Intro. to Kubernetes (K8s)

ytshih (2026, CC BY-SA)

Outline

- Background
- What is Kubernetes
- Kubernetes Components
  - Control Plan Components
  - Node Components
- Kubernetes Plugin Interfaces
- Bootstrapping a Kubernetes Cluster

Background

Software Development and Operations - Microservices

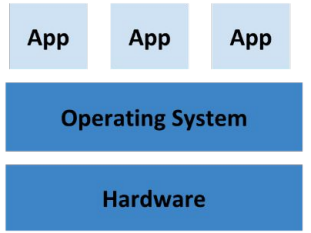
- Monolithic Architecture
  - One-for-all application that do all the things (auth, view, logic)
  - Provides better response time and less overhead
  - Struggles to scale under heavy, high-volume traffic
- Microservices Architecture
  - Easier to test independently
    - Faster time-to-market
    - Easier to develop when the application is too big
  - Hard to operate
    - The application can have hundreds of services

Software Development and Operations - CI/CD

- Continuous Integration
  - Delivers a stream of small, frequent changes
  - Tested by automated deployment pipeline and deployed into production
    - Preserve product quality from large develop team
    - Less human intervention
- Continuous Delivery
  - Deliver to production environment

What we have before containers

- Traditional (Bare metal) deployment
  - Run applications on physical servers
  - Back in the days
    - Restricted hardware capabilities (Computility, Disk, Memory...)
      - e.g. C10k problem in *1999*
    - Monolithic applications
  - New service? New server!

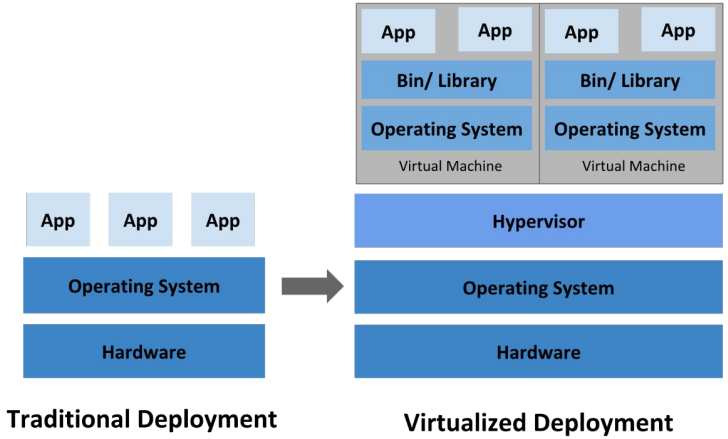


Traditional Deployment

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

What we have before containers (cont.)

- Virtualized deployment
  - Run multiple Virtual Machines on a single physical server
  - **K**ernel-based **V**irtual **M**achine was merged into mainline Linux 2.6.20 (2007)
  - Allows isolation on the same machine
  - Better scalability than bare metal deployment
  - New service? New virtual machine!

