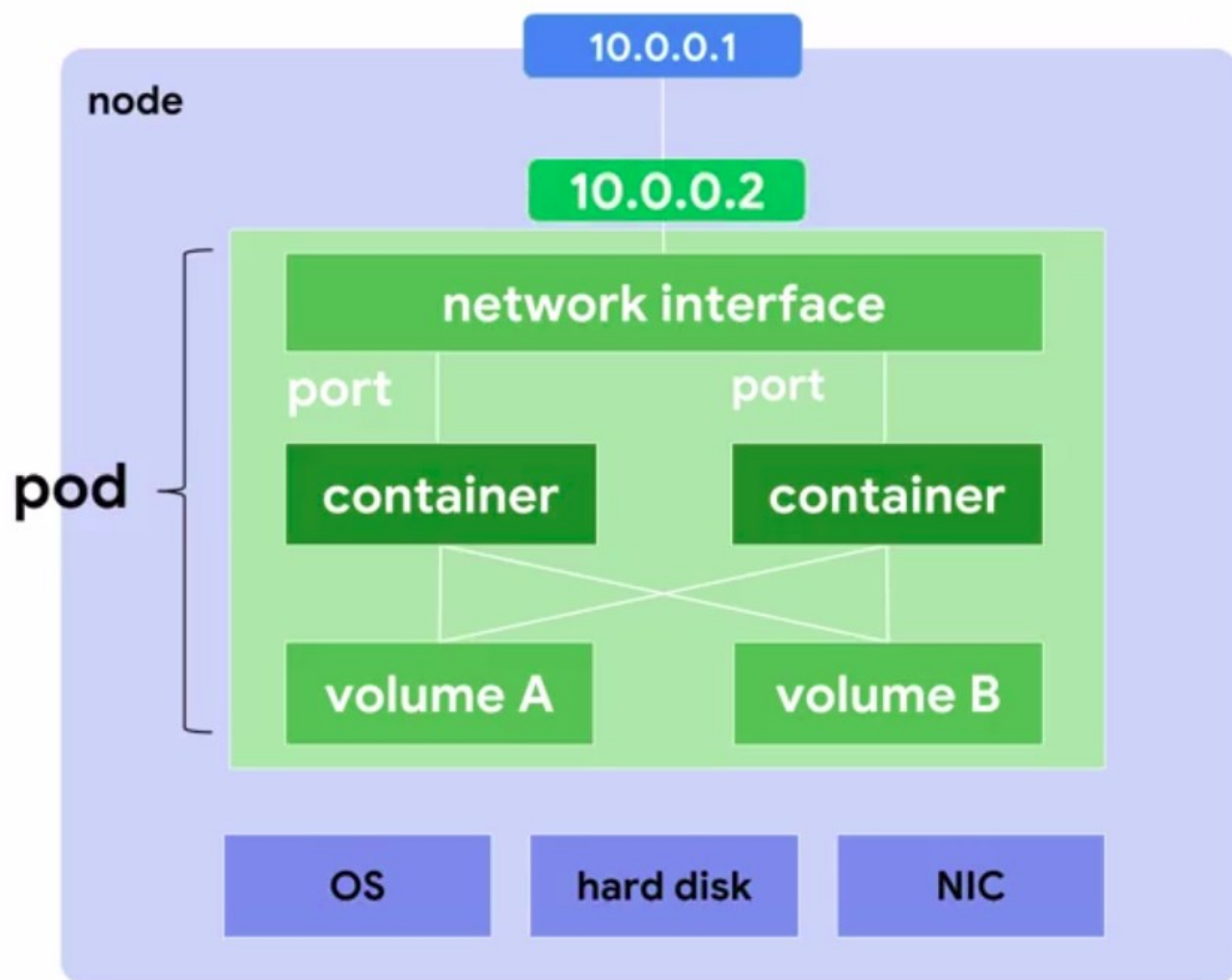
Underneath the pod you have the node's hardware, OS, and NIC



Inside you have the pod with its containers

It gets its own network namespace with unique IP and set of ports

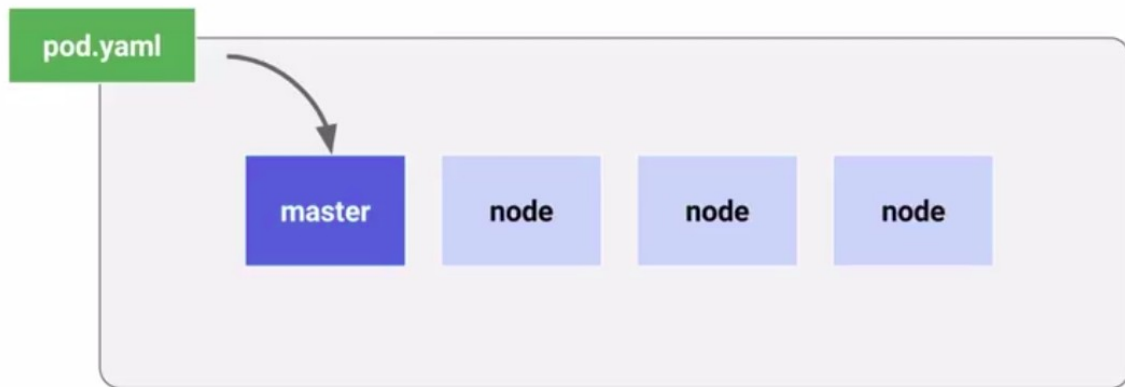
Data is stored in **volumes** in memory or persistent disks



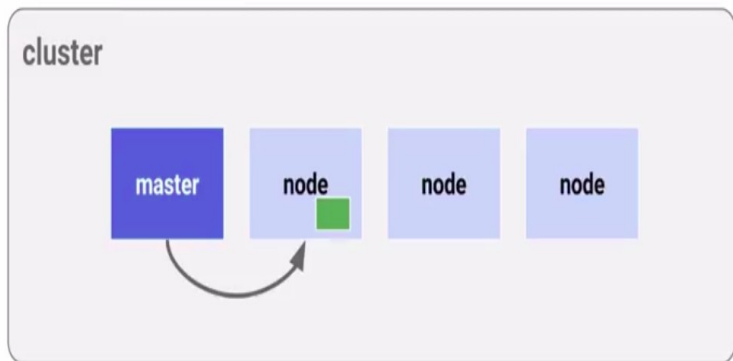
You define a pod with a YAML file

