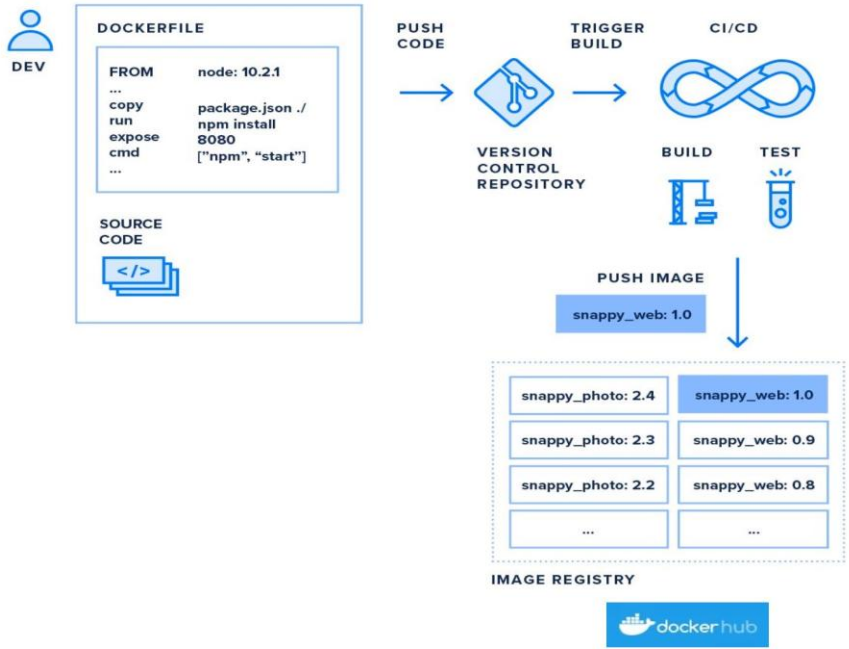
Container Orchestration with Kubernetes

Outline

- Introduction to Container Orchestration
- Orchestration Technologies Simplified
- Introduction to Kubernetes
- Concept of a Kubernetes cluster
- Exploring Kubernetes Architecture

Container Ecosystem

- Container
- Container Images
- Container Runtime
- Container Registries

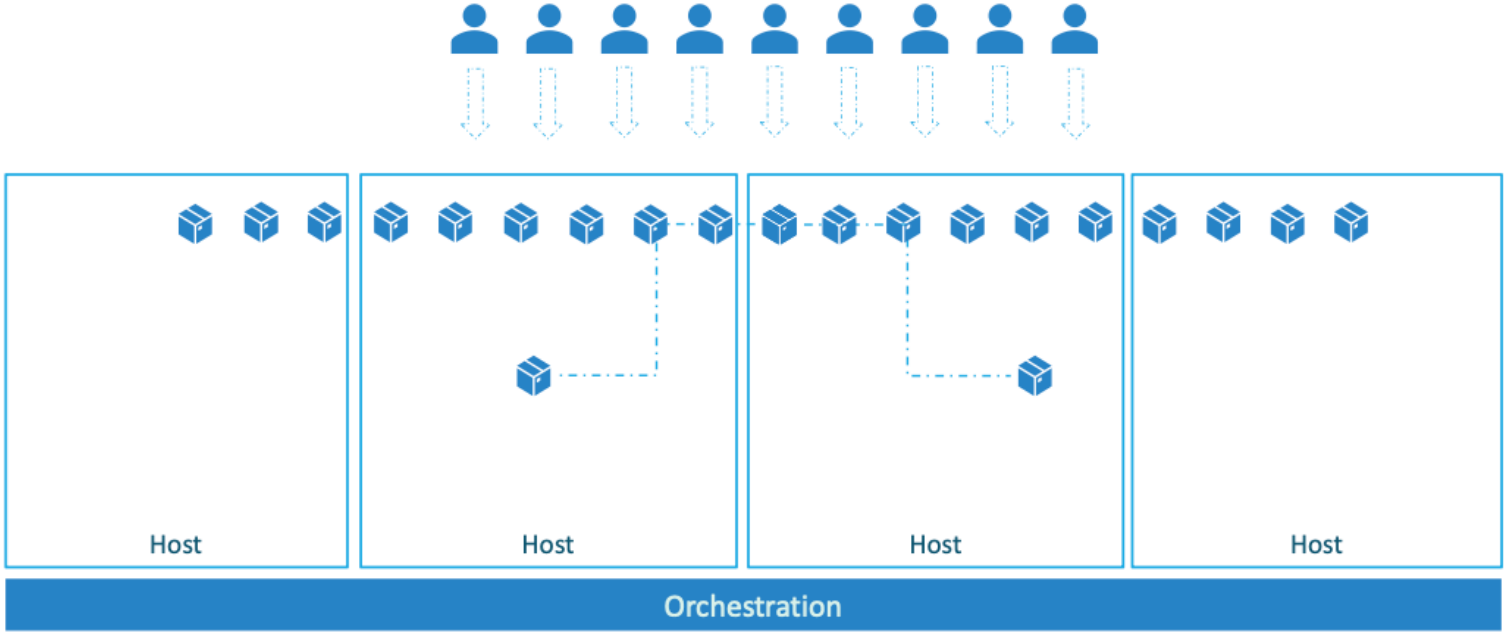


Container Clusters

- What happens when we have lots of containers running on many virtual machines?

- How can we make it easy to deploy, scale, restart, and manage all these containers?

Container Orchestration



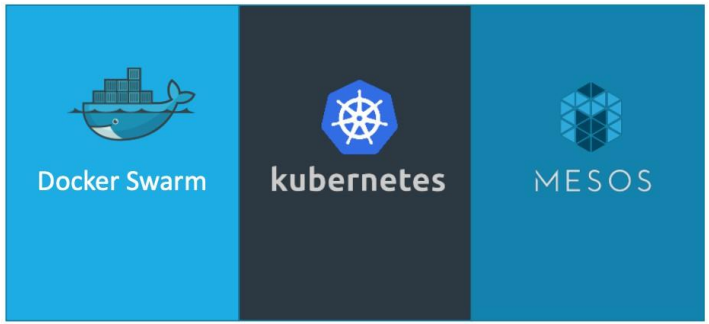
Cont.

- An orchestrator is a system that deploys and manages applications. For example, Kubernetes can:
  - Deploy your application
  - Scale it up and down dynamically based on demand
  - Self-heal it when things break
  - Perform zero-downtime rolling updates and rollbacks
- The best part? Once set up, Kubernetes operates autonomously, requiring minimal supervision.

Cont.

- What problems do container orchestration solutions solve in this scenario?
  - Management
  - Abstraction of hardware
  - Networking
  - Scheduling
  - Scaling
  - Deployment
  - Service Discovery

Orchestration Technologies



- Docker Swarm
  - Docker's container orchestration tool
  - Easy setup and quick start
  - Limited advanced autoscaling capabilities
- Mesos by Apache
  - Apache Mesos for container orchestration
  - Complex setup, steep learning curve
  - Supports many advanced features

Kubernetes (K8s)

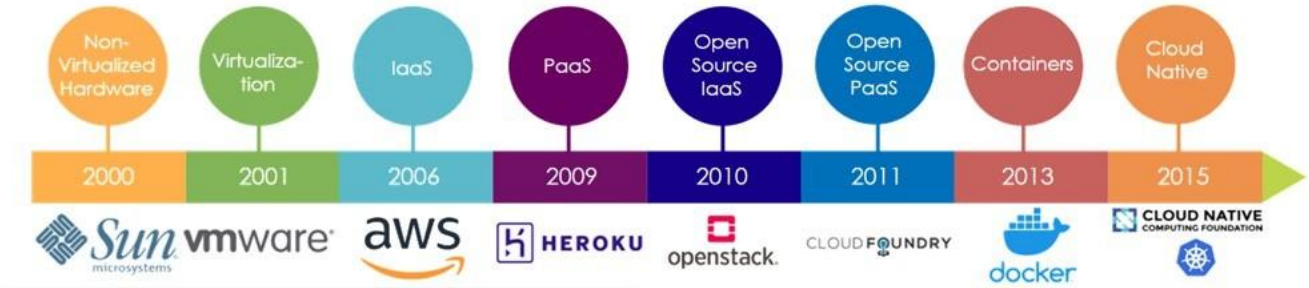
- **Brief Kubernetes History -**

Kubernetes, also known as k8s, is an open-source system for automating deployment, scaling, and management of containerized applications.

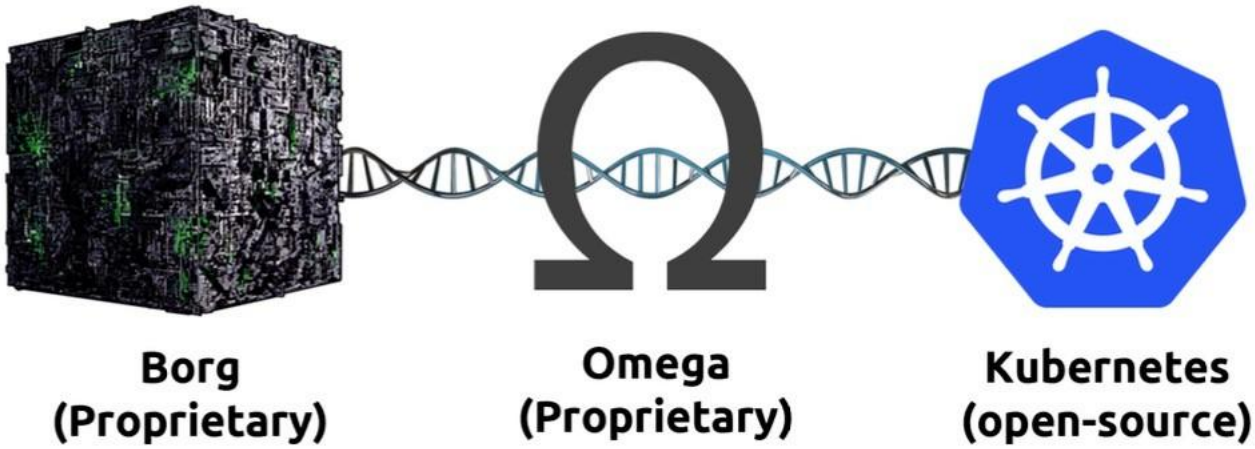
Name Kubernetes – Greek, meaning helmsman or pilot.
K8s- the eight letters between the "K" and the "s".
Google open-sourced the Kubernetes project in 2014.
15 years of Google’s experience.

Cont.

- Grew in popularity, open-source velocity increased
- Now the most popular container cluster (most cloud platforms have managed K8s offering)
- Features added regularly and frequently
- Cloud Native / CNCF - Kubernetes, Prometheus, Fluentd



Cont.


Borg
(Proprietary)
Omega
(Proprietary)
Kubernetes
(open-source)

Cont.

- Computers have hardware components like CPU, memory, storage, and networking.
- Modern operating systems hide these complexities from developers.
- Kubernetes does the same for cloud and datacentre resources.

