DISTRIBUTED AND CLOUD COMPUTING

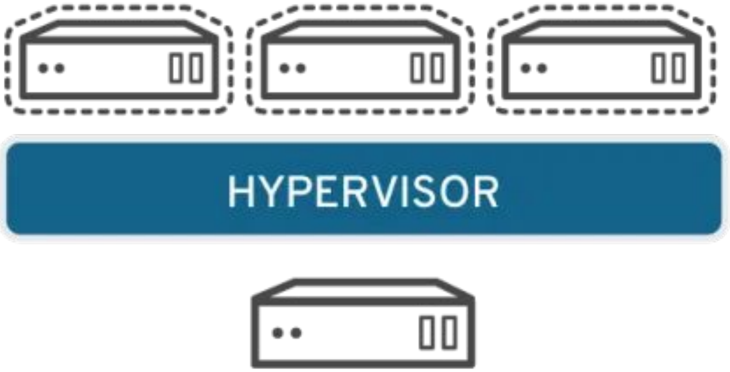
LAB 12: K8S COMPONENTS & RESOURCES

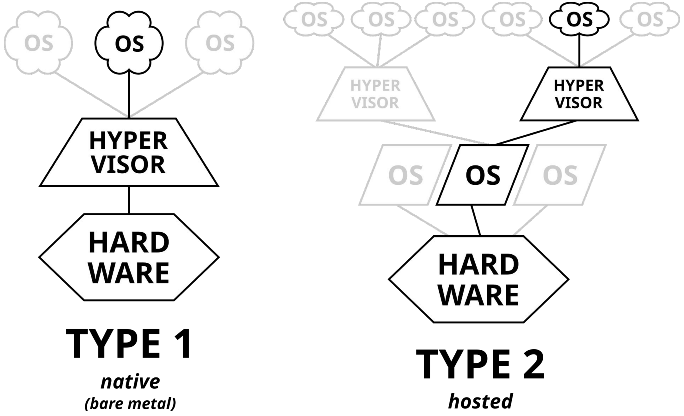
(Module: K8S & CLOUD BASICS)

https://aws.amazon.com/what-is/virtualization/
https://www.redhat.com/en/topics/virtualization/what-is-virtualization
https://en.wikipedia.org/wiki/Hypervisor

Virtualization: Enabling Cloud Computing

- **Virtualization**:
  a. creates virtual representations of hardware
  b. powers cloud computing services by helping organizations manage infrastructure
- **Why?**
  Isolation
  a. More flexible & secure resource utilisation.
  b. Painless environment setup & recovery effort (keep everything tidy).
- **Virtual Machine Monitor (VMM, or Hypervisor)**: a type of computer software, or hardware that creates and runs **virtual machines (VMs)**.





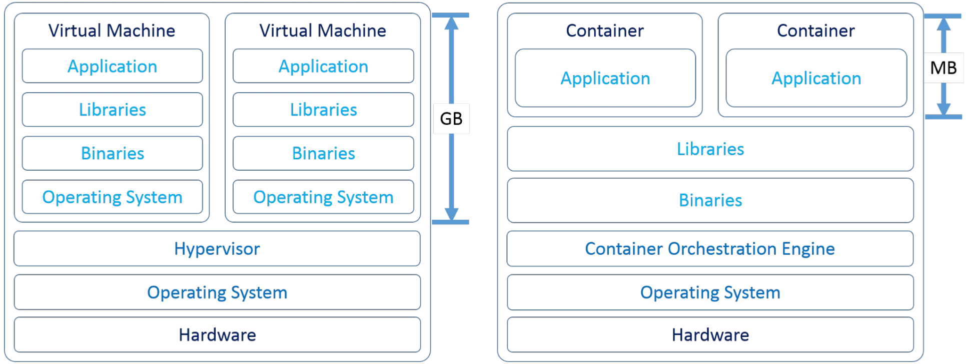
https://cloud.google.com/learn/what-are-containers
https://www.alibabacloud.com/en/knowledge/difference-between-container-and-virtual-machine
https://dbconvert.com/blog/content/images/size/w1600/2021/02/docker-apps.png

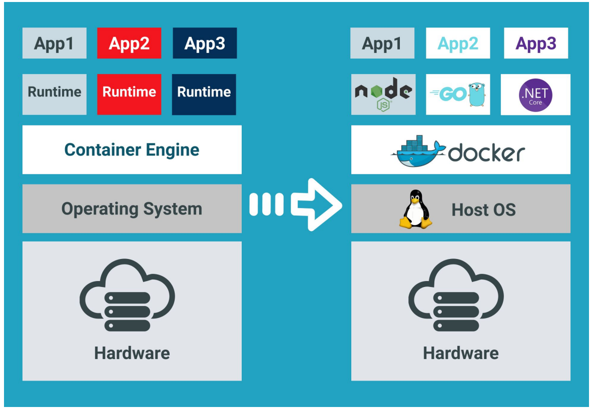
Containerization

- **Containers**: lightweight packages of application code together with dependencies required to run your software services.
- **Containers - compared to VMs**:
  a. **Efficient**:
     - containerized apps share the hosting OS kernel, thus using fewer resources.
     - even easier environment setup.

  b. **Lightweight**: do not carry OS; only contain necessary libraries & tools.