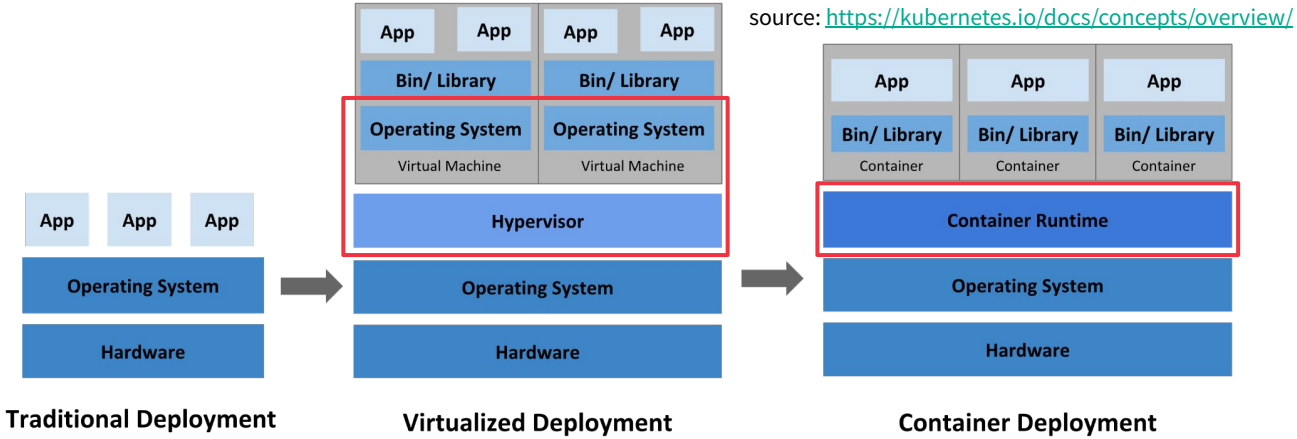


source: https://kubernetes.io/docs/concepts/overview/

What we have before containers (cont.)

- Container deployment
  - Share the Operating System among the applications
  - User namespaces support was merged in mainline Linux 3.8 (2013)
  - Agile application creation
  - Decoupling application from infrastructure

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



What do containers solve

- From bare metal
  - Lacking resource boundaries when running multiple services on the same machine
  - Expensive to maintain many physical servers and scale up
- From VM
  - The additional cost from operating system

When shouldn’t I use containers

- Need ABSOLUTE isolation
  - For stability, security, ...
- Need many OS-level functionalities that can’t be containerized
  - e.g. Running different versions of kernel modules
- ~~Afraid of YAML~~

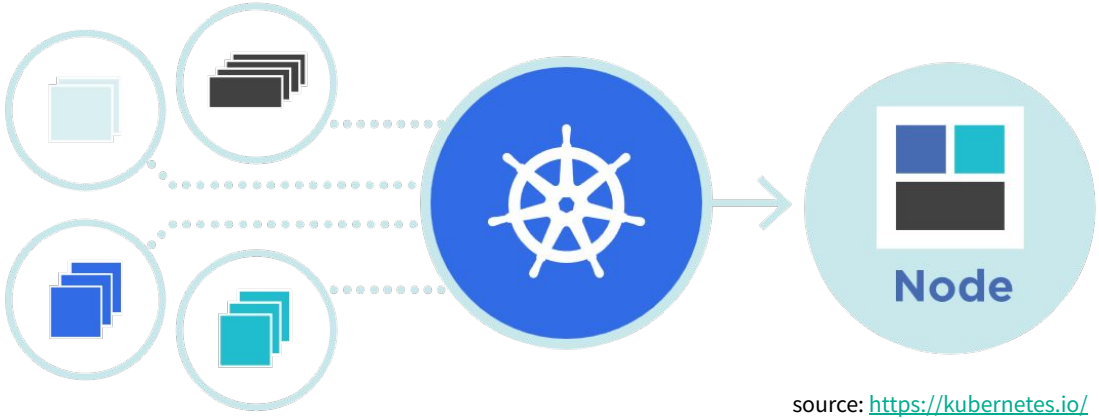
What we have learned before Kubernetes in SA

- Systemd service
  - For apps that run in bare metal / VM
  - Hard to isolate running environment
    - e.g. libraries
  - Hard to rollout and rollback
- Docker
  - The go-to option for local containers
  - FYI: Support high availability and load balancing via Docker Swarm
    - Focus on easy to use, drop-in solution for existing docker-compose file
    - Lack of functionality compared to Kubernetes
      - e.g., Role-Based Access Control (RBAC), cloud vendor support

What is Kubernetes

What is Kubernetes

- A portable, extensible, open source platform for managing **containerized** workloads
- Facilitate both **declarative configuration** and **automation**
- Has a large, rapidly growing ecosystem



source: https://kubernetes.io/

Why do we need Kubernetes

- Service discovery and load balancing
- Storage orchestration (PV, PVC)
- Automated rollouts (upgrade) and rollbacks (downgrade)
- Service self-healing (auto restart)
- ...
- **Fits well with modern workflows**