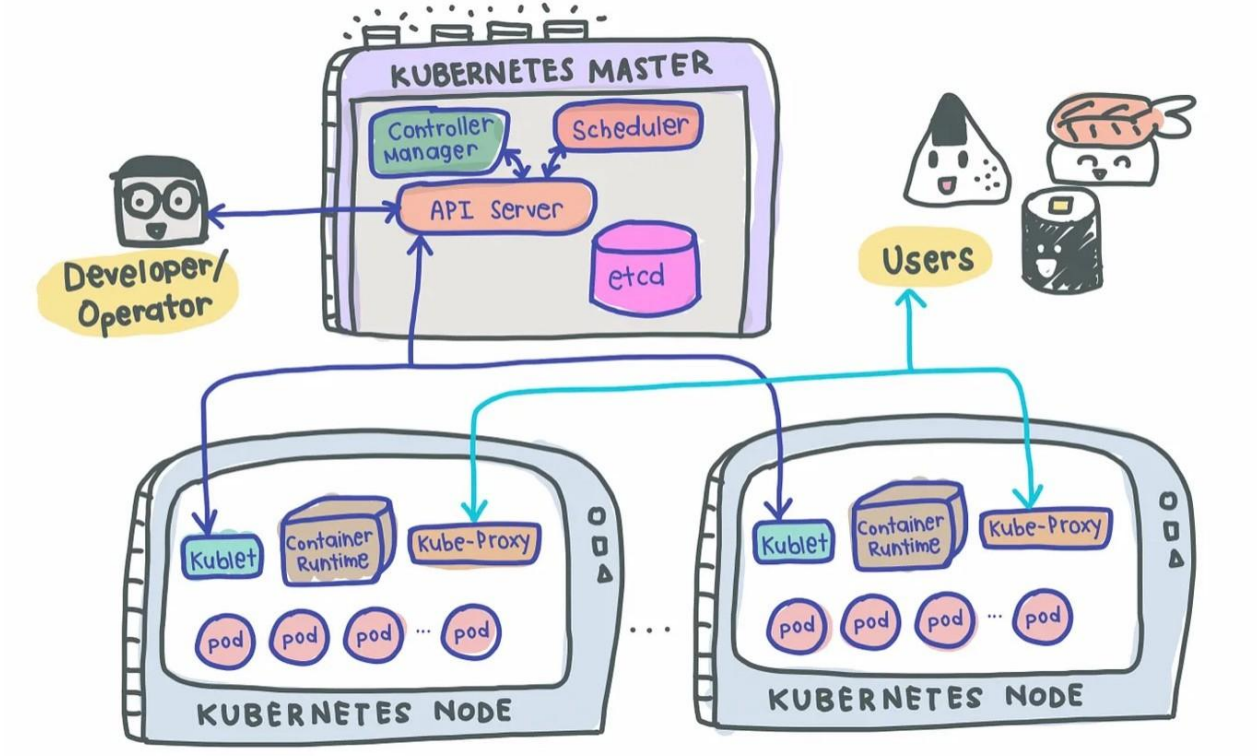
Cont.

- It handles resource allocation, eliminating manual configuration.
- No need to name servers or manage infrastructure assets.
- Developers tell Kubernetes to deploy and manage apps in a cloud-native way.

Cont.

- Kubernetes principles of operation –
  - At the highest level, Kubernetes is two things:
    - A cluster to run applications on
    - An orchestrator of cloud-native microservices apps

Kubernetes as a cluster



Cont.

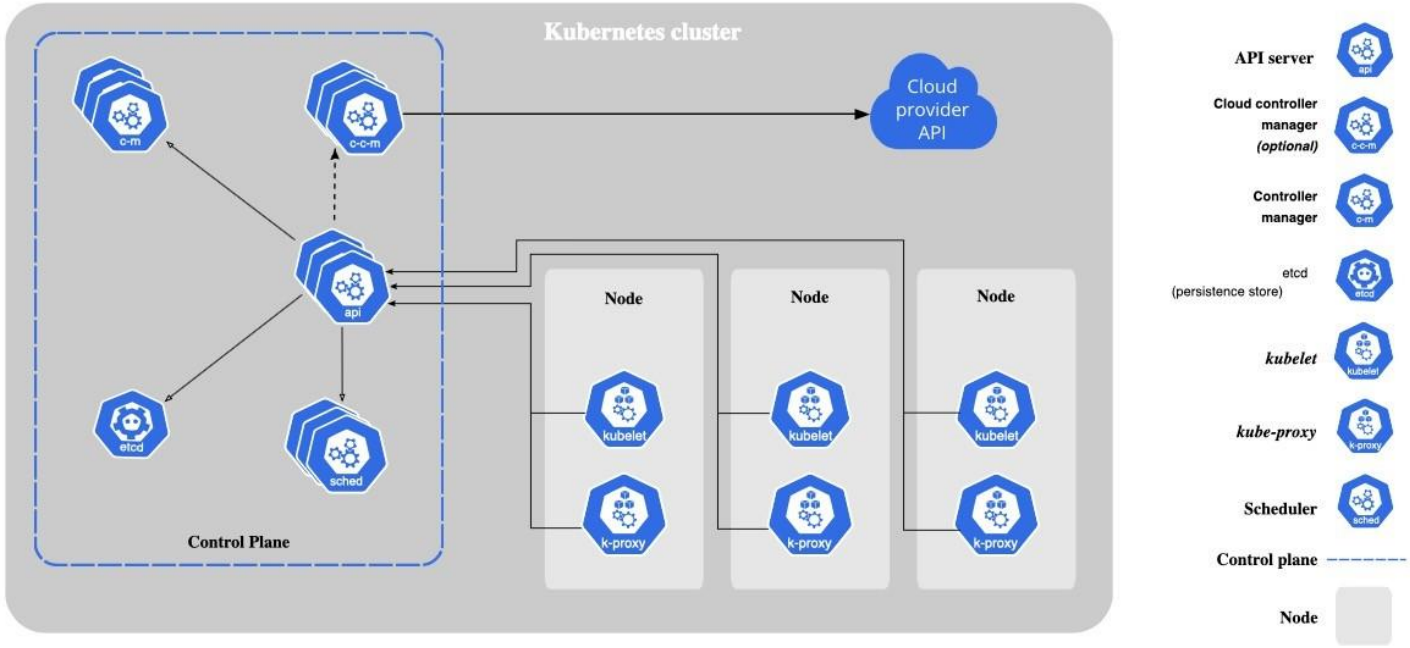
- Kubernetes is like any other cluster –

- A bunch of machines to host applications. We call these machines “nodes”, and they can be physical servers, virtual machines, cloud instances, Raspberry Pis, and more.

Cont.

- Kubernetes cluster consists of a control plane and nodes.
- Control plane handles API, scheduling, and cluster state storage.
- Nodes execute user applications.
- Control plane: The "brains" of the cluster, responsible for smart features.
- Nodes: The "muscle," handling everyday application execution tasks.

Cont.


Kubernetes cluster
c-m
c-c-m
Cloud provider API
api
etcd
sched
Control Plane
Node
Node
Node
kubelet
kubelet
kubelet
k-proxy
k-proxy
k-proxy
API server
Cloud controller manager (optional)
Controller manager
etcd (persistence store)
kubelet
kube-proxy
Scheduler
Control plane
Node

Kubernetes as an orchestrator

- Microservices are like individual players on a football team.
- Kubernetes plays the role of the coach, organizing and managing the team.
- It ensures a coordinated and smooth performance, reacting to events.

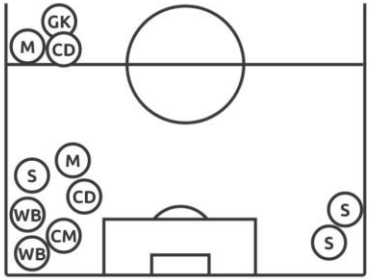


Figure 2.1

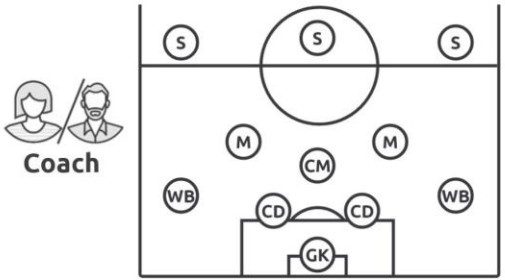


Figure 2.2

- In sports, this is called coaching; in apps, it's orchestration.
- Kubernetes orchestrates cloud-native microservices, just like a coach does in football.

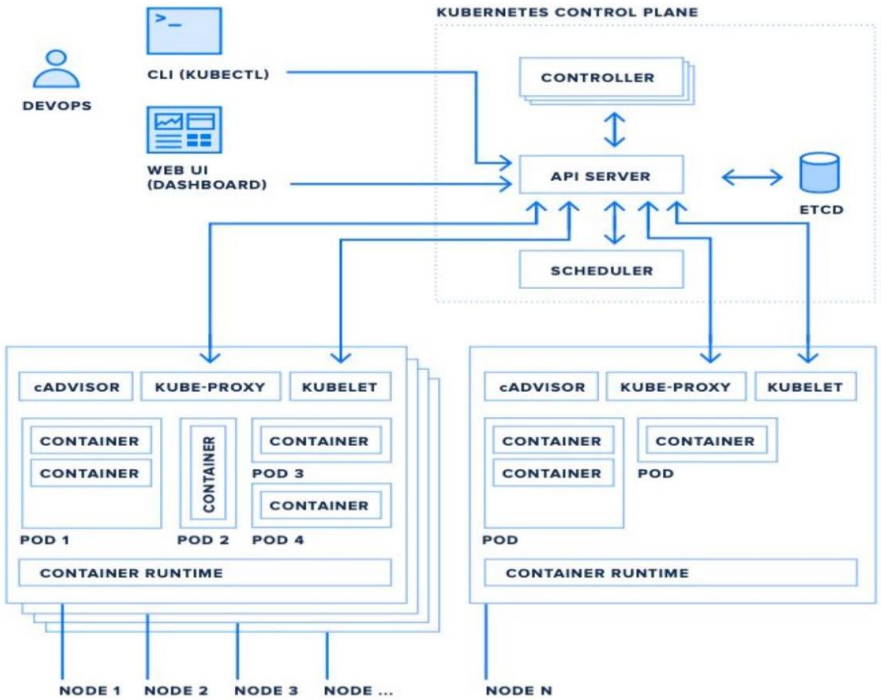
Cont.

- Let’s look at how it works?
  - Running applications on Kubernetes involves a simple process:
    - Design your applications in your preferred language.
    - Package each application as a container.
    - Place containers in Kubernetes Pods.
    - Deploy Pods to the cluster using controllers like Deployments, Daemon Sets, etc.

Cont.

- Kubernetes employs controllers for features like self-healing, scaling, and rollouts. Some controllers are for stateless apps, others for stateful apps.
- Kubernetes prefers declarative management: define desired states in YAML files.
- Kubernetes watches and maintains application alignment with the declared state.
- If discrepancies arise, Kubernetes attempts to rectify them.