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
    - name: my-app
      image: my-app
    - name: nginx-ssl
      image: nginx
  ports:
    - containerPort: 80
    - containerPort: 443
```

You upload the YAML file to the master



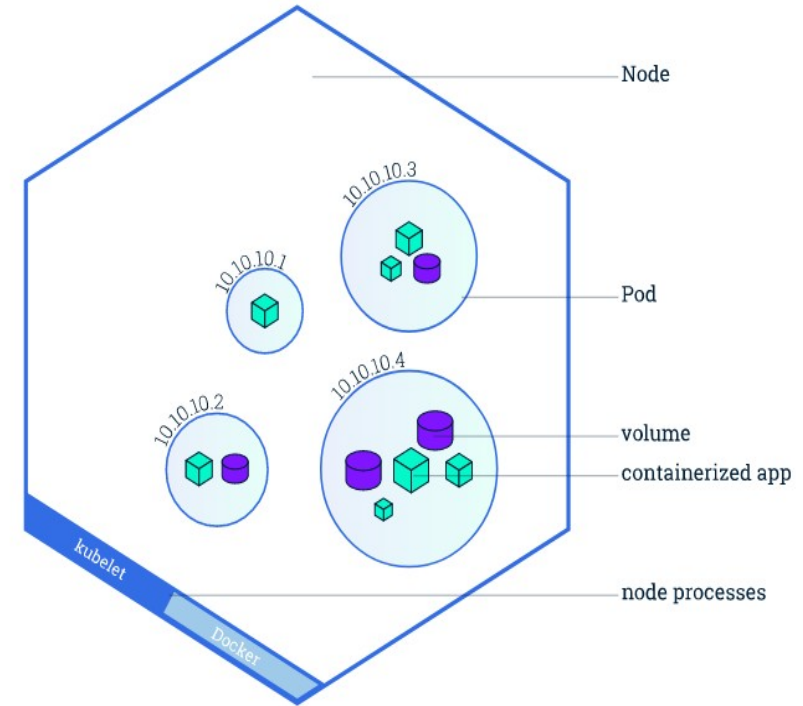
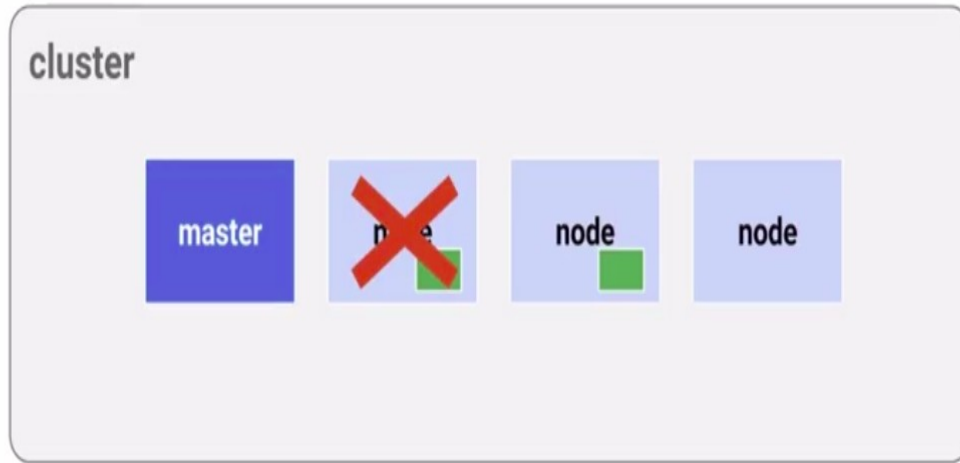
And the master creates a pod on your set of nodes



A pod file is composed of several parts; for example...