Kubernetes Alternatives for Running Services

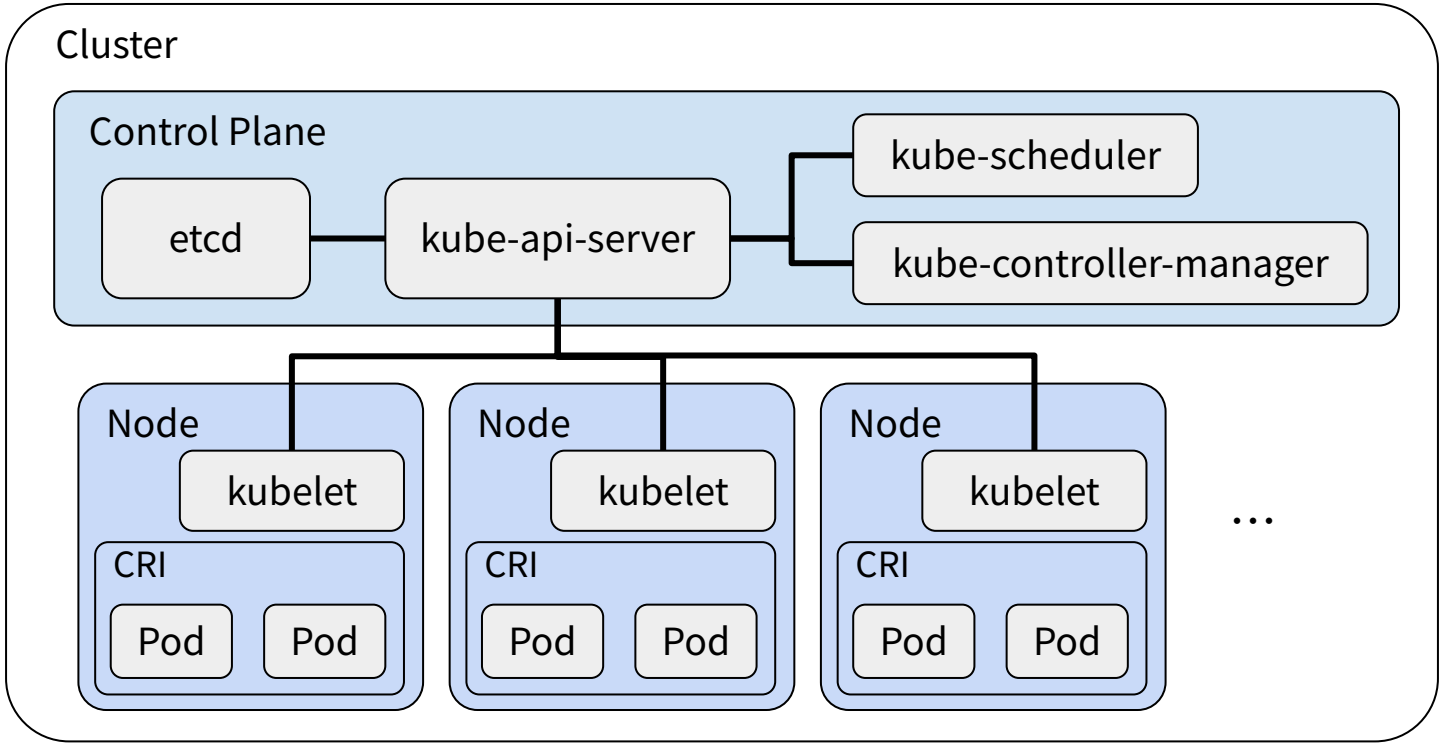
- HashiCorp Nomad (2015)
- Docker Swarm (2016)
- Amazon ECS / Google Cloud Run / AWS Fargate ...
- Other Platform-as-a-Service(PaaS) solutions

Kubernetes Distributions

- Packaged version of K8s, with additional tools, or extensions
- Cloud managed
  - Amazon Elastic Kubernetes Service (EKS)
  - Google Kubernetes Engine (GKE)
  - Azure Kubernetes Service (AKS)
- Common on-premise distributions
  - **K3s (Rancher Labs, SUSE)**
  - **Mirantis K0s**
  - Red Hat OpenShift
  - VMware Tanzu Kubernetes Grid
  - MicroK8s (Canonical)
  - Rancher Kubernetes Engine (RKE)
  - Amazon EKS Anywhere
  - Docker Desktop Kubernetes
  - Mirantis Kubernetes Engine (MKE)
  - Sidero Labs Talos Linux