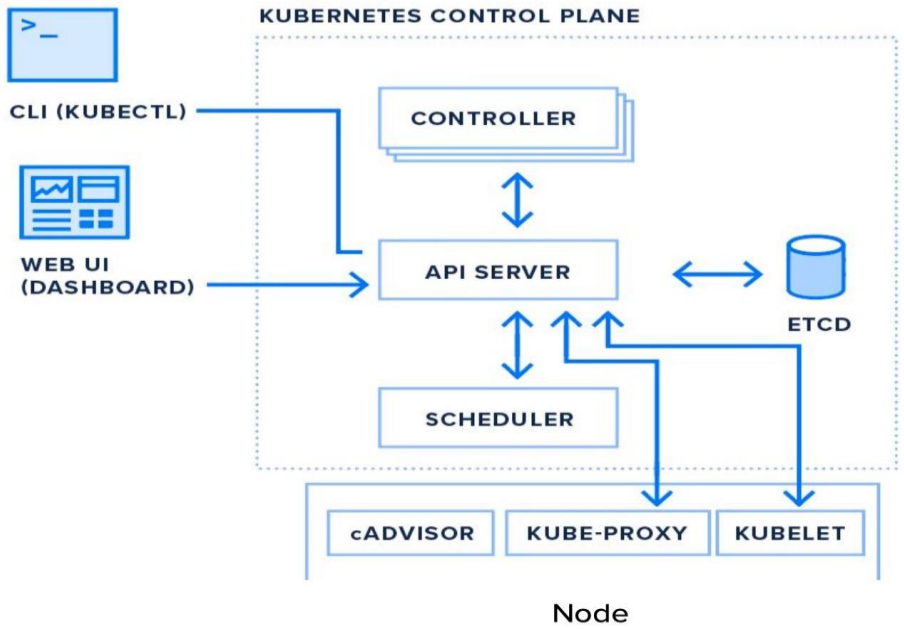
Kubernetes Architecture



- Client-Server architecture
  - Server: Control Plane
  - Clients: Nodes

Cont.

- Control Plane
  - API server
  - Scheduler
  - Controllers
    - Kubernetes
    - Cloud
  - Etcd



Cont.

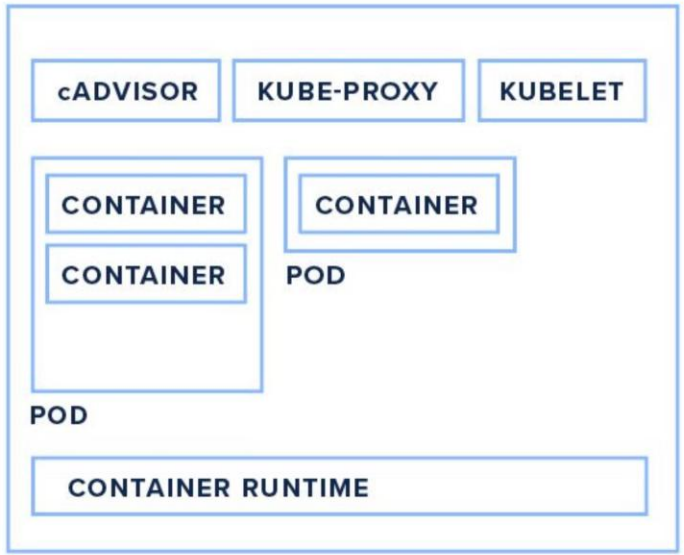
- A Kubernetes cluster consists of control plane nodes and worker nodes.
- Control plane nodes, sometimes called Masters or Heads, manage the cluster's intelligence.
- Production environments require multiple high-availability control plane nodes (typically 3 or 5).
- Best practice: Do not run user applications on control plane nodes.

Cont.

- The control plane comprises essential components, including:
  - API Server: The central communication hub, accessible via RESTful API.
  - Cluster Store: Stores cluster configuration and state, based on etcd.
  - Controller Manager: Implements background controllers for cluster monitoring.
  - Scheduler: Assigns work tasks to healthy worker nodes based on complex logic.
  - Cloud Controller Manager: Integrates cloud services in supported public cloud platforms.

Cont.

- Nodes
  - Kubelet
  - Kube-proxy
  - cAdvisor
  - Container runtime



Cont.

- Worker nodes are responsible for:
- Watching the API server for work assignments.
- Executing work assignments.
- Reporting back to the control plane via the API server.

Cont.

- Key components of a worker node:
  - Kubelet: The primary Kubernetes agent responsible for task execution and reporting.
  - Container Runtime: Used for container-related tasks (e.g., image handling).
  - Kube-proxy: Manages local cluster networking, IP assignment, and routing.
  - Kubernetes is moving away from Docker as a container runtime, favouring containerd.
  - Containerd is a lightweight, community-supported runtime compatible with the Container Runtime Interface (CRI).
  - Kube-proxy handles routing and load-balancing on the Pod network.

Cont.

- How do I interact with a Kubernetes cluster?
  - Hit REST API directly
    - Can use curl, client libraries, etc.
  - Kubectl
    - Command-line tool to interact with control plane
    - Abstracts away multiple REST API calls
    - Provides “get” “create” “delete” “describe”, etc. functionality
    - Filtering results
  - Set up kubectl
    - cp k8s_config_file ~/.kube/config
    - May need to create this directory, depending on your OS
    - kubectl cluster-info

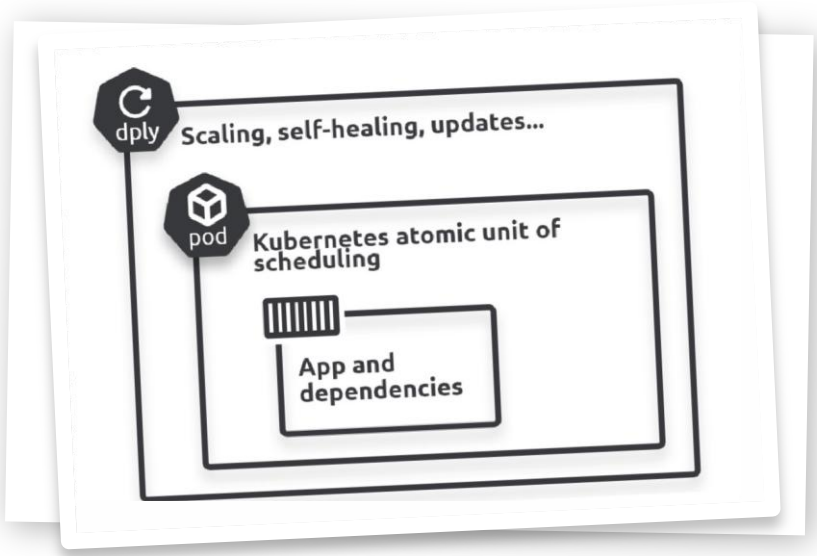
Some Kubectl Commands..

- kubectl get
- kubectl apply
- kubectl rollout status
- kubectl rollout undo
- kubectl create
- kubectl delete
- kubectl expose
- kubectl edit
- kubectl patch

Packaging applications for Kubernetes!

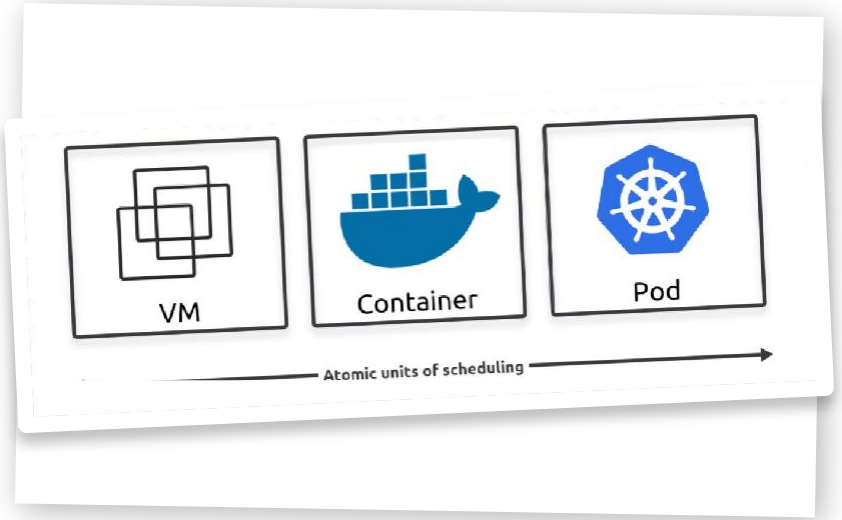
Cont.

- Packaging apps for Kubernetes
  - An application needs to tick a few boxes to run on a Kubernetes cluster. These include.
    - Packaged as a container
    - Wrapped in a Pod
    - Deployed via a declarative manifest file

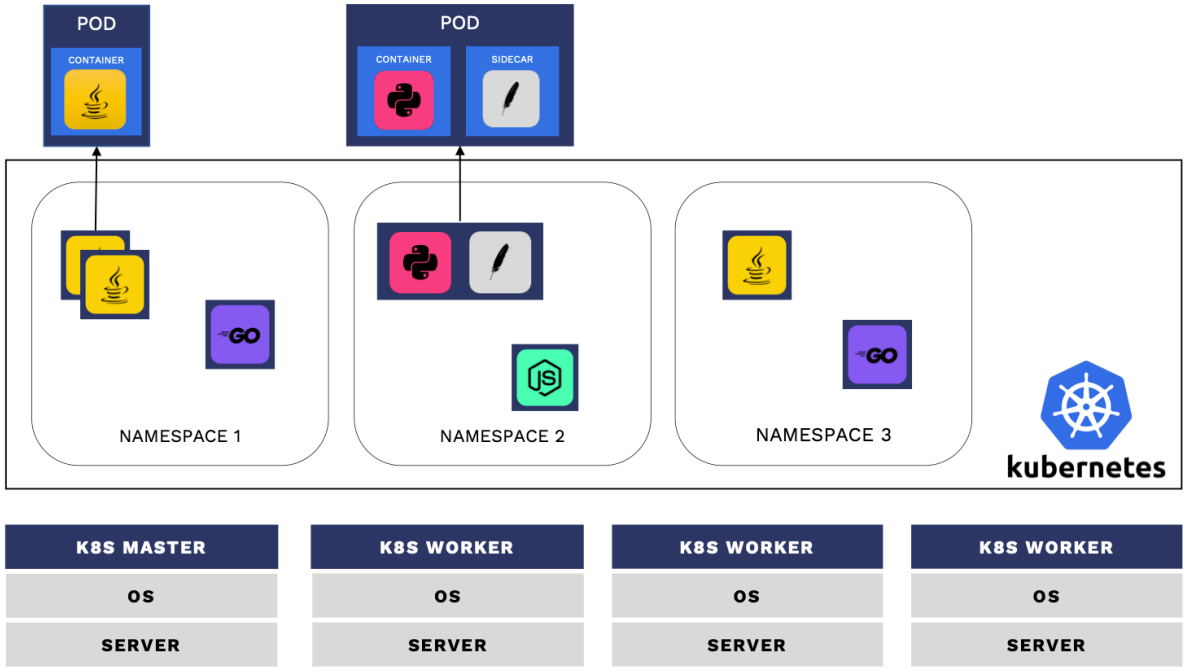


Pods

- In the Kubernetes world, the atomic unit of scheduling is the Pod.
- Every container in Kubernetes must run inside a Pod.
- Pods are constructs for running one or more containers.