```
apiVersion: v1      — API version
kind: Pod           — pod resource
metadata:
  name: my-app      — pod name
spec:
  containers:       — two containers
  - name: my-app
    image: my-app
  - name: nginx-ssl
    image: nginx
  ports:            — NGINX front end on two ports
  - containerPort: 80
  - containerPort: 443
```

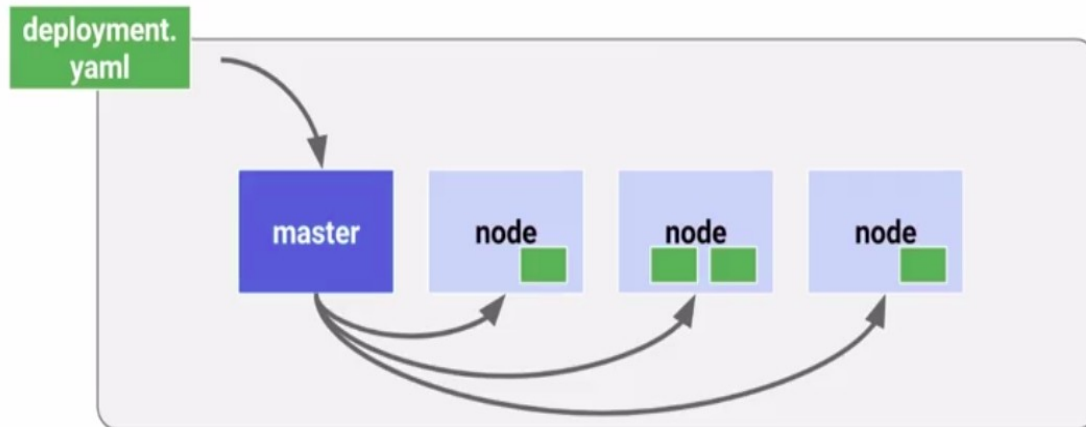
A deployment ensures that N pods are running in a cluster at any given time



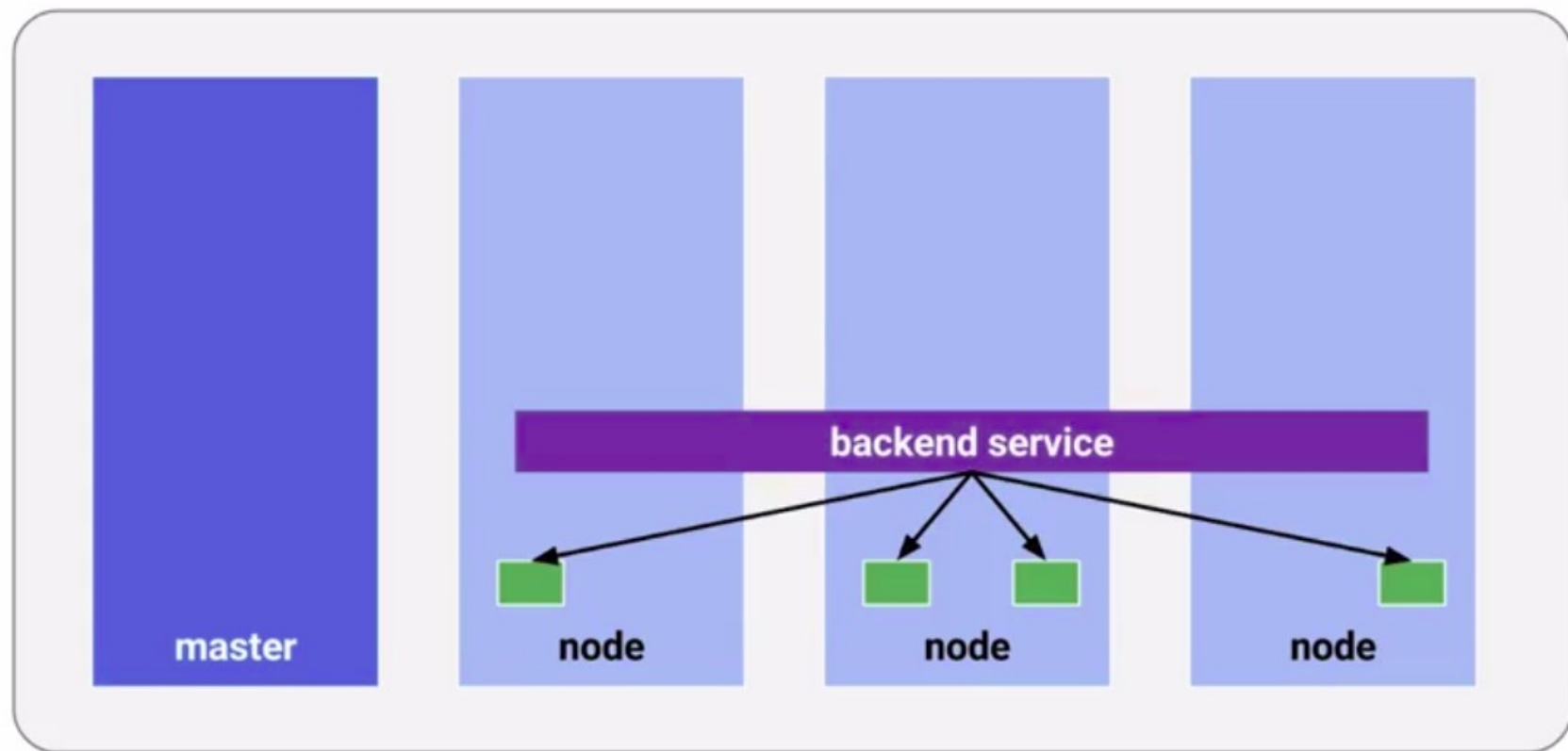
```
kind: Deployment — deployment resource
apiVersion: v1.1
metadata:
  name: frontend — deployment name
spec:
  replicas: 4 — replicas
  selector: — pod selector
    role: web — role=web
  template:
    metadata:
      name: web
      labels: — pod label
        role: web — role=web
    spec:
      containers: — pods
      - name: my-app
        image: my-app
      - name: nginx-ssl
        image: nginx
        ports:
          - containerPort: 80
          - containerPort: 443
```

You define a deployment with a YAML file

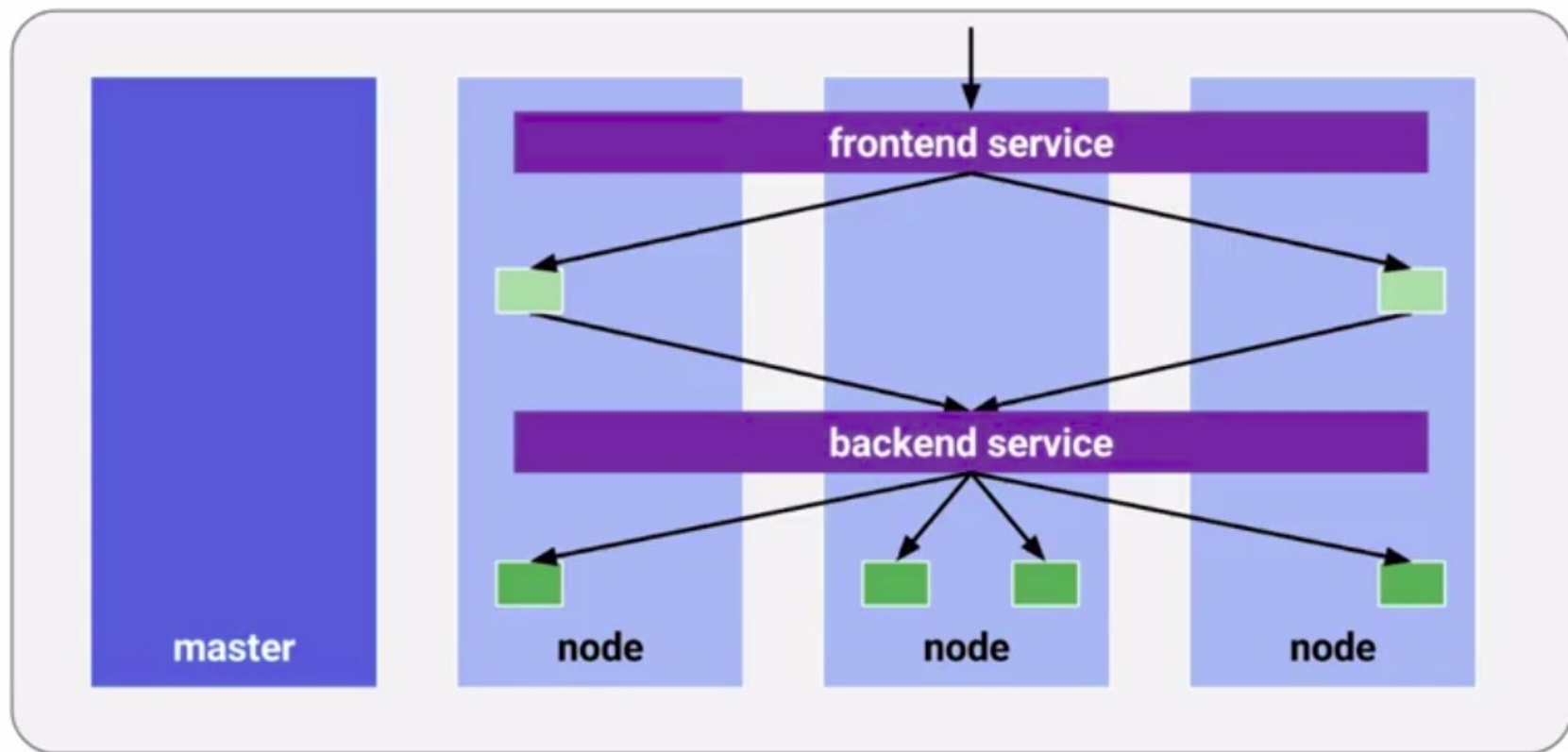
You upload the YAML file to the master, and the scheduler decides where to run the pods