Kubernetes Components

Cluster Architecture Overview



Kubernetes Components

- Control Plane Components
  - Manage overall state for Kubernetes cluster
- Node Components
  - Run on **every node**
  - Maintain running pods
  - Provide the Kubernetes runtime environment
- Note that Control Plane might not be a pod, or a process
  - The environment in which components run can vary
  - It’s more like a logical thing that provide a certain functionality that we need

Control Plane Components

- kube-apiserver
  - The core component server that exposes the Kubernetes HTTP API
- etcd
  - Consistent and highly-available key value store for all API server data
- kube-scheduler
  - Assign Pods not yet bound to a node
- kube-controller-manager
  - Runs controllers to implement Kubernetes API behavior

Node Components

- kubelet
  - Ensures that Pods are running
- kube-proxy (optional)
  - Maintains network rules on nodes to implement Services
- Container runtime
  - Software responsible for running containers

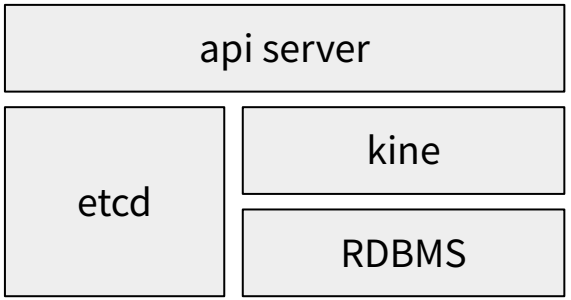
Control Plane Components

kube-apiserver

- Provides Kubernetes RESTful API
- Design to scale horizontally
  - Scales by deploying more instances
  - Other components perform leader election using EndpointSlices object
- The **only** components to communicate with etcd
  - Receive both internal and external requests for Kubernetes resources

etcd

- Consistent and highly-available key value store
- Used for all cluster data
- Sensitive to network and disk I/O latency
- Kine (Kine is not etcd) can be used to store data on RDBMS
  - An etcdshim that translates etcd API
    - shim: library that transparently intercepts API calls
  - Kine is used in some K8s distributions, such as K3s



kube-scheduler

- Assigns a node for each newly created Pod according to the following resource requirements:
  - Hardware
  - Software
  - Policy constraints
  - Affinity and anti-affinity specifications
  - Data locality
  - Inter-workload interference
  - Deadlines

kube-controller-manager

- Runs built-in controllers
  - **Node controller**: Responsible for noticing and responding when nodes go down
  - Job controller: Watches for job objects and creates Pods to run
  - EndpointSlice controller: Provide links between Services and Pods
  - ServiceAccount controller: Create default ServiceAccounts for new namespaces
  - ...

More on Controllers

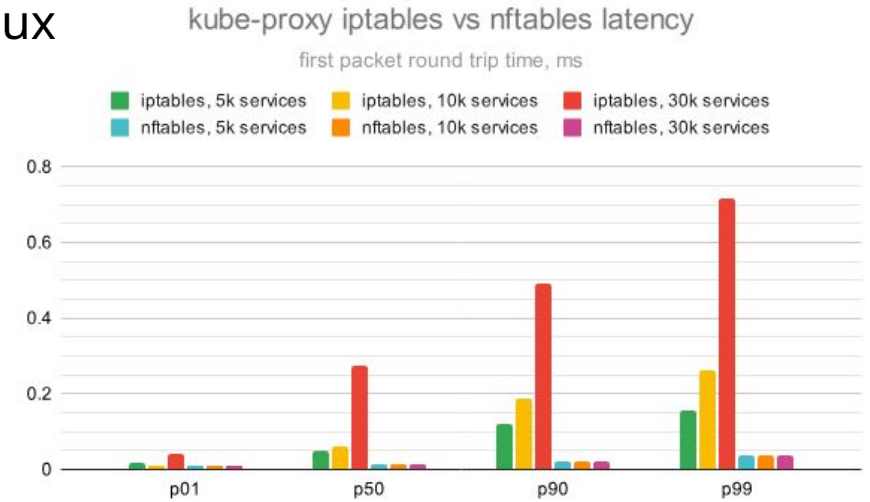
- Watch the state, then make or request changes to resources
  - Current state => Desired state
  - For example, node controller checks heartbeat of each node
    - Update node's condition to *Unknown* if a node fails
- Can be deployed outside the control plane
  - Run as a set of Pods on worker nodes
  - External to Kubernetes

kubelet

- Makes sure that containers are running in a Pod
- Ensures that the containers described in PodSpecs are running and healthy
- Directly manages **static Pods**
  - Pods that bound to one kubelet on a specific node

kube-proxy

- A network proxy that runs on each node
- Implements part of the Kubernetes **Service** concept
  - Forwards packets for Services
- Use iptables, ipvs or nftables on Linux
  - nftables mode is GA as of v1.33