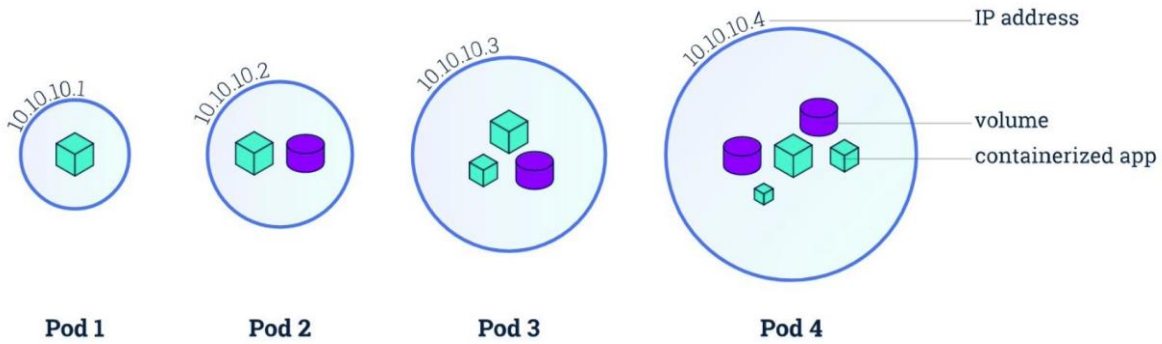


Cont.


POD
CONTAINER
POD
CONTAINER
SIDECAR
NAMESPACE 1
NAMESPACE 2
NAMESPACE 3
kubernetes
K8S MASTER
K8S WORKER
K8S WORKER
K8S WORKER
OS
OS
OS
OS
SERVER
SERVER
SERVER
SERVER

Cont.

- While a single-container Pod is common, multi-container Pods are used for specialized scenarios.
- Multi-container Pods can have tightly coupled containers sharing resources.



Cont.

- All containers within the same Pod share the same network stack, volumes, IPC namespace, and IP address.
- Pods are mortal and can die; replacements have different IDs and IP addresses.
- Pod immutability means you replace a Pod with a new one to apply changes.

Pod

Pod Manifest (cat k8s/flask-pod.yaml)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: flask-pod
  labels:
    app: flask-helloworld
spec:
  containers:
  - name: flask
    image: digitalocean/flask-helloworld:latest
    ports:
    - containerPort: 5000
```

Cont.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    name: nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

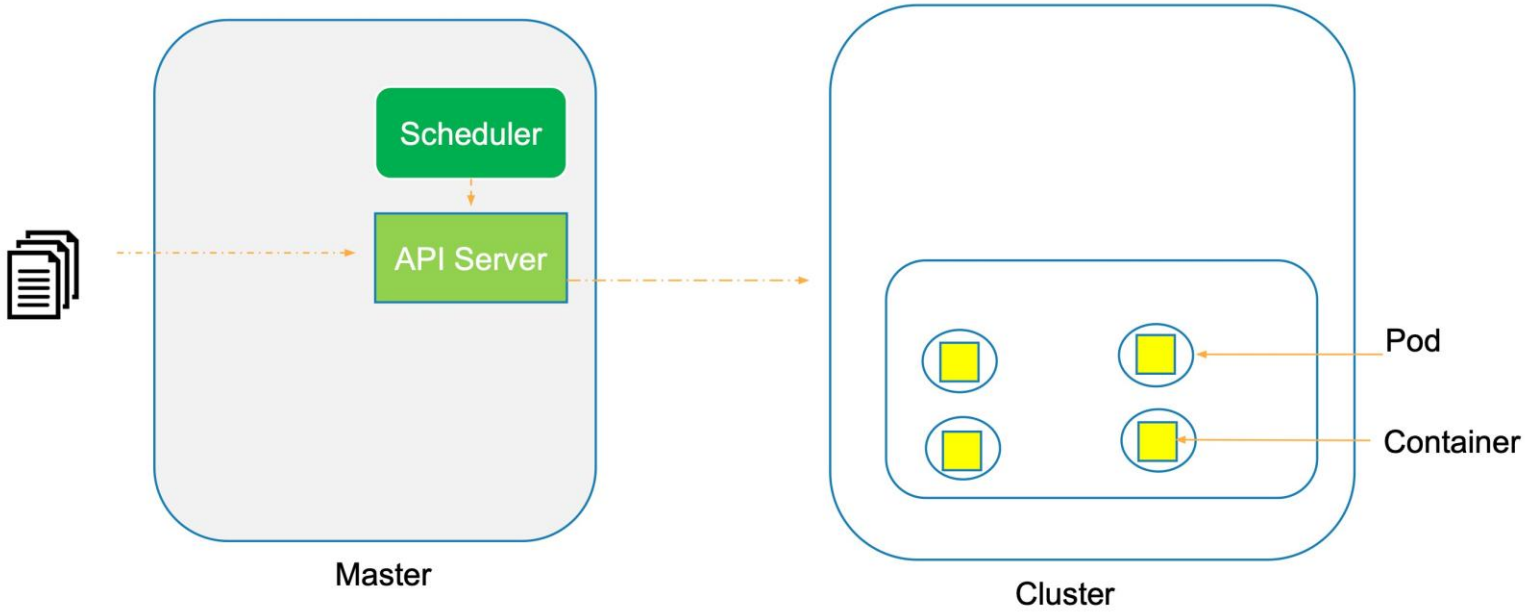
Create the pod as shown below:

```
$ kubectl create -f templates/pod.yaml
pod "nginx-pod" created
```

Get the list of pod:

```
$ kubectl get pods
NAME        READY     STATUS    RESTARTS   AGE
nginx-pod   1/1       Running   0          22s
```

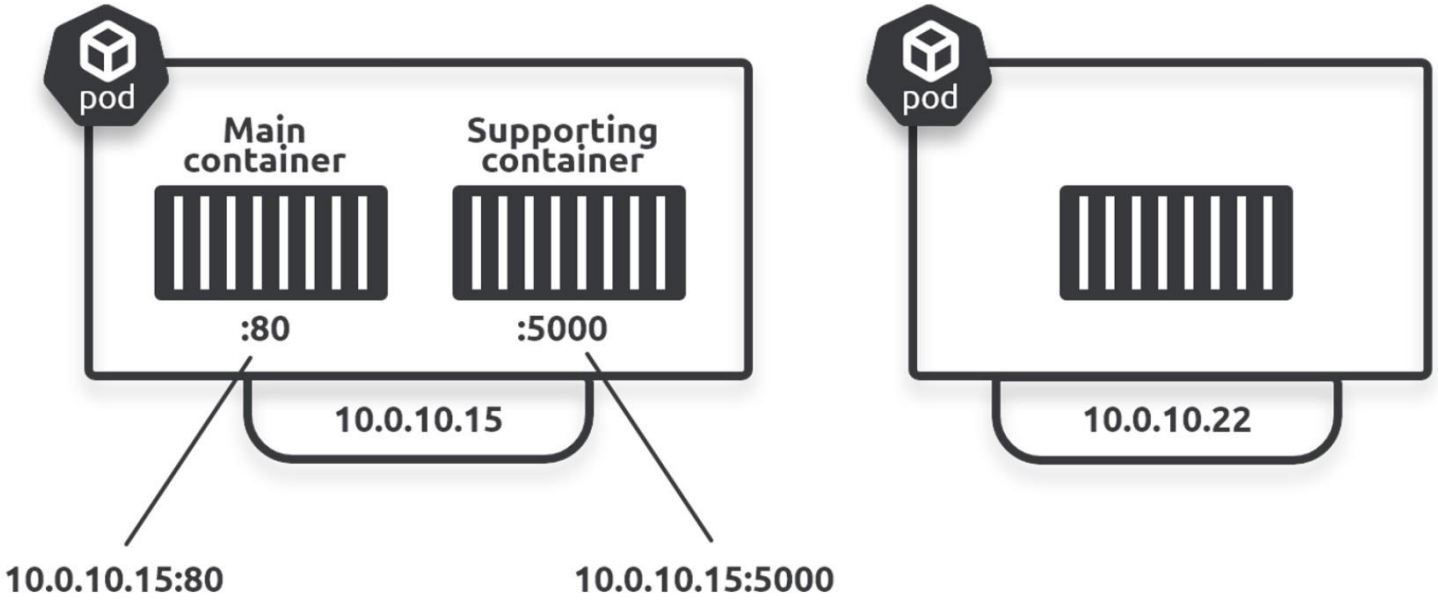
Pod Deployment



Pod

s

- Pods and shared networking –



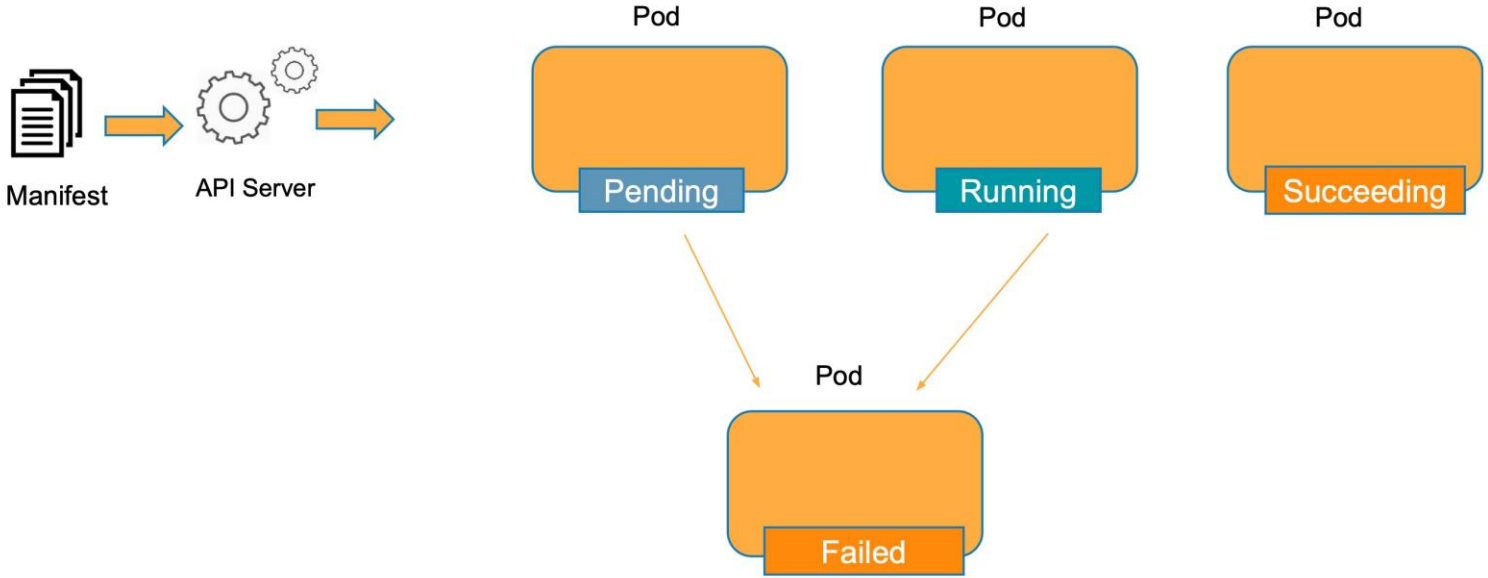
Cont.

- Sidecar pod – Kubernetes where you run a primary container (the main application) alongside one or more additional containers (sidecar containers) within the same Pod.

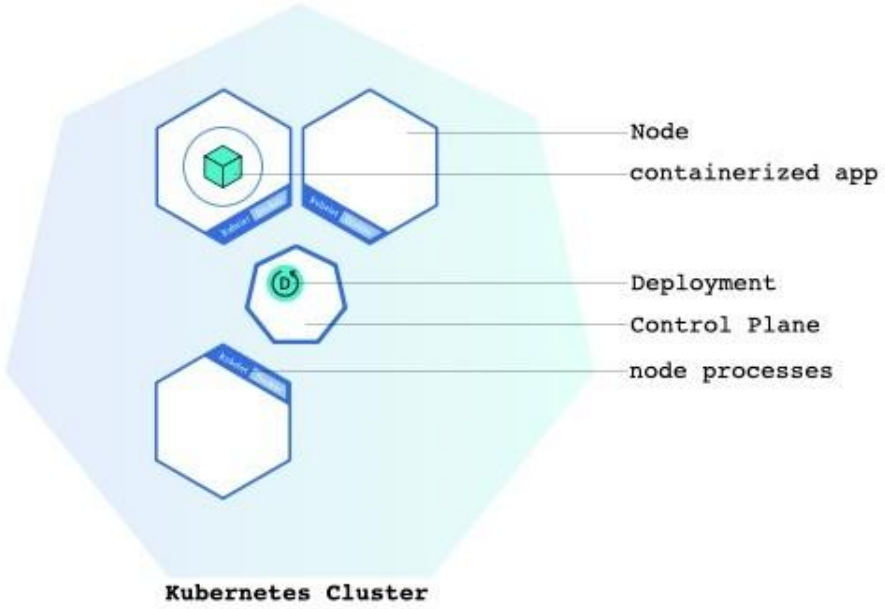
yaml Copy code

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: app-with-logging
spec:
  containers:
    - name: main-app
      image: my-main-app:latest
      ports:
        - containerPort: 80
    - name: logging-sidecar
      image: log-collector:latest
```