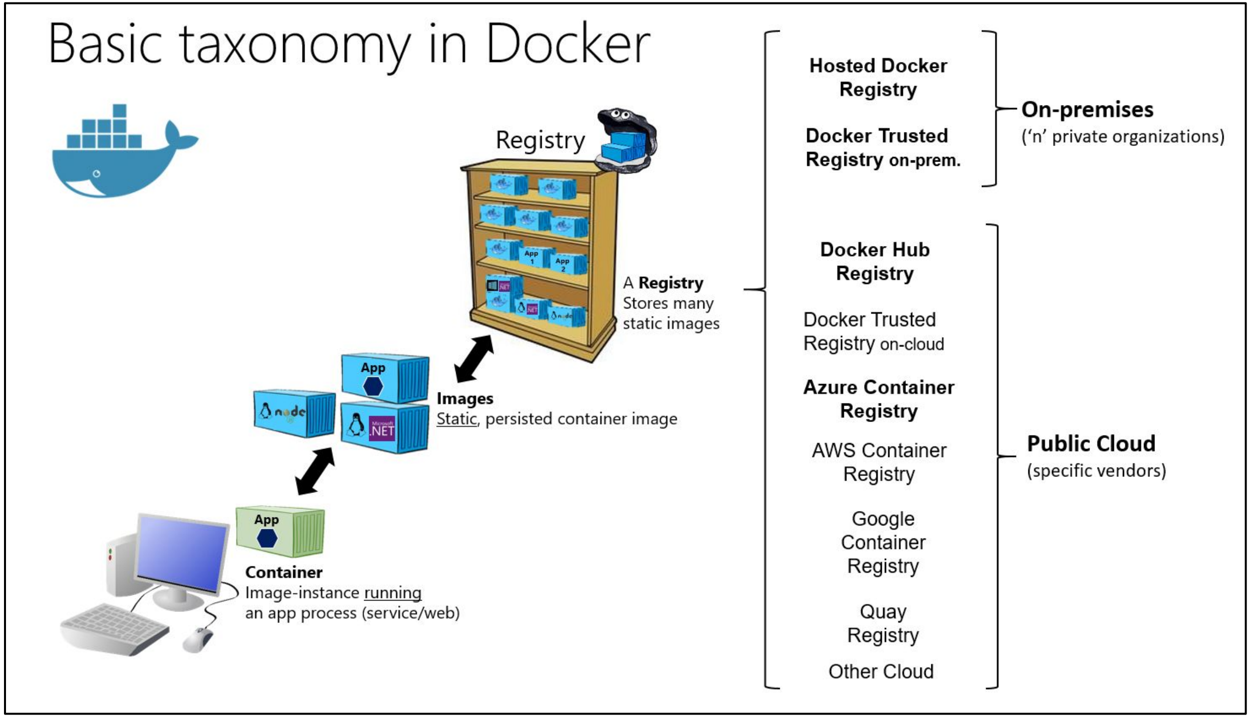
  c. **Portability**: OS-independent, thus movable to PCs/VMs/Cloud.





Docker: Basic Concepts

- **Registry**
  - Docker Hub
- **Image**
  - OS like ubuntu
  - apps like postgres
- **Container**
  - image instance



https://learn.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-containers-images-registries

