Kubernetes

Adyanth Hosavalike

We need more than just packing and isolation

Scheduling: Where should my containers run?

Lifecycle and health: Keep my containers running despite failures

Discovery: Where are my containers now?

Monitoring: What's happening with my containers?

Auth{n,z}: Control who can do things to my containers

Aggregates: Compose sets of containers into jobs

Scaling: Making jobs bigger or smaller

...

Open Source Containers: Kubernetes

Greek for *"Helmsman"*; also the root of the word *"Governor"* and *"cybernetic"*

- Container orchestrator
- Builds on Docker containers
  - also supporting other container technologies
- Multiple cloud and bare-metal environments
- Supports existing OSS apps
  - cannot require apps becoming cloud-native
- Inspired and informed by Google's experiences and internal systems
- **100% Open source**, written in **Go**

Let users manage **applications**, not machines

Primary concepts

Container: A sealed application package (Docker)

Pod: A small group of tightly coupled Containers

Labels: Identifying metadata attached to objects

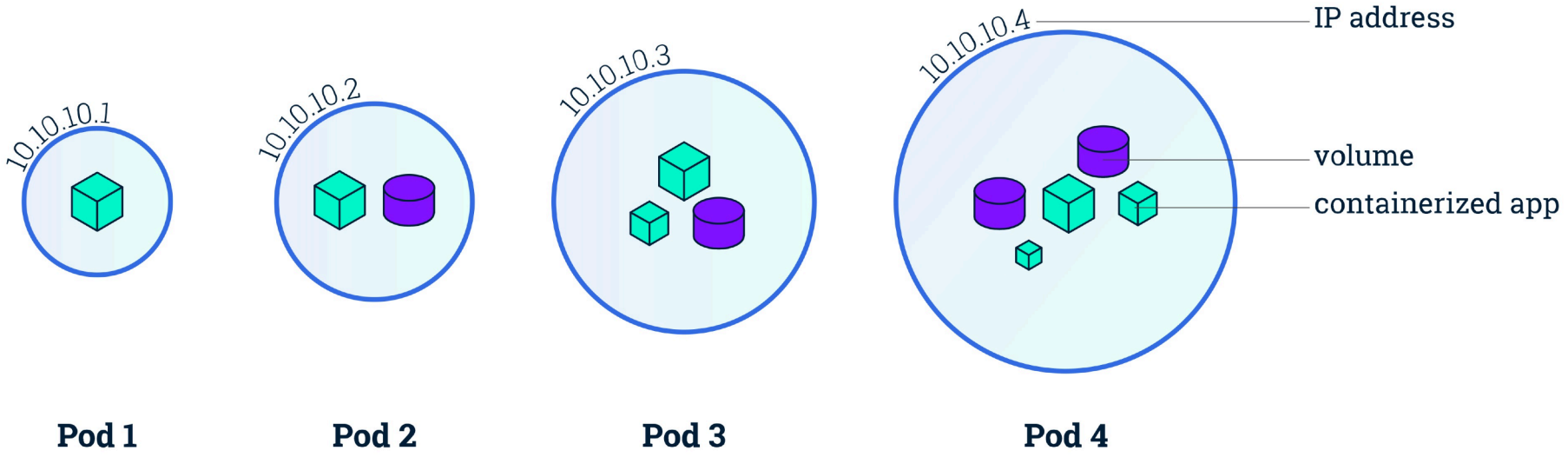
Selector: A query against labels, producing a set result

Controller: A reconciliation loop that drives current state towards desired state