Lifecycle of Pod



Deployment



- Higher-level controllers like Deployments, DaemonSets, and StatefulSets are commonly used to deploy Pods.
- Deployments, for instance, enhance Pods with features like self-healing, scaling, rollouts, and rollbacks.
- These controllers operate as watch loops, ensuring the cluster's observed state aligns with the desired state.

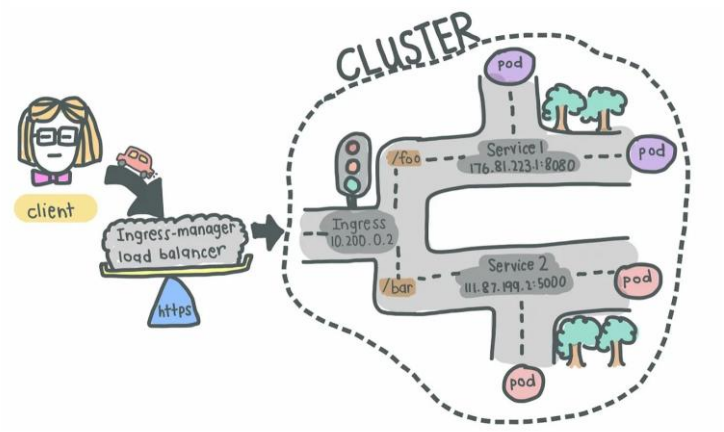
Namespaces

- An abstraction that allows you to divide a cluster into multiple scoped “virtual clusters”
  - E.g., Each team gets its own Namespace with associated resource quota
- Primary mechanism for scoping and limiting access
- Kubernetes usually starts with 3 Namespaces by default
  - Default
  - kube-system
  - kube-public

Creating a Namespace

- List namespaces with kubectl:
  - kubectl get namespaces
  - kubectl get ns
- Create your own:
  - kubectl create ns flask
- Specify a namespace with kubectl:
  - kubectl -n flask get all

Services



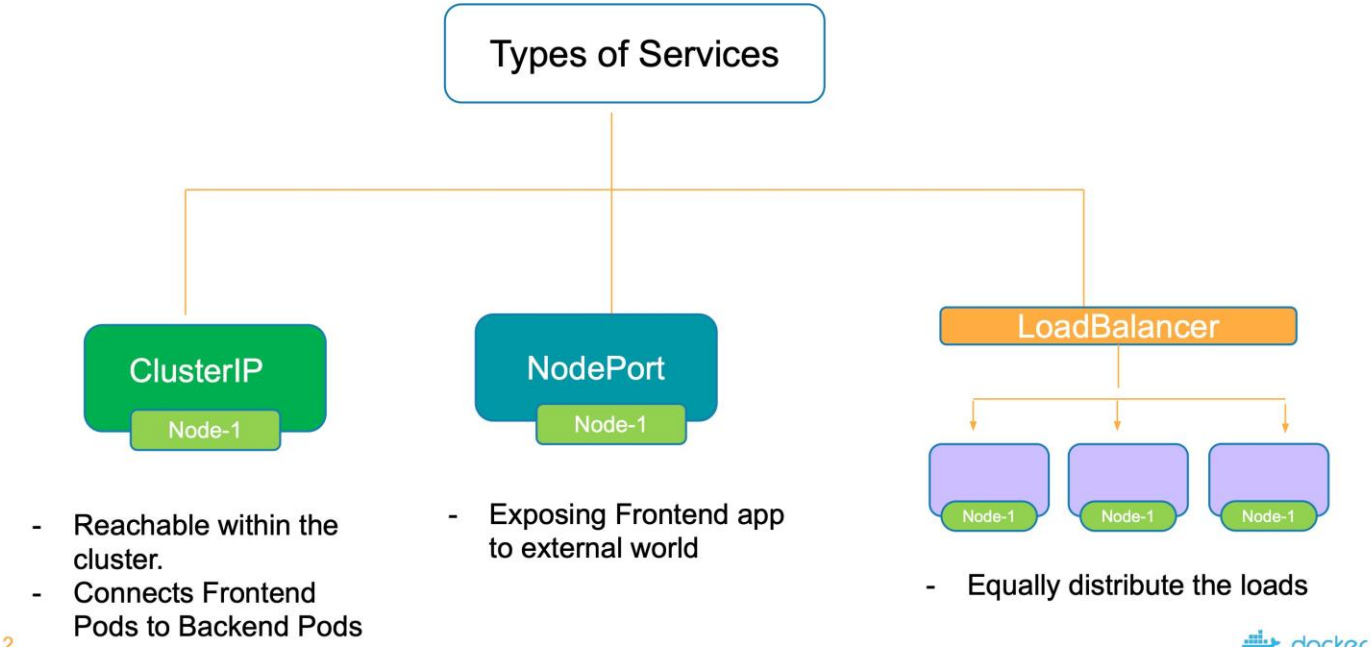
How Ingress works with Ingress Controller (Illustrated by Author)

- Imagine that you have been asked to deploy web app
- How does this frontend web app expose to outside world?
- How do front end app connected to backend database?
- How do we resolve Pod IP changes, when they die?

Cont.

- Services provide reliable networking for Pods.
- They have stable DNS names, IP addresses, and ports.
- Automatically update themselves as Pods come and go.
- Act as TCP and UDP load balancers, ensuring consistent access to Pods.
- For application-layer routing, Ingress is used.

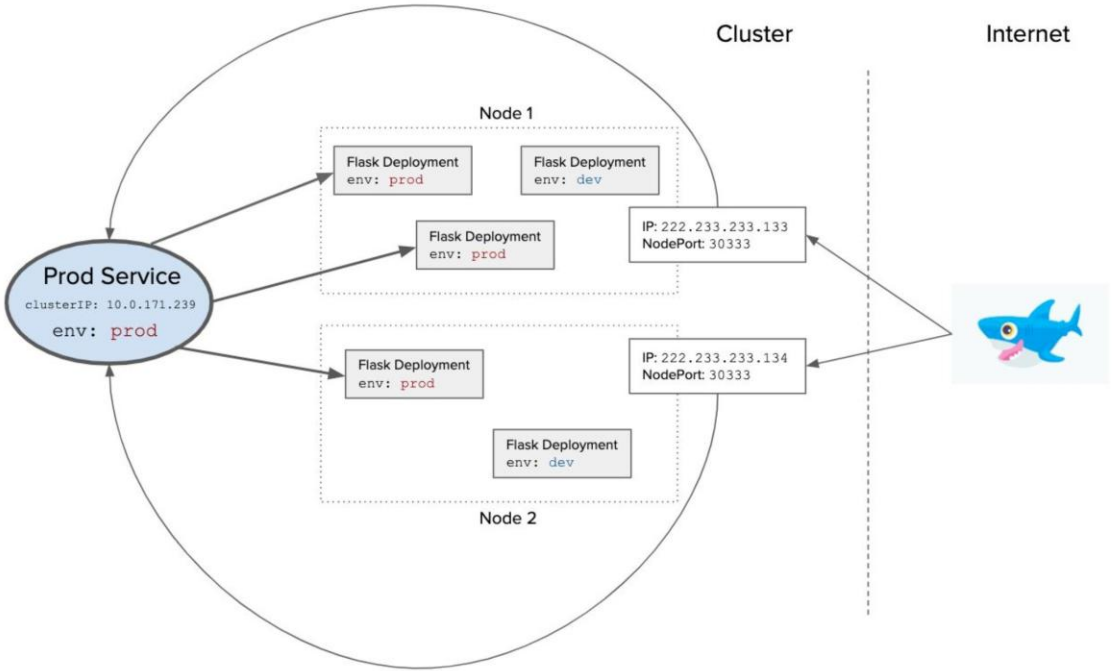
Types of Service



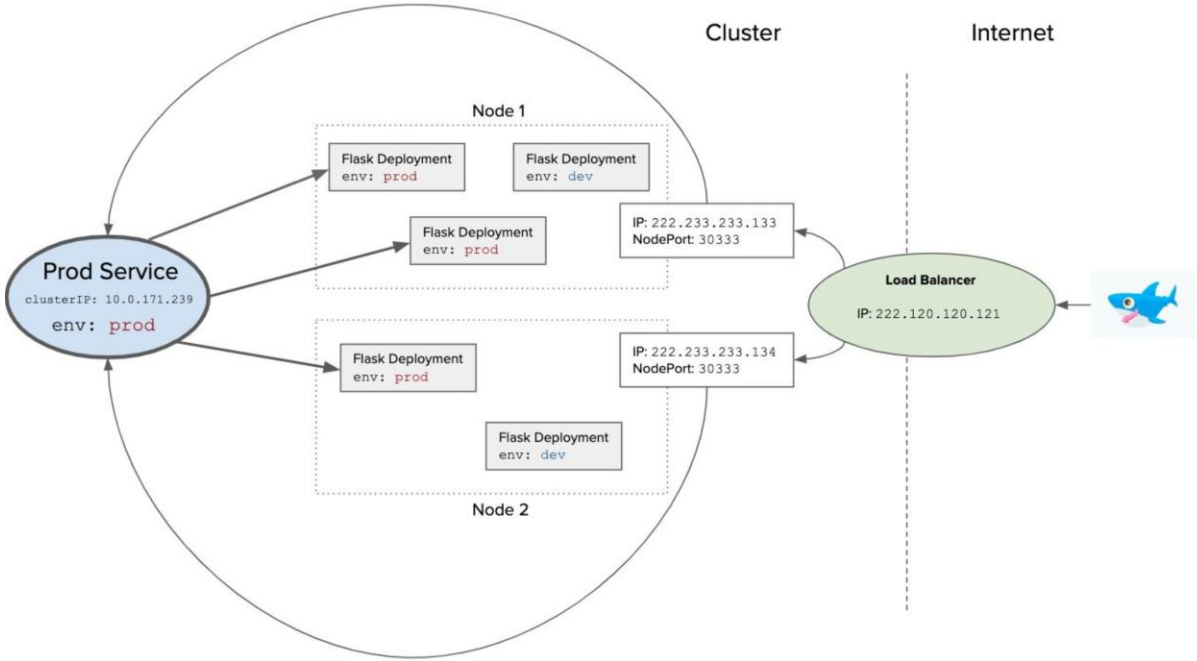
Cont.