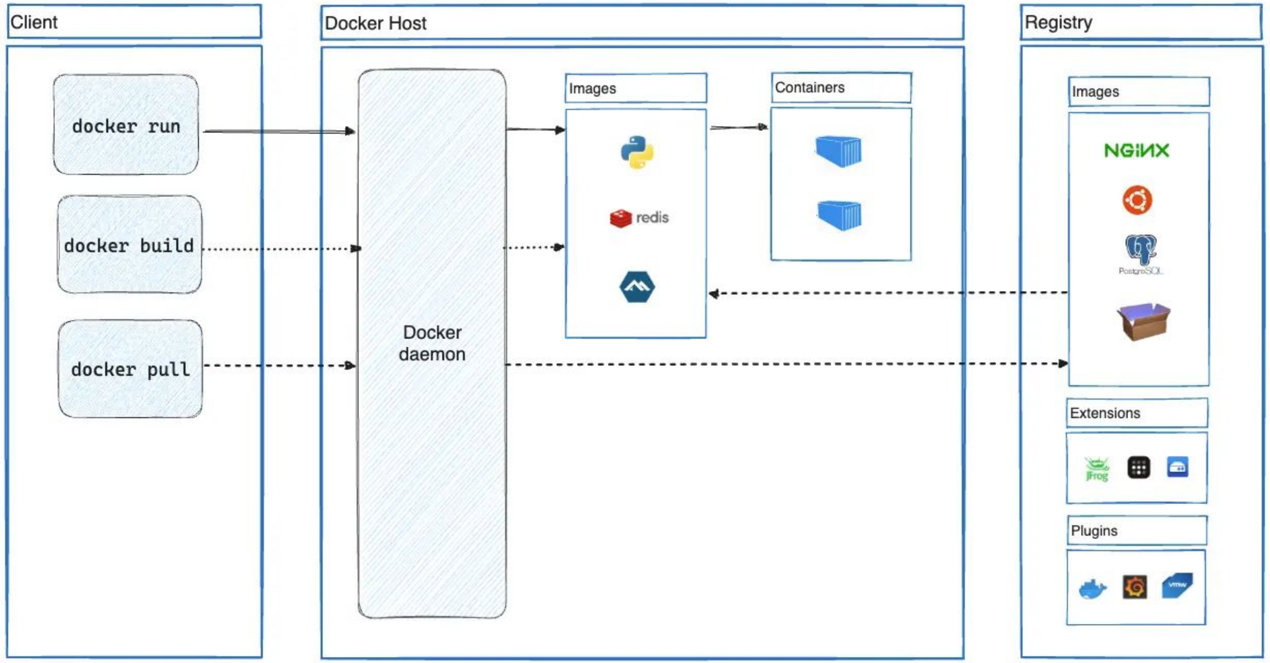
ubuntu image: https://hub.docker.com/_/ubuntu

postgres image: https://hub.docker.com/_/postgres

Docker: Architecture

- **Registry**
  - pull images
  - `docker pull`
- **Image**
  - build images
  - requires [`Dockerfile`]
  - `docker build`
- **Container**
  - run images
  - `docker run`

- **Docker CLI/Client**
  - `docker` commands
- **Docker daemon**
  - middleman between Docker Client & Docker objects
- **Docker Desktop**
  - includes the above

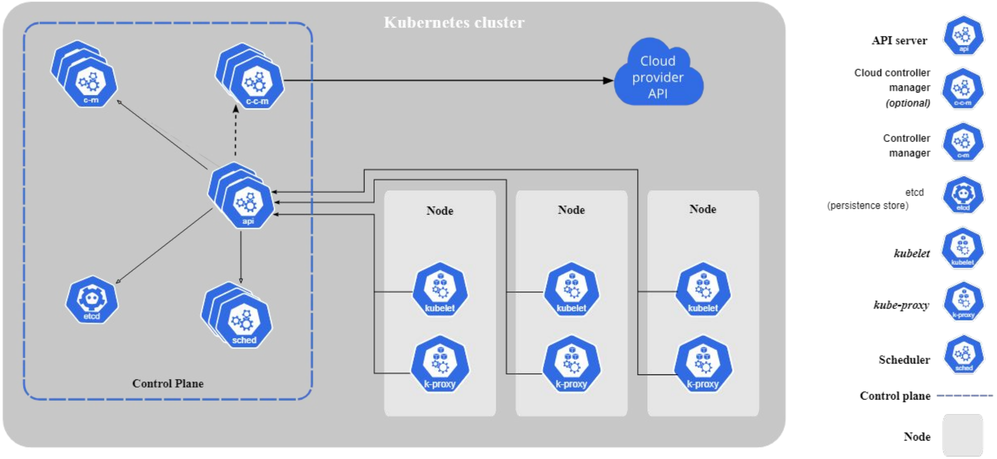


https://docs.docker.com/get-started/docker-overview/#docker-architecture

Docker Compose, Swarm & Kubernetes

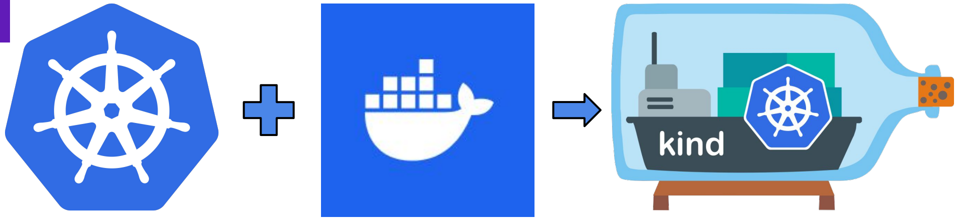
- **Docker Compose**: a tool for defining and running multi-container applications on a single host.
  - Requires compose.yaml.
- **Docker - Swarm mode**: an advanced feature for managing a cluster of Docker daemons - running multi-container applications on multiple hosts.
- **Kubernetes (K8s)**: a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

WE ARE HERE!