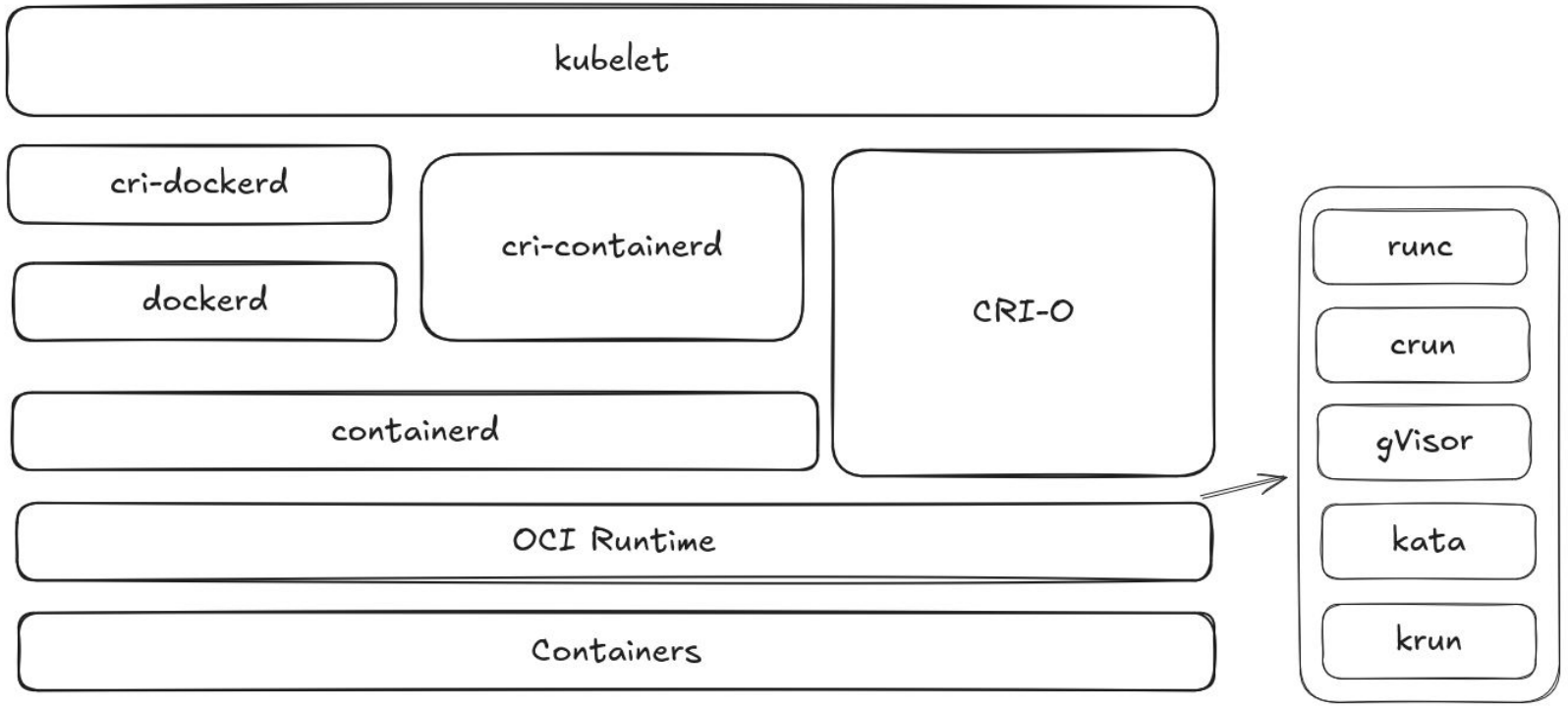


source: https://kubernetes.io/blog/2025/02/28/nftables-kube-proxy/

Container Runtimes

- Responsible for running containers
- Requires to use a runtime that conforms with the **Container Runtime Interface (CRI)** after v1.35
- Common container runtimes
  - containerd
  - CRI-O
  - Docker Engine
  - Mirantis Container Runtime