A service assigns a fixed IP to your pod replicas and allows other pods or services to communicate with them



You can have multiple services with different configurations and features



You define a service with a YAML file

```
kind: Service — resource
apiVersion: v1
metadata:
  name: web-frontend
spec:
  ports: — ports
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
  selector: — pod selector
    role: web
  type: LoadBalancer — type is load balancer
```

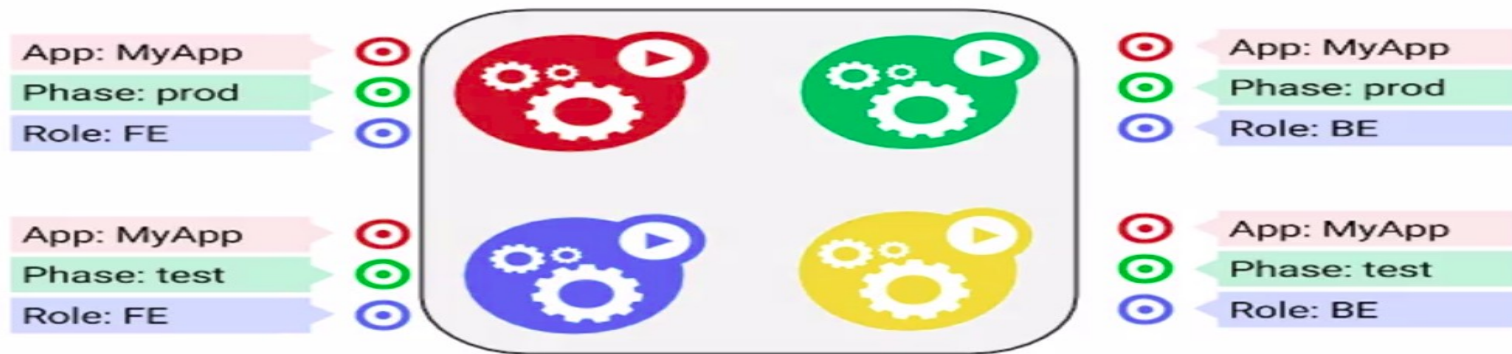
Labels are metadata you can assign to any API object and represent identity

They are

- The only grouping mechanism for pods
- Search by selectors