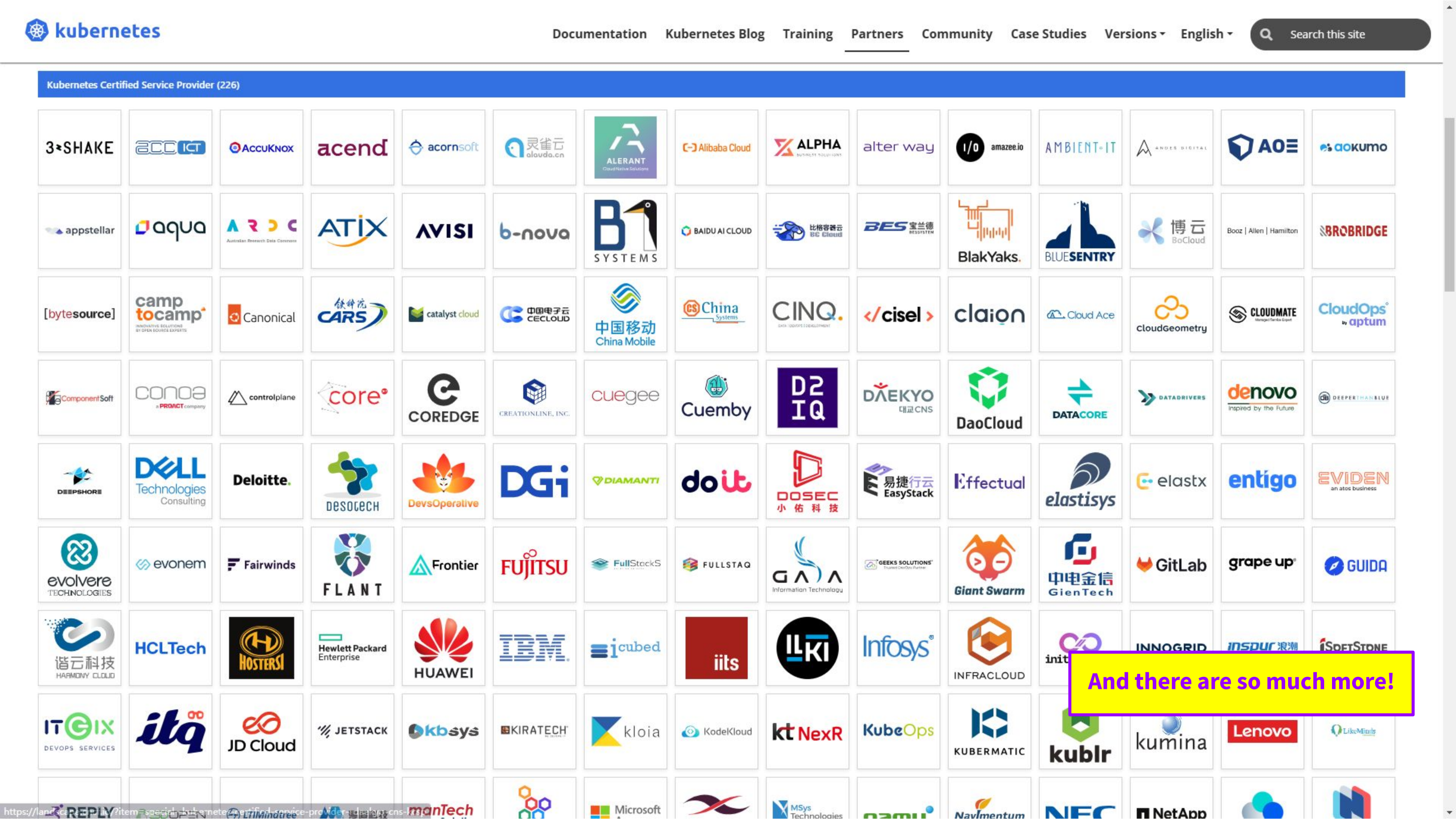
https://docs.docker.com/compose/
https://docs.docker.com/engine/swarm/
https://kubernetes.io/docs/concepts/

Kubernetes (K8s)