- ClusterIP
  - Expose the service on a Cluster-internal IP
- NodePort
  - Expose the service on each Node’s IP at a static port (“NodePort”)
- LoadBalancer
  - Create an external LoadBalancer which routes requests to Nodeport & ClusterIP services
- Aside: Ingress Controllers

NodePort Service



LoadBalancer Service



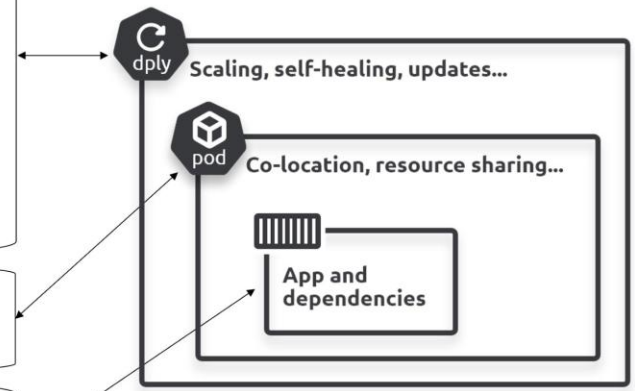
digitalocean.com

Kubernetes Services and Deployments

- K8s Deployments –
  - Kubernetes offers several controllers that augment Pods with important capabilities. The Deployment controller is specifically designed for stateless apps.
  - Augment Pods with self-healing, scalability, rolling updates, and rollbacks.
  - Behind-the-scenes, Deployments use ReplicaSets to do most of the work with Pods.