You can query for labels that map to a value like the entire app



App = MyApp

Or narrow your search with multiple labels like your app's frontend



App = MyApp, Role = FE

Or your app's backend

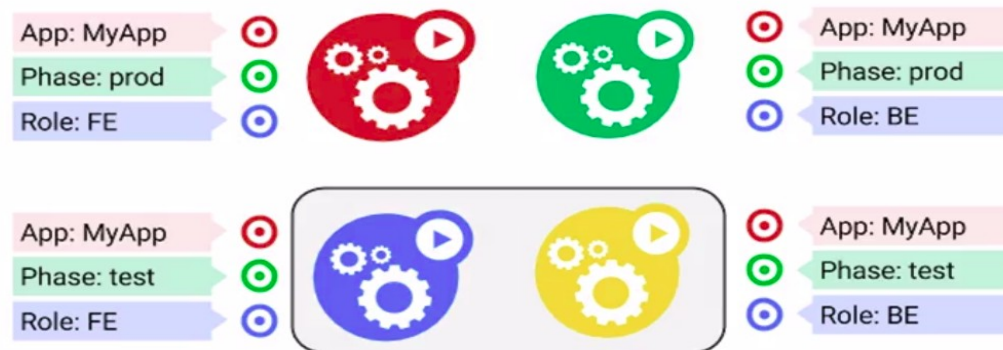


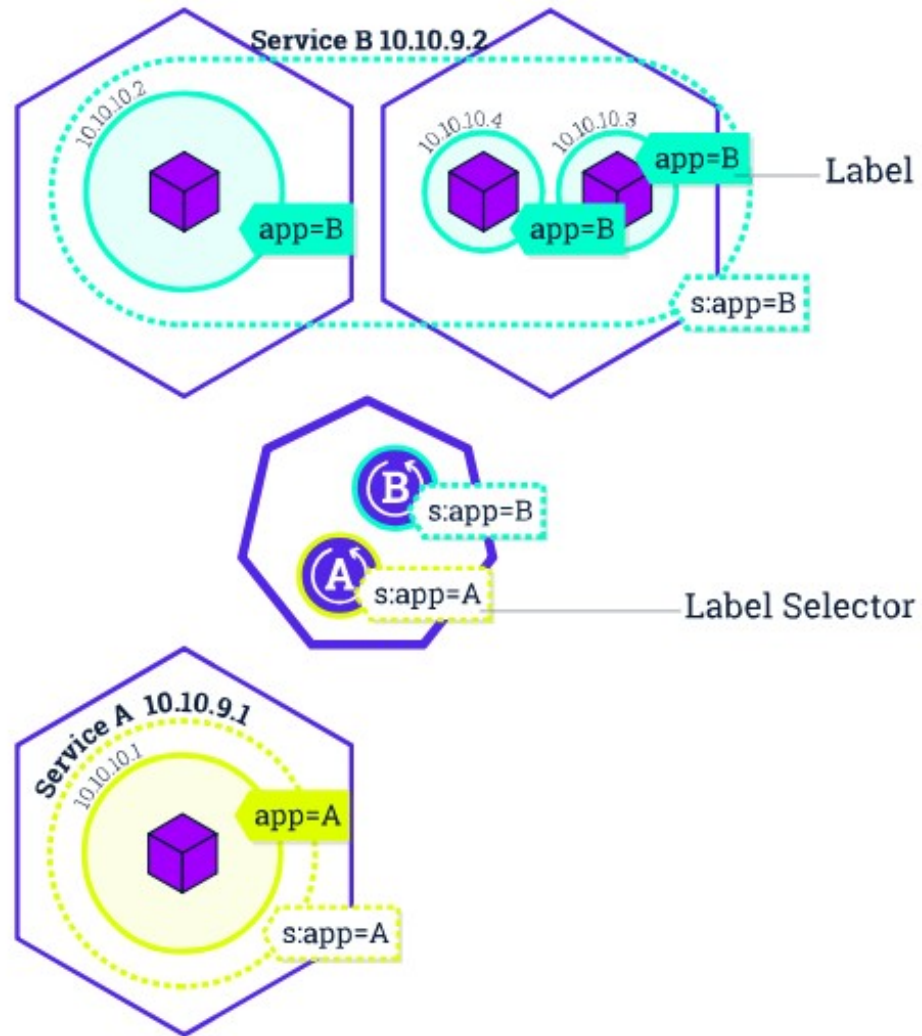
App = MyApp, Role = BE

Or your app's production release

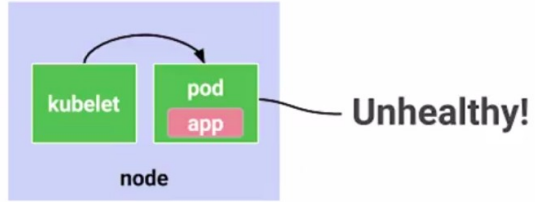


Or your app's test phase



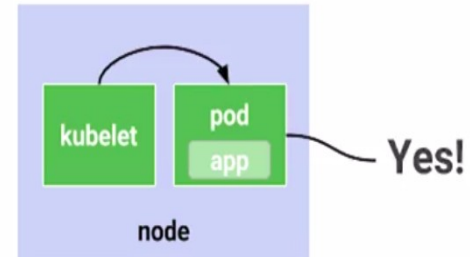
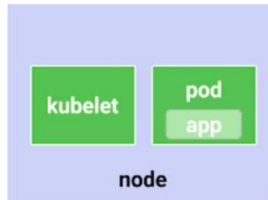


Kubelet checks whether the pod is alive and healthy; if it gets a negative response or no reply...