- "K8s is an open-source container orchestration system for automating software deployment, scaling, and management. The name originates from Greek κυβερνήτης (kubernḗtēs), meaning governor, helmsman, or pilot."
- Primarily written in Go and originated from Google Borg.
- Features:
  a. multi-machine container orchestration
     - horizontal scaling
     - self-healing
     - automated rollouts & rollbacks
     - automatic bin packing
  b. service discovery & load balancing
  c. …
- K8s is currently the most popular software deployment solution for containerized applications, especially in the context of cloud-native application architectures.

https://github.com/docker
https://kind.sigs.k8s.io/
https://en.wikipedia.org/wiki/Kubernetes
https://en.wikipedia.org/wiki/Helmsman
https://github.com/kubernetes/kubernetes

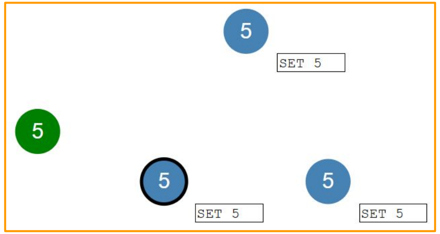


Kubernetes Certified Service Provider (226)

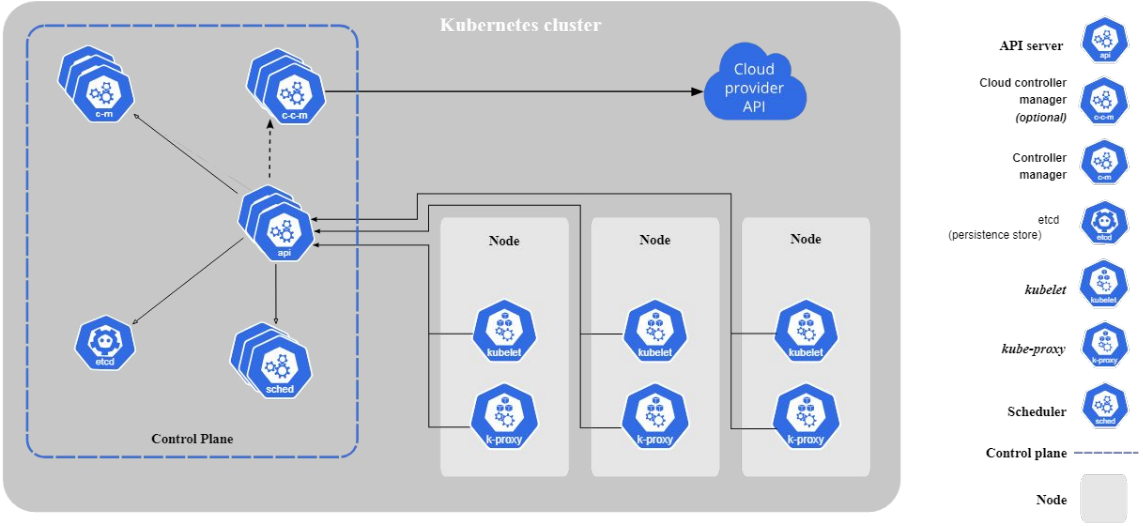
And there are so much more!

https://kubernetes.io/docs/concepts/overview/components/

Kubernetes Components



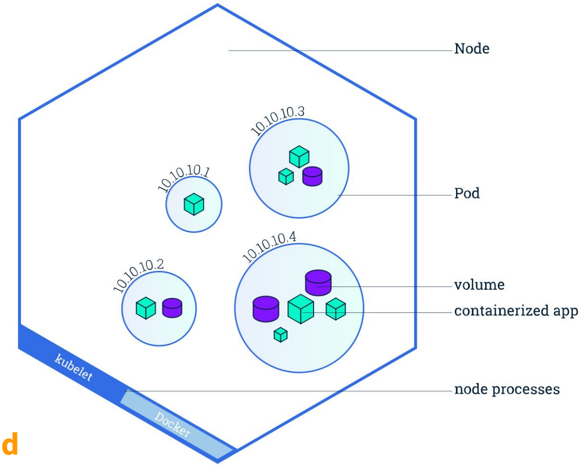
- Control Plane
  a. etcd (distributed KV store following the Raft consensus protocol)
  b. API Server: exposes K8s API to clients like ` kubectl
  c. Controller Manager: control loops that monitor system states & match desired states
  d. Cloud Controller Manager (*): cloud-specific control logic
  e. Scheduler: performs pod scheduling to cluster nodes
- Worker Node
  a. kubelet
     - node agent
     - “local official”
  b. kube-proxy (*)
     - network manager
  c. Container Runtime
     - Containerd
     - CRI-O
     - . . .