Cont.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  replicas: 10
  selector:
    matchLabels:
      app: hello-world
  minReadySeconds: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-pod
        image: nigelpoulton/k8sbook:1.0
        ports:
        - containerPort: 8080
```



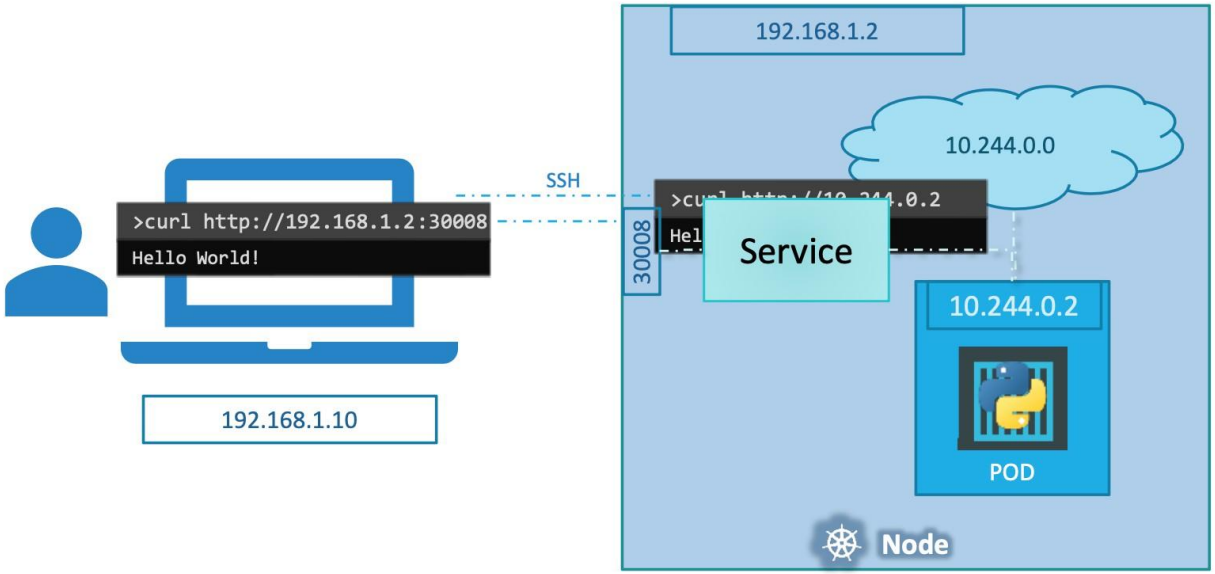
Cont.

- Deployments in Kubernetes manage declarative updates for Pods and ReplicaSets.
- They define a desired state, and the Deployment Controller ensures the actual state matches it.
- Deployments can create new ReplicaSets or replace existing ones while managing resources.
- Creating a Deployment involves defining its desired state in a YAML manifest and applying it to the cluster.

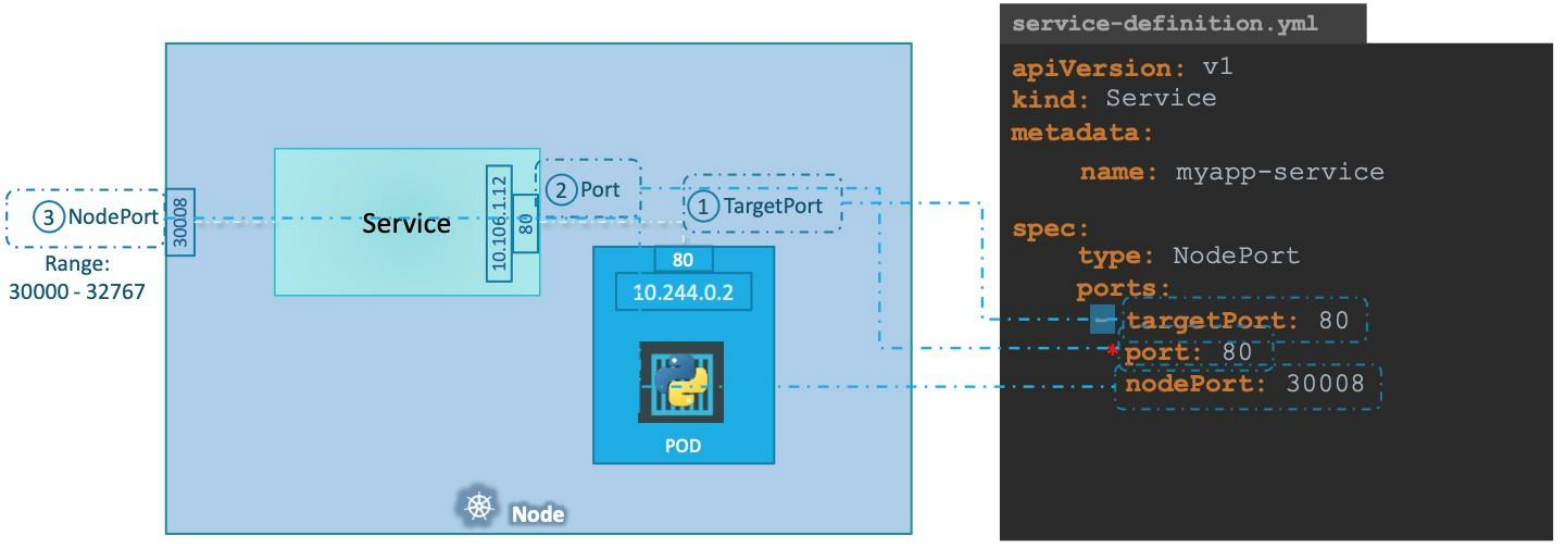
Cont.

- Typical use cases for Deployments include:
  - Rolling out a ReplicaSet and monitoring the rollout's success.
  - Updating the PodTemplateSpec to create a new ReplicaSet and manage Pod transitions.
  - Rolling back to a previous Deployment revision if needed.
  - Scaling up to handle more load.
  - Pausing rollouts for multiple fixes and resuming for a new rollout.
  - Using Deployment status as an indicator of a stuck rollout.
  - Cleaning up unnecessary older ReplicaSets.

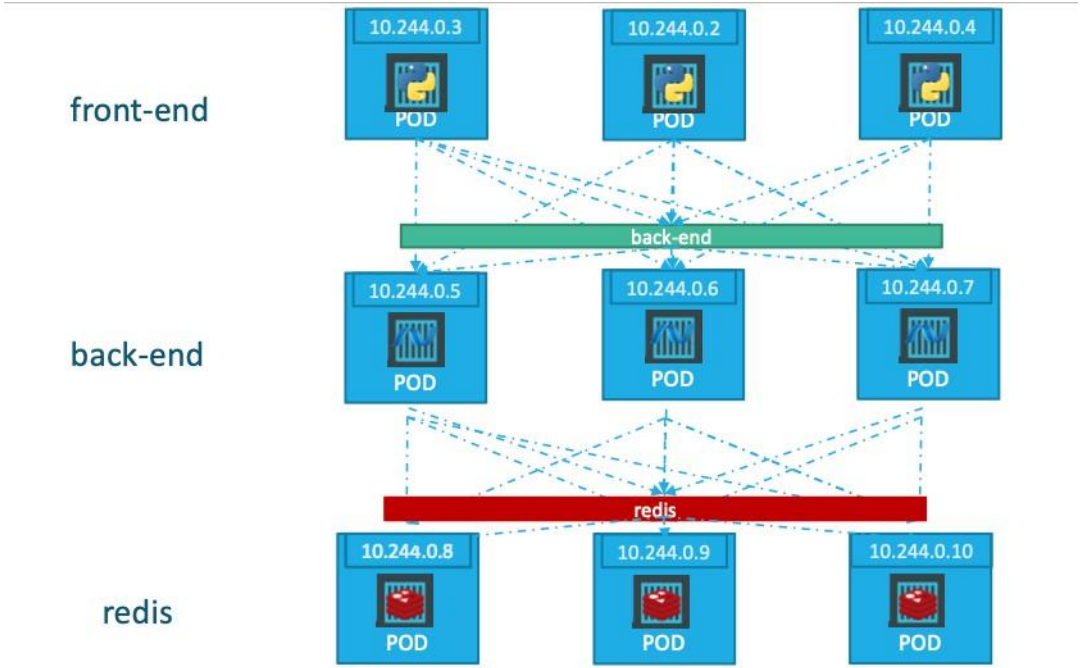
Kubernetes Services



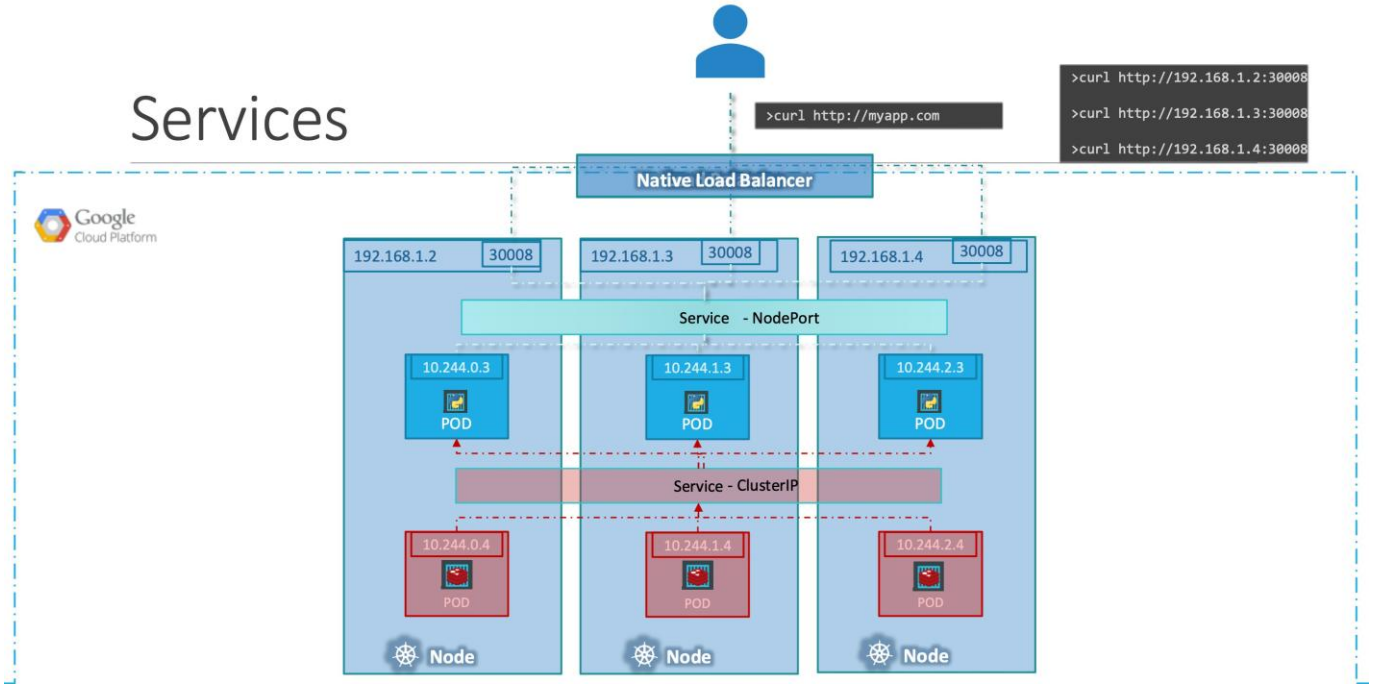
Cont.


3 NodePort
30008
Range:
30000 - 32767
Service
10.106.1.12
80
2 Port
1 TargetPort
80
10.244.0.2
POD
Node
service-definition.yml
apiVersion: v1
kind: Service
metadata:
name: myapp-service
spec:
type: NodePort
ports:
- targetPort: 80
port: 80
nodePort: 30008

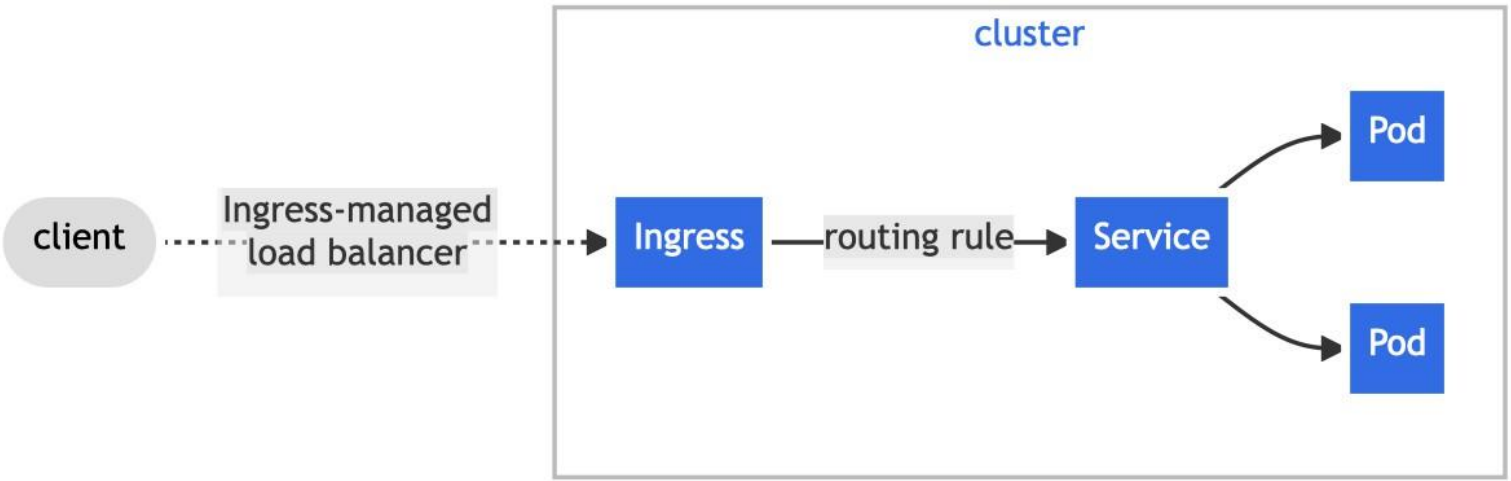
Cont.


front-end
back-end
redis
10.244.0.3
10.244.0.2
10.244.0.4
POD
POD
POD
back-end
10.244.0.5
10.244.0.6
10.244.0.7
POD
POD
POD
redis
10.244.0.8
10.244.0.9
10.244.0.10
POD
POD
POD

Cont.


Services
>curl http://myapp.com
>curl http://192.168.1.2:30008
>curl http://192.168.1.3:30008
>curl http://192.168.1.4:30008
Native Load Balancer
Google Cloud Platform
192.168.1.2
30008
192.168.1.3
30008
192.168.1.4
30008
Service - NodePort
10.244.0.3
POD
10.244.1.3
POD
10.244.2.3
POD
Service - ClusterIP
10.244.0.4
POD
10.244.1.4
POD
10.244.2.4
POD
Node
Node
Node

Ingress


cluster
client
Ingress-managed
load balancer
Ingress
routing rule
Service
Pod
Pod

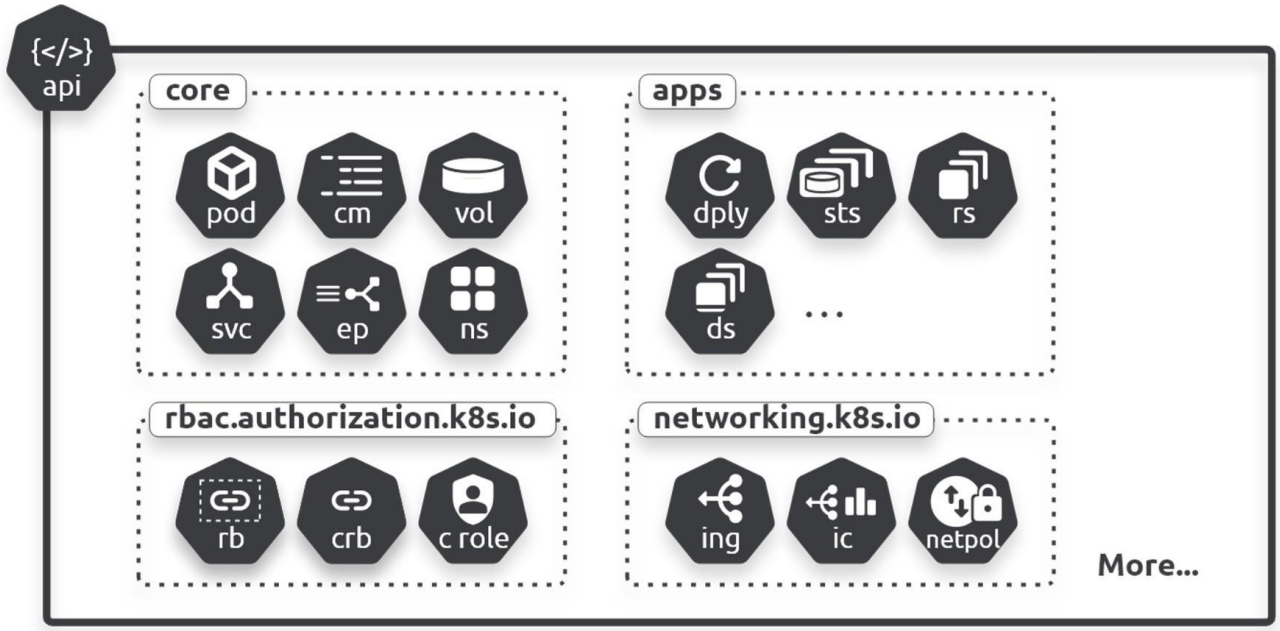
The Kubernetes API and Custom Resources

- **API is Like a Control Panel:** Think of the Kubernetes API like a control panel for managing your cluster.
- **Access Point:** It's the way you talk to your Kubernetes cluster, both from inside and outside.
- **Actions Through HTTP Verbs:** You use HTTP verbs (like GET, POST, PUT, PATCH, DELETE) to perform actions on your cluster via the API.

Cont.

- **Resources:** Everything in Kubernetes, like Pods, Services, and ConfigMaps, is represented as a "resource" accessible through the API.
- **Namespaces and Paths:** Resources are organized into namespaces, and you access them using paths like /api/v1 or /apis/apps/v1.

Cont.


{</>}
api
core
pod
cm
vol
svc
ep
ns
apps
dply
sts
rs
ds
...
rbac.authorization.k8s.io
rb
crb
c role
networking.k8s.io
ing
ic
netpol
More...

Cont.

- What Are They? Custom resources are like adding new buttons to your control panel.
- Extend Kubernetes: They allow you to extend Kubernetes to work with your own unique applications and services.
- Examples: You can create custom resources like "Database" or "GameServer" that Kubernetes didn't originally know about.

Cont.

- CustomResourceDefinitions (CRDs): You define what your custom resources look like using CRDs.
- Kubernetes Understands: Once you define a custom resource, Kubernetes understands how to create, update, and delete instances of it, just like its built-in resources.
- Easy Management: It simplifies the management of your custom applications because Kubernetes knows how to handle them once you've defined the rules.

StatefulSet

- **Workload Management:**
  - StatefulSet is a Kubernetes workload API object designed for managing stateful applications.
- **Pod Deployment and Scaling:**
  - It oversees the deployment and scaling of a group of Pods.
  - Provides guarantees regarding the ordering and uniqueness of these Pods.
- **Container Specification:**
  - Like Deployments, StatefulSets manage Pods based on an identical container specification (container image, resource requirements, etc.).

Cont.

- **Sticky Identity:**
  - StatefulSets maintain a "sticky" identity for each of their Pods.
  - While Pods within a StatefulSet are created from the same specification, they are not interchangeable.
  - Each Pod has a persistent identifier that remains the same even when rescheduled.
- **Storage and Persistence:**
  - StatefulSets are well-suited for workloads requiring storage volumes and data persistence.
  - Although individual Pods may fail, the persistent Pod identifiers simplify matching volumes to new Pods that replace failed ones.

DaemonSet

- **DaemonSet Functionality:**
  - A DaemonSet ensures that all or a subset of nodes in a Kubernetes cluster run a copy of a specific Pod.
  - It dynamically manages Pods, adding them to new nodes as they join the cluster and removing them when nodes are removed.
- **Garbage Collection:**
  - When a node is removed or a DaemonSet is deleted, it triggers the garbage collection of the Pods it created, ensuring resources are properly cleaned up.

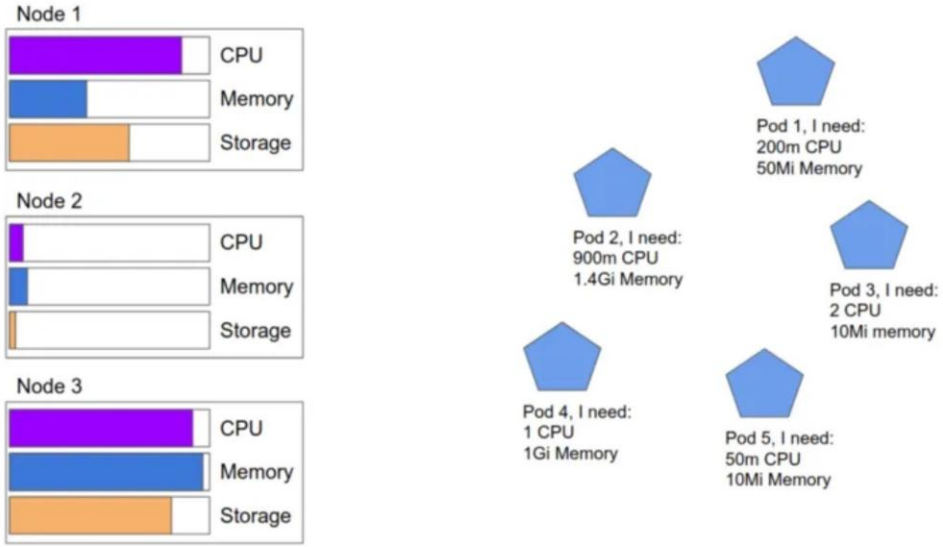
Cont.

- **Typical Use Cases:**
  - DaemonSets are commonly used for running system-level daemons or agents on every node in the cluster.