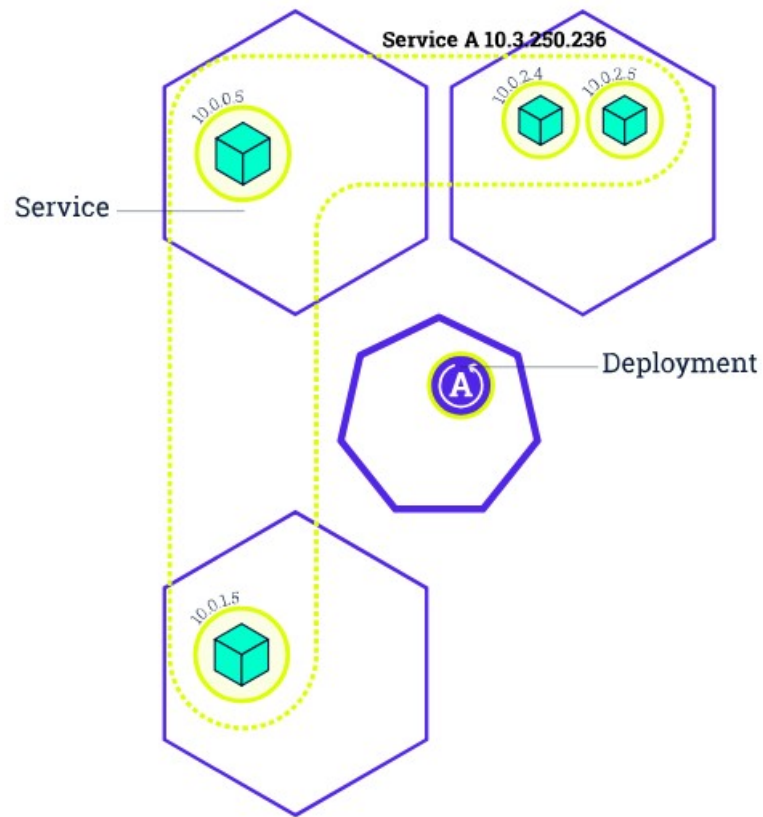
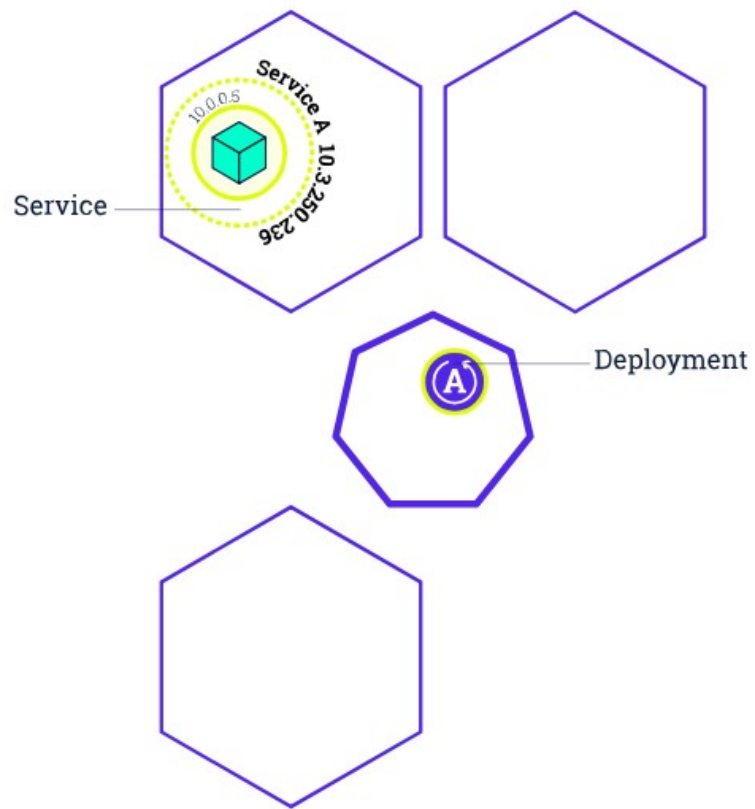


And continues until it gets a healthy reply

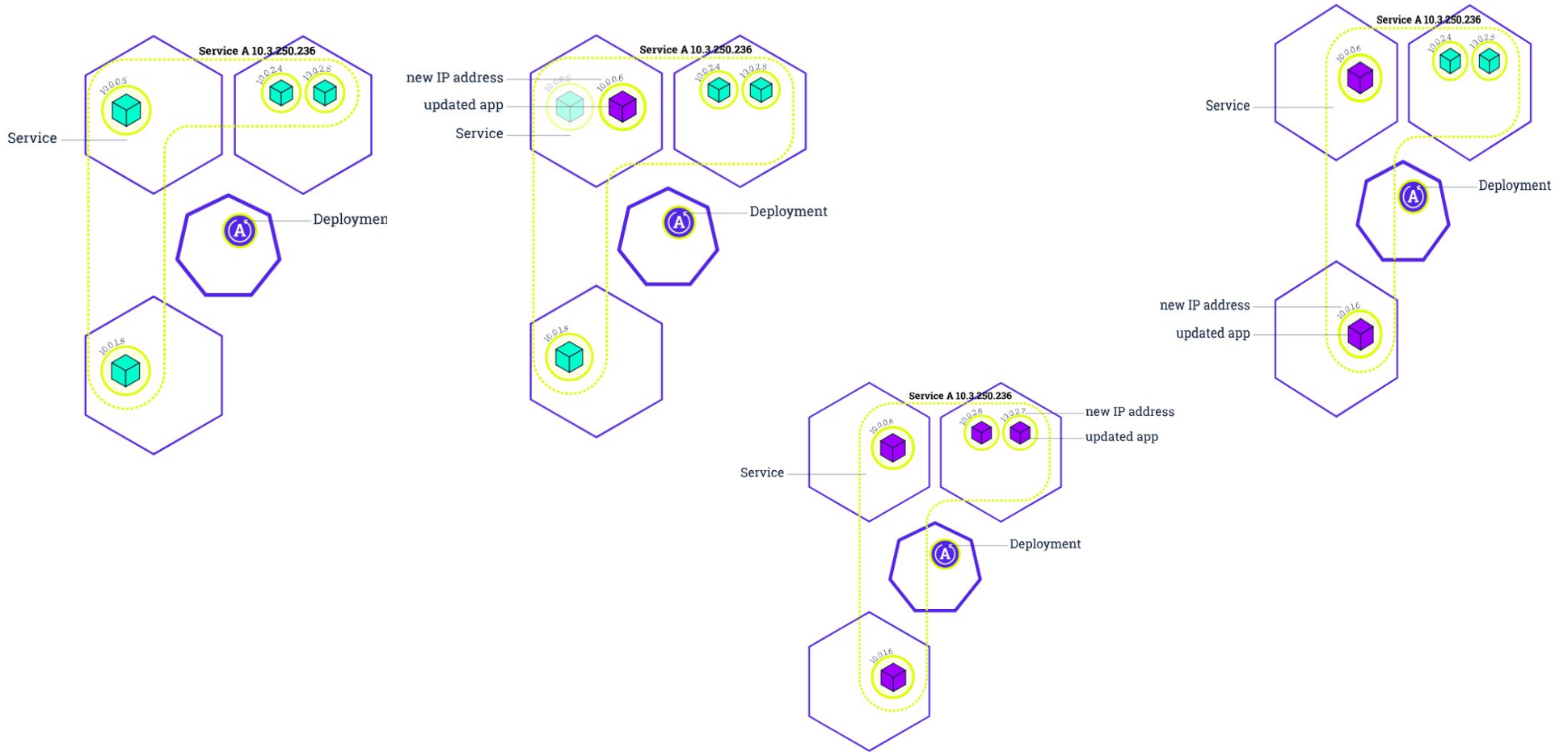
Kubelet restarts the pod



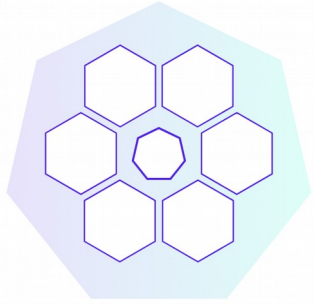
Scaling overview



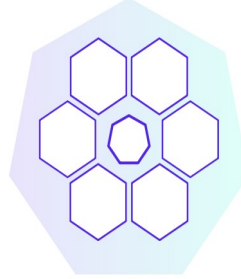
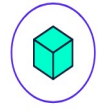
Rolling updates overview



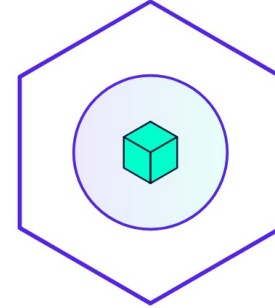