More on Container Runtimes



More on OCI Runtimes

- Runtimes that conforms OCI Runtime Specifications
- Implementations
  - opencontainers/runc
  - containers/crun
  - containers/crun - krun
  - google/gvisor
  - kata-containers/runtime

Kubernetes Plugin Interfaces

Kubernetes Plugin Interfaces

- Container Runtime Interface
- Container Network Interface
- Container Storage Interface
- Container Device Interface

Container Runtime Interface

- The main protocol for the communication between the kubelet and Container Runtime
- A Container Runtime is needed for each node so that the Pods can run
- The runtime should conforms with the Container Runtime Interface

Container Network Interface

- To configure network interfaces in Linux containers
- Concerns itself only with
  - Network connectivity
  - Recycling allocated resources
- Common CNI plugins
  - flannel-io/flannel
  - projectcalico/calico
  - cilium/cilium

Bootstrapping a cluster with kubeadm

kubeadm

- A tool built to provide best-practice “fast paths” for creating Kubernetes clusters
- Only cares about bootstrapping, not about provisioning machines
- User have to manually install kubelet and container runtime
  - Not shipped with kubeadm unlike k3s or k0s
- Spawn control plane components in Kubernetes resources

More on Control Plane

- Control plane can be inside control plane Nodes by containers
  - e.g. kubeadm, Talos Linux
- Or externally managed
  - e.g. k0s, k3s, managed by single binary
  - e.g. Arch Linux kubernetes-control-plane, by distro packages

Environment

- Debian 13 (or anything with Linux kernel >= 5.13, preferably LTS)
- Install a Container Runtime (we use CRI-O here)
- Install kubeadm, kubelet, kubectl
- The **version** of cri-o, kubeadm, kubelet, kubectl should match
- Disable swap
  - `swapoff -a`
- Set `net.ipv4.ip_forward=1`
  - `sysctl -w net.ipv4.ip_forward=1`
- Enable `br_netfilter`
  - `modprobe br_netfilter`

Bootstrap cluster

- Bootstrap a single node (control plane + worker node) for simplicity
  - **DO NOT** put control plane and worker node on the same machine in production environment
  - Control plane should have 3 nodes in order to provide HA
  - You should figure out how to bootstrap a fully HA cluster in HW
- Specify pod network CIDR for CNI plugin

```
kubeadm init --pod-network-cidr=10.244.0.0/16
```

Kubernetes Resources

Kubernetes Resources

- Node
  - Machine to run Pods on
- Namespace
  - Isolating groups of resources within cluster
- Pod
  - The smallest deployable unit of containers
- Other resources will be covered in the next lecture

Node

- Kubernetes runs workload by placing containers into Pods to run on Nodes.
- Node can have labels and taints.
  - Labels: Identifying attributes of nodes for affinity (e.g. arch, OS, hostname)
  - Taints: Allowing node to repel a set of pods (e.g. NoSchedule, NoExecute)

Namespace

- Isolate groups of resources within a single cluster
- Only for namespaced objects
  - Deployments, Services, etc.
  - Not for cluster-wide objects, e.g. StorageClass, Nodes
- Default namespaces
  - default
  - kube-system
  - ...