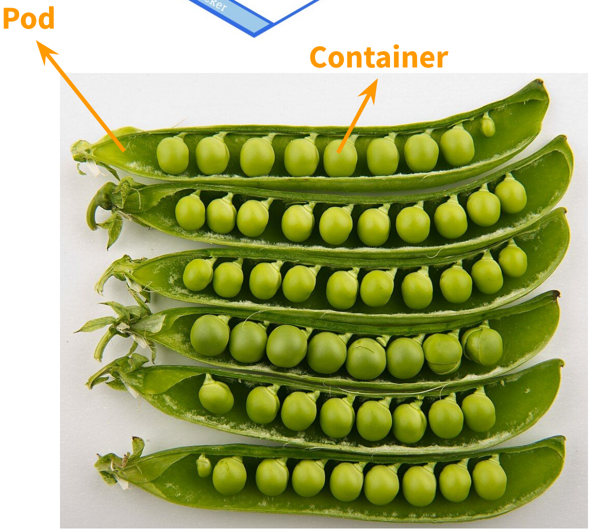


Docker uses Containerd!

Kubernetes Resources/Objects

- Pod: smallest deployable unit - group of containers
  a. Containers from the same pod are tightly coupled.
- ReplicaSet & Deployment
  a. Deployment owns ReplicaSets to manage sets of pod replicas, in order to execute **stateless** workloads.
- StatefulSet: maintains a group of pods to support stateful applications with persistent storage & network identities.
- Job: one-off tasks that run to completion and then stop.
- Service: expose applications behind a single endpoint
- AutoScalers
  a. Pod AutoScaler:
     ■ Horizontal (scale in/out)
     ■ Vertical (scale up/down)
  b. Cluster AutoScaler: autoscales cluster nodes
- ...





https://en.wikipedia.org/wiki/Pea
https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/

Kubernetes CLI - `kubectl`

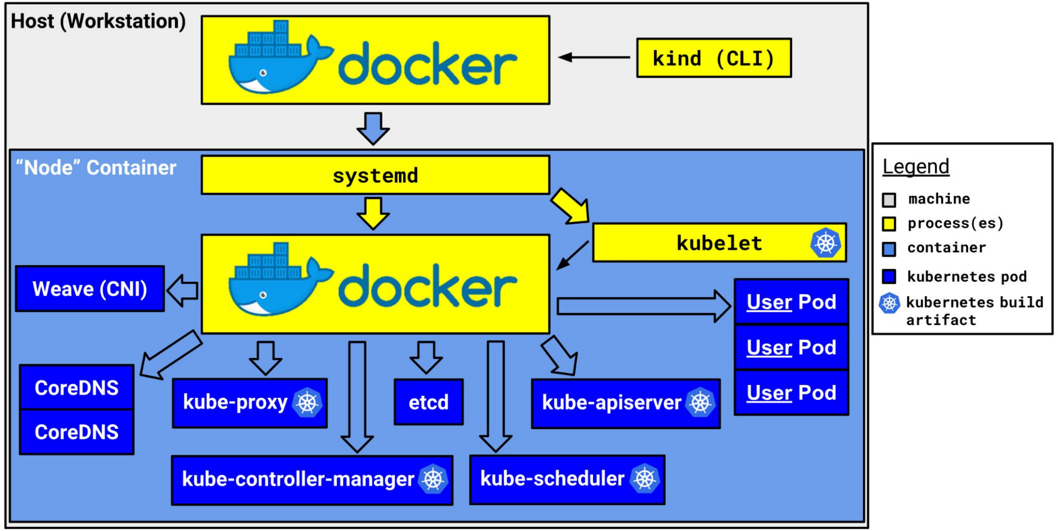
A cheat sheet for `kubectl` is provided here.

- To communicate with the control plane, K8s provides a CLI tool called `kubectl`.
- `kubectl` finds and switches between different K8s clusters by recording **contexts** in its configuration file.
- There are 3 approaches to manage K8s resources/objects:
  a. Imperative commands
     - Operate (create, replace, etc.) directly on live cluster objects.
     - E.g., `kubectl create deployment nginx --image=nginx`
  b. Imperative object configuration
     - Store object state configurations in YAML/JSON files, then use commands to operate on these objects via configuration files.
     - E.g., `kubectl create -f nginx-service.yaml`
  c. **Declarative object configuration**
     - Let `kubectl` examine a group of configuration YAML/JSON files and automatically detect & apply changes without specifying the actual operations
     - E.g., `kubectl apply -f configs/`

Kind for K8s

- "Kind is a tool for running local K8s clusters using Docker container nodes."
- K8s usually manages multiple cluster nodes, but sometimes people want to test the cluster on a local laptop environment. Kind can set up cluster nodes as docker containers.
- Compared to other solutions like minikube and k3s, Kind is lightweight, fast, and test-centric. These features suit the CI/CD scenario well.