Kubernetes Basics Modules



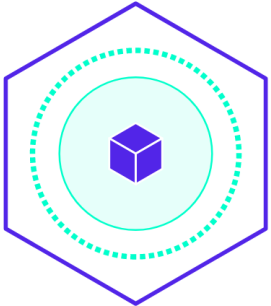
1. Create Cluster



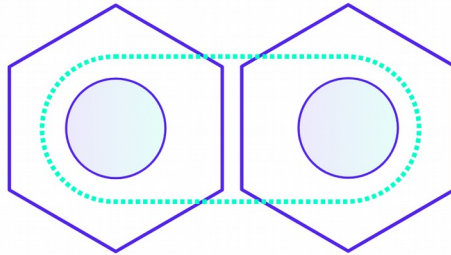
2. Deploy App



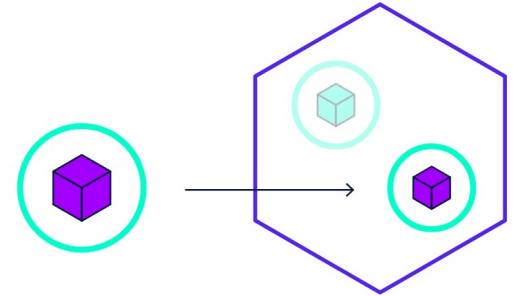
3. Explore App



4. Expose App publicly



5. Scale up App



6. update App

References

- <https://www.coursera.org/learn/google-kubernetes-engine>
- <https://kubernetes.io/docs/tutorials/kubernetes-basics/>
- <https://www.docker.com/get-started>