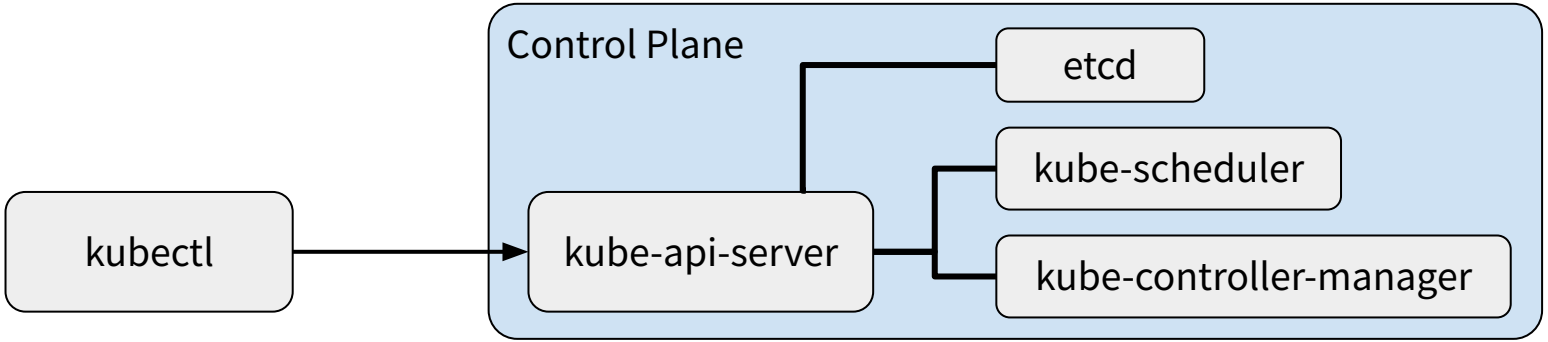
Pod

- A group of one or more containers
  - With shared storage and network resources
- The smallest deployable units of computing
  - Co-located and co-scheduled
  - Usually runs a single container
- Pod can be scheduled with affinity (according to labels) and toleration (according to taints).

Using Kubernetes

kubectl

- Command line tool for communicating with control plane using Kubernetes API
- Read `$HOME/.kube/config` for client config file
  - Alternatively, specify by `KUBECONFIG` env var or `--kubeconfig` flag



Configure kubectl

- kubectl reads config file in `$HOME/.kube/config`
- kubeadm puts admin config to the cluster in `/etc/kubernetes/admin.conf`
- Move the admin config to your home directory
- Setup shell auto completion
- Set alias for `kubectl` (cause you will use it a lot)

```
mkdir -p $HOME/.kube && sudo cp /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
source <(kubectl completion bash)
echo "source <(kubectl completion bash)" >> ~/.bashrc
alias k=kubectl
complete -o default -F __start_kubectl k
```

Setup Kubernetes

- Remove the **taint** that prevents workloads to schedule on
  - Kubeadm apply NoSchedule on control plane nodes for stability
  - In the single node setup we have to remove the taint
- Install flannel for CNI plugin
  - You will see the status of the node is `NotReady` before CNI plugin applied

```
kubectl taint node $HOSTNAME node-role.kubernetes.io/control-plane:NoSchedule-
kubectl apply -f https://github.com/flannel-io/flannel/releases/latest/download/kube-flannel.yml
```

First glimpse of Kubernetes

- `kubectl [command] [TYPE] [NAME] [flags]`
- Print the supported API resources on the server
  - `kubectl api-resources`
- Create a new namespace named `test`
  - `kubectl create namespace test`
- Describe contexts in kubeconfig
  - `kubectl config get-contexts`
- Switch namespace to `test`
  - `kubectl config set-context --current --namespace test`

Create your first pod

- Run a busybox image as pod named `test` on the cluster with custom command `sh`
  - `kubectl run --image busybox -it test -- sh`
- Display pods that currently running
  - `kubectl get pods`
- Execute `hostname` in the main container of pod `test`
  - `kubectl exec test -- hostname`

Inspect your first pod

- Display pod `test` in yaml
  - `kubectl get pod/test -o yaml`
- Show details of pod `test`
  - `kubectl describe pod/test`
- Get documentation for `pod.spec`
  - `kubectl explain pod.spec`
- Delete pod `test`
  - `kubectl delete pod/test`

References

- Kubernetes - https://kubernetes.io
- NFTables mode for kube-proxy - https://kubernetes.io/blog/2025/02/28/nftables-kube-proxy/
- flannel - https://github.com/flannel-io/flannel
- Kubernetes Distributions: Types, Popular Distros and Best Practices - https://www.tigera.io/learn/guides/kubernetes-security/kubernetes-distributions/
- Microservice Architecture pattern - https://microservices.io/patterns/microservices.html
- Kubernetes in Action, Second Edition - Marko Lukša, Kevin Conner