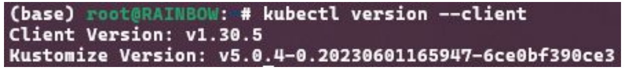
kind - Kubernetes IN Docker



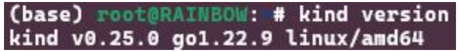
https://kind.sigs.k8s.io/docs/design/initial/

Enable kubectl auto-completion (reference): echo 'source <(kubectl completion bash)' >>~/.bashrc

TASK: Kind for K8s - Quickstart



```
(base) root@RAINBOW:~# kubectl version --client
Client Version: v1.30.5
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
```

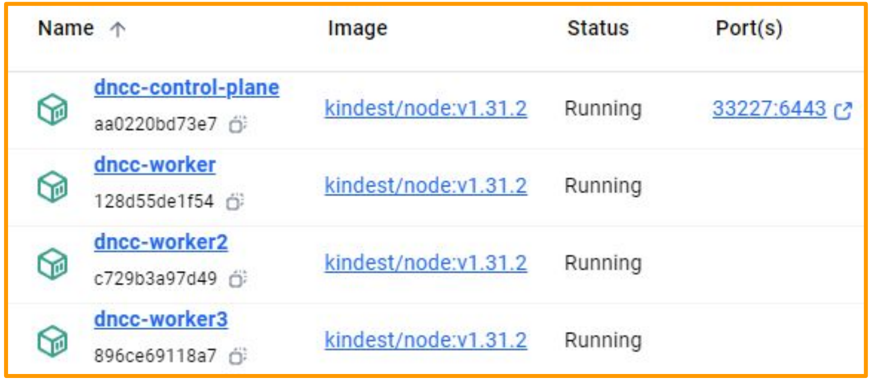


```
(base) root@RAINBOW:~# kind version
kind v0.25.0 go1.22.9 linux/amd64
```

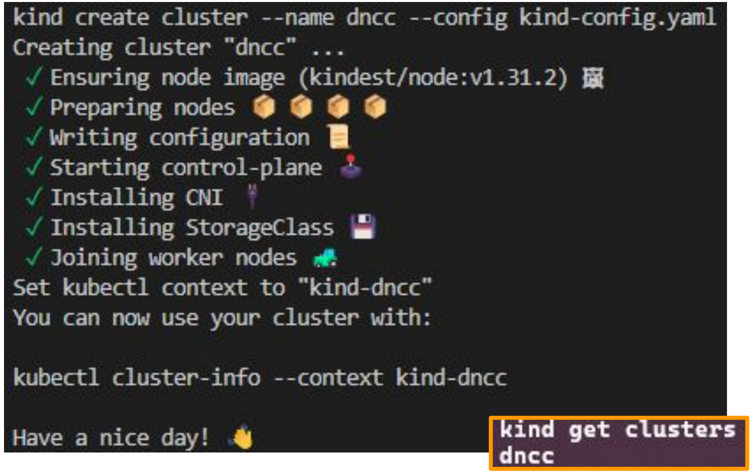
Use Kind to explore Kubernetes on a single machine.

> Reference codebase: `k8s_quickstart`

1. Set up Kubernetes CLI - `kubectl` (link; verify via `kubectl version --client`).
2. Set up Kind (link - manually download if cURL is too slow; verify via `kind version`).
3. Create/Update the `kind-config.yaml` to specify a cluster template with 1 control plane + 3 worker nodes. Then, create the multi-node cluster with name “dncc” using Kind:
   - `kind create cluster --name dncc --config kind-config.yaml`
   - Verify with `kind get clusters`



Name ↑	Image	Status	Port(s)
dncc-control-plane aa0220bd73e7	kindest/node:v1.31.2	Running	33227:6443
dncc-worker 128d55de1f54	kindest/node:v1.31.2	Running	
dncc-worker2 c729b3a97d49	kindest/node:v1.31.2	Running	
dncc-worker3 896ce69118a7	kindest/node:v1.31.2	Running	



```
kind create cluster --name dncc --config kind-config.yaml
Creating cluster "dncc" ...
 ✓ Ensuring node image (kindest/node:v1.31.2) 🖼
 ✓ Preparing nodes 📦 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-dncc"
You can now use your cluster with:

kubectl cluster-info --context kind-dncc

Have a nice day! 👋
```

```
kind get clusters
dncc
```

The same effect can be applied via `kubectl apply -f nginx-service.yaml`

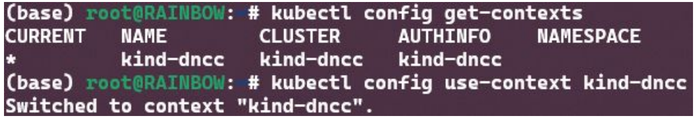
Remember to clean the k8s resources and kind clusters after testing.