  - Examples include cluster storage daemons, log collection daemons, and node monitoring daemons.
- **Simple vs. Complex Configurations:**
  - In simpler cases, one DaemonSet is used to cover all nodes for a specific type of daemon.
  - More complex setups may involve multiple DaemonSets for the same type of daemon, each with different configurations (flags, resource requests) tailored to different hardware types.

Jobs

- Task Automation: Jobs automate tasks in Kubernetes.
- Retry Until Success: They keep retrying tasks until a certain number of successful completions are achieved.
- Clean-Up: Deleting a Job cleans up the Pods it created.
- Reliability: Jobs ensure tasks run reliably, even if Pods fail.
- Parallel Execution: You can use Jobs to run multiple tasks in parallel.
- Scheduled Jobs: For scheduled tasks, use CronJobs.

Manging Resources

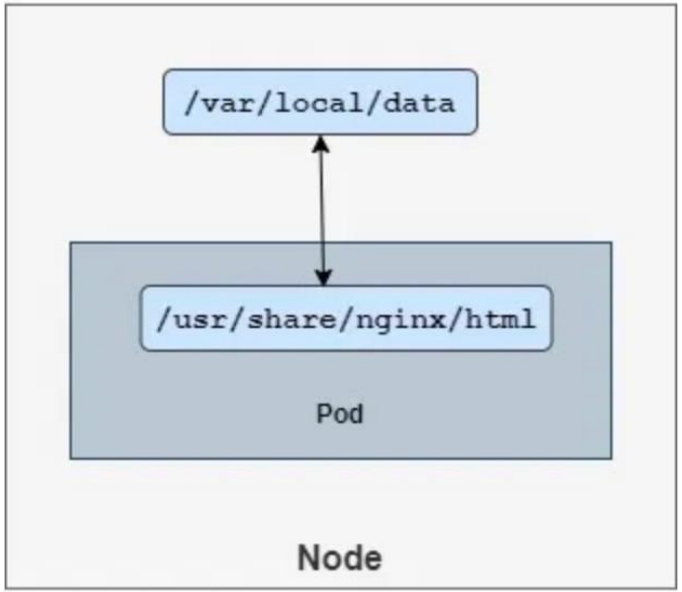


Pods requests and nodes resources

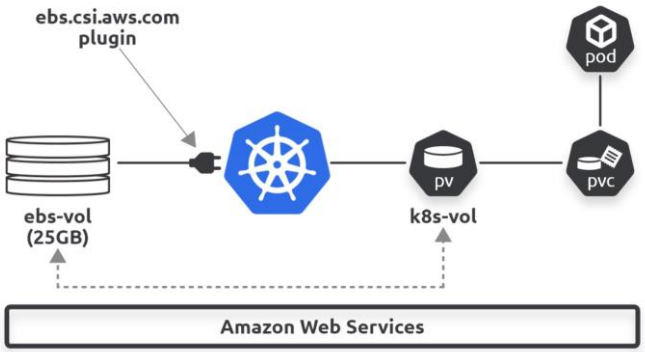
Cont.

- Kubernetes manages various resources like CPU, memory, and more.
- Containers are based on Linux namespaces for isolation.
- Control groups (Cgroups) limit resource consumption.
- Kubernetes schedules pods based on resource requests.
- Resource limits prevent container overloads.
- Kubernetes uses QoS classes for pod priorities.

Kubernetes Storage



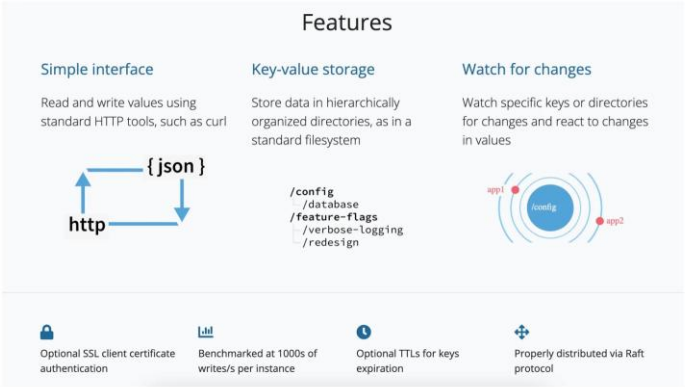
Cont.



- The architecture involves storage providers on the left, which can be traditional enterprise storage or cloud storage services.
- The plugin layer in the middle acts as a connector between external storage and Kubernetes. Modern plugins use the Container Storage Interface (CSI).
- On the right is the Kubernetes persistent volume subsystem, consisting of API objects like PersistentVolumes (PV), PersistentVolumeClaims (PVC), and StorageClasses (SC).

Etcd

- etcd: A distributed, consistent key-value store.
- Use in Kubernetes: Stores cluster configuration and metadata.
- Example: Storing a service endpoint: /myapp/config/service-endpoint - 192.168.0.100:8080.
- API: Offers an API for reading, writing, and watching data.



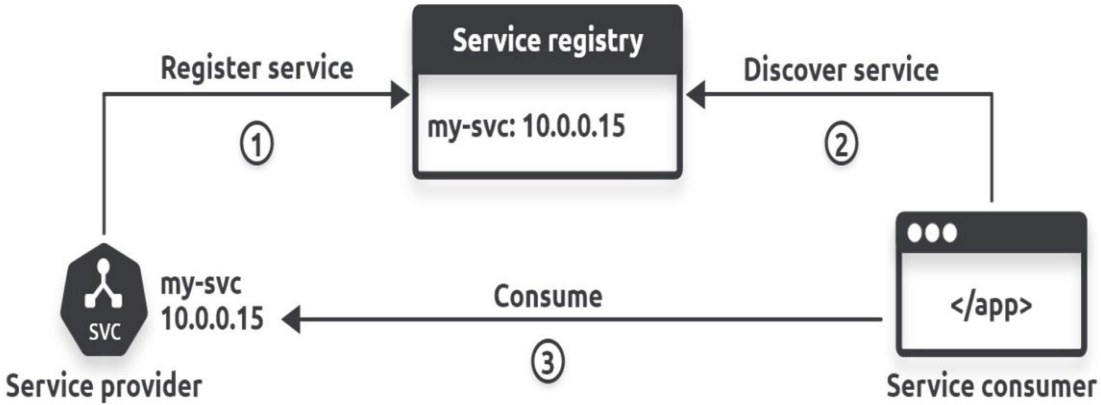
Cont.

- High Availability: Can run in a cluster for fault tolerance.
- Data Types: Supports both ephemeral and persistent data.
- Kubernetes Recovery: Vital for recovering cluster state after failures.
- Security: Provides authentication and authorization for data access.

Service Discovery and Networking

- Service Discovery: Kubernetes provides a built-in mechanism for service discovery, allowing applications to locate and communicate with each other within the cluster.

- DNS-Based Service Names: Kubernetes assigns DNS names to services automatically. For example, if you have a service named "web-service" in the "myapp" namespace, it can be reached at "web-service.myapp.svc.cluster.local."



Cont.

- Pod-to-Pod Communication: Pods can communicate with each other using these DNS names, regardless of the nodes they are running on. This simplifies the configuration as pods can rely on service names rather than explicit IP addresses.
- Load Balancing: Services automatically distribute incoming traffic across the pods backing the service. This load balancing ensures high availability and fault tolerance.

Cont.

- Example: Imagine you have multiple pods running a web application. Instead of keeping track of each pod's IP address, you can simply use the service name (e.g., "web-service") to access the application. Kubernetes will handle routing the requests to the appropriate pods.

Kubernetes Security Practices:

- **Authentication**: Kubernetes supports various authentication methods, including client certificates, bearer tokens, and more. This ensures that only authenticated entities can access the cluster.
- **Authorization**: Kubernetes has Role-Based Access Control (RBAC) to define who can perform actions on resources. You can grant or restrict access at a fine-grained level.

Cont.

- **RBAC (Role-Based Access Control)**: RBAC allows you to define roles and role bindings, specifying what actions are permitted on which resources for different users or groups.
- **Pod Security Policies**: These policies define security constraints that pods must adhere to. For instance, you can restrict pods from running as the root user or accessing host namespaces.

Cont.

- **Network Policies:** Kubernetes Network Policies allow you to control the traffic between pods by defining rules for ingress and egress traffic. This enhances network security within the cluster.

Helm :a K8S “Package Manager”

- Tool for managing Kubernetes applications
- Think “apt-get” for Ubuntu / package managers
- Architecture
  - Helm (client)
  - Tiller (server, runs in the cluster)
- How it works
  - Charts
  - helm install stable/wordpress
- Sample apps: Wordpress, Prometheus, MySQL, Drupal, ...

Cont.

- Helm Charts: Helm packages Kubernetes resources into a convenient format called "Charts." These Charts include configurations, templates, and dependencies needed to deploy complex applications.
- Simplify Deployment: Helm simplifies the deployment of applications by providing a consistent and repeatable way to define, install, and upgrade applications on Kubernetes.

Cont.

- Example: Suppose you want to deploy a WordPress application. Instead of manually creating pods, services, and config files, you can use a pre-made Helm Chart for WordPress. This Chart contains all the necessary Kubernetes resources and configurations. You can install it with a single Helm command, making the deployment process more efficient and less error-prone.