TASK: Kind for K8s - Quickstart

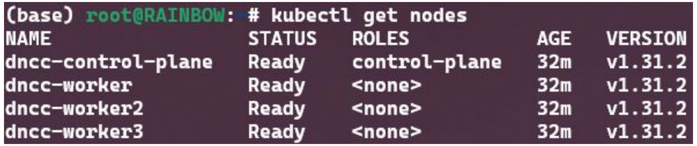
Use Kind to explore Kubernetes on a single machine.

> Reference codebase: `k8s_quickstart`

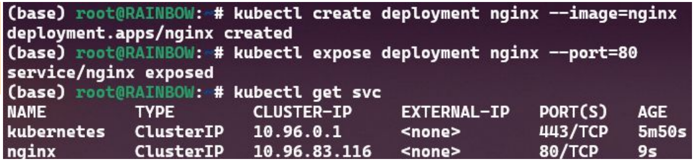
4. Switch to the cluster context if not already:
   - ○ `kubectl config use-context kind-dncc`
5. Now use `kubectl` to play with the cluster.
6. Check nodes in the cluster:
   - ○ `kubectl get nodes`
7. Test imperative commands now.
8. `kubectl create deployment nginx --image=nginx`
9. `kubectl expose deployment nginx --port=80`
   - ○ Record Cluster-IP of the exposed service:
     i. `kubectl get svc`
10. Enter control plane node and test connection to the nginx server now.
11. `docker exec -it dncc-control-plane /bin/bash`
    - ○ `curl <Cluster-IP>`
12. Explore other `kubectl` commands (link).



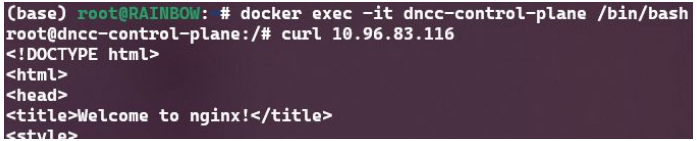
```
(base) root@RAINBOW:~# kubectl config get-contexts
CURRENT   NAME        CLUSTER     AUTHINFO    NAMESPACE
*         kind-dncc   kind-dncc   kind-dncc
(base) root@RAINBOW:~# kubectl config use-context kind-dncc
Switched to context "kind-dncc".
```



```
(base) root@RAINBOW:~# kubectl get nodes
NAME                  STATUS   ROLES           AGE   VERSION
dncc-control-plane    Ready    control-plane   32m   v1.31.2
dncc-worker           Ready    <none>          32m   v1.31.2
dncc-worker2          Ready    <none>          32m   v1.31.2
dncc-worker3          Ready    <none>          32m   v1.31.2
```

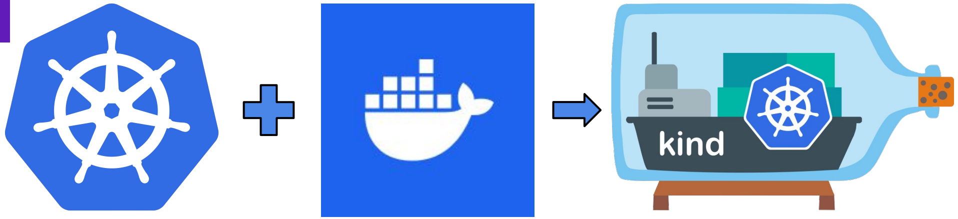


```
(base) root@RAINBOW:~# kubectl create deployment nginx --image=nginx
deployment.apps/nginx created
(base) root@RAINBOW:~# kubectl expose deployment nginx --port=80
service/nginx exposed
(base) root@RAINBOW:~# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP   5m50s
nginx        ClusterIP   10.96.83.116   <none>        80/TCP    9s
```



```
(base) root@RAINBOW:~# docker exec -it dncc-control-plane /bin/bash
root@dncc-control-plane:/# curl 10.96.83.116
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
```

Summary



- Tracing back:
  - Virtualization → Containerization
  - Docker / Docker Compose / Docker Swarm → Kubernetes (K8s)
- K8s Components
  - Control Plane: etcd, API Server, Controller Manager & Cloud Controller Manager (*), Scheduler
  - Worker Node: kubelet, kube-proxy (*), Container Runtime
- K8s Resources/Objects
  - Pod
  - ReplicaSet & Deployment
  - Service
  - Others: StatefulSet, Job, AutoScalers
- K8s CLI - `kubectl`
  - Imperative commands *vs.* Declarative object configuration
- Kind for K8s: multi-machine cluster on 1 local machine - nodes as docker containers