Service: A set of pods that work together

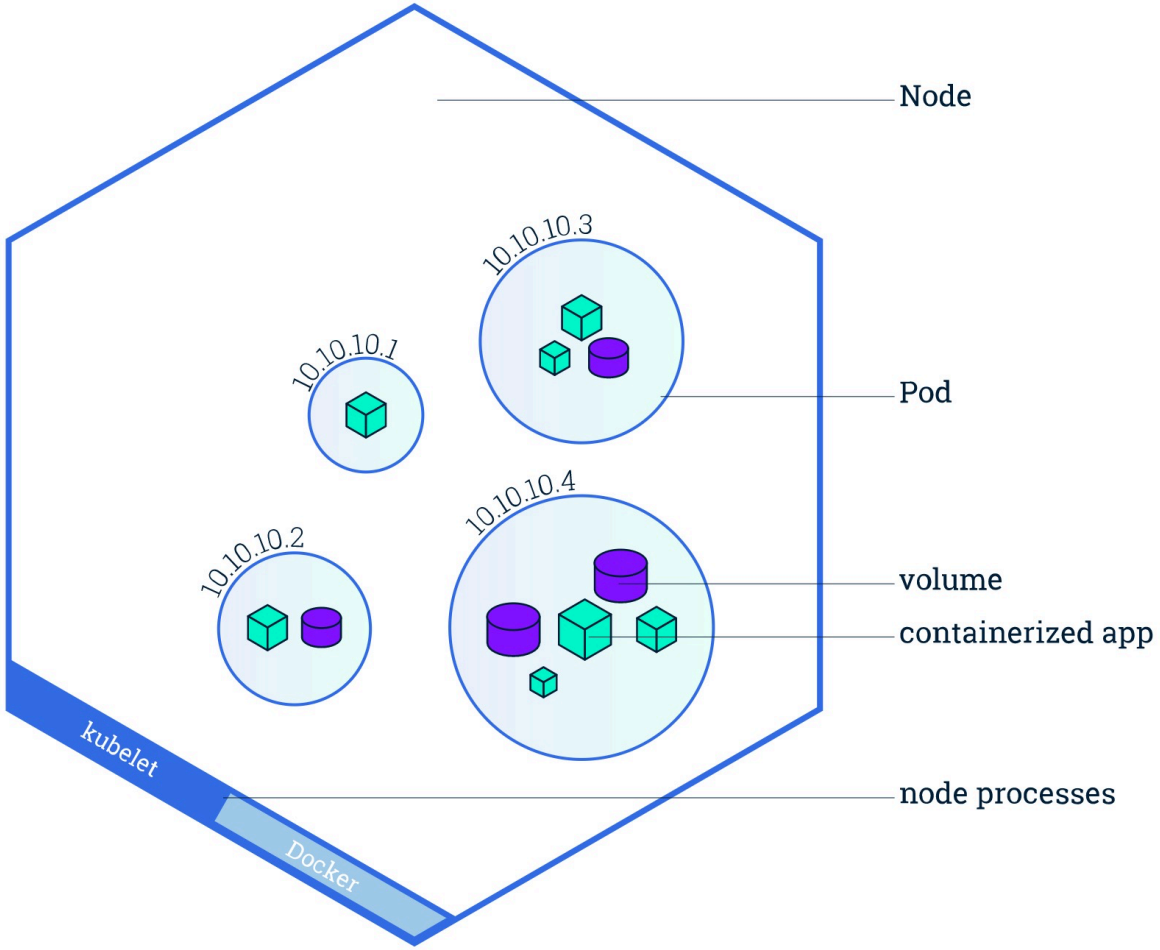
Pod

- a Kubernetes abstraction that represents a group of one or more application containers, and some shared resources for those containers
  - 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

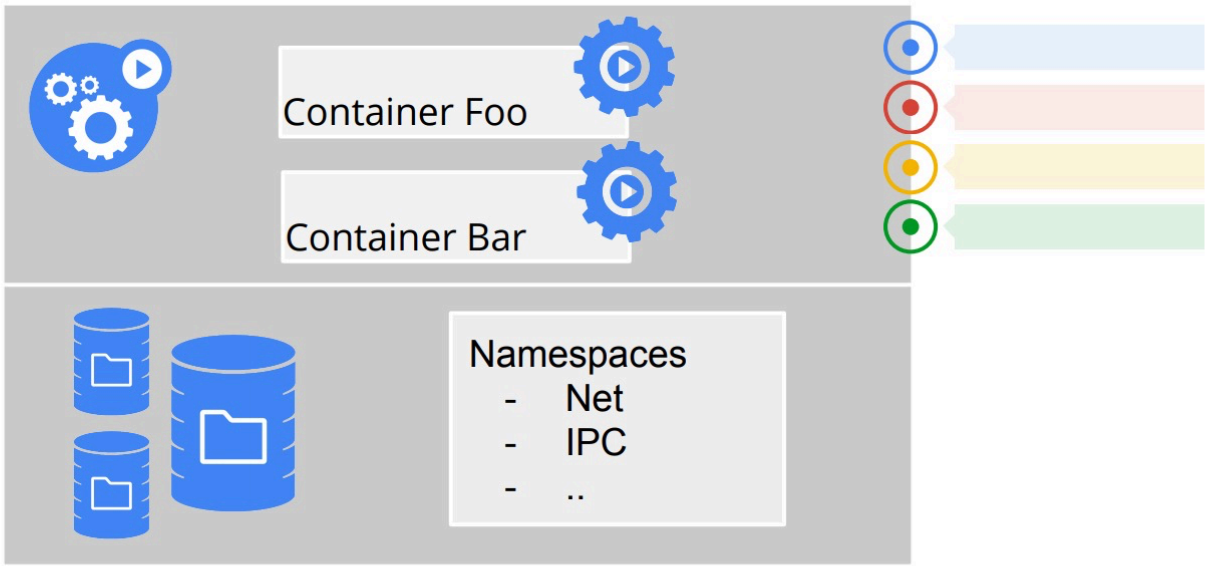


Node

- A node is a worker machine (either VM or physical machine)
- One pod runs on one node, one node can run multiple pods
- Nodes managed by control plane



Pods: Grouping containers



Why Pods instead of containers?

Labels

Arbitrary metadata

Attached to any API object

Generally represent **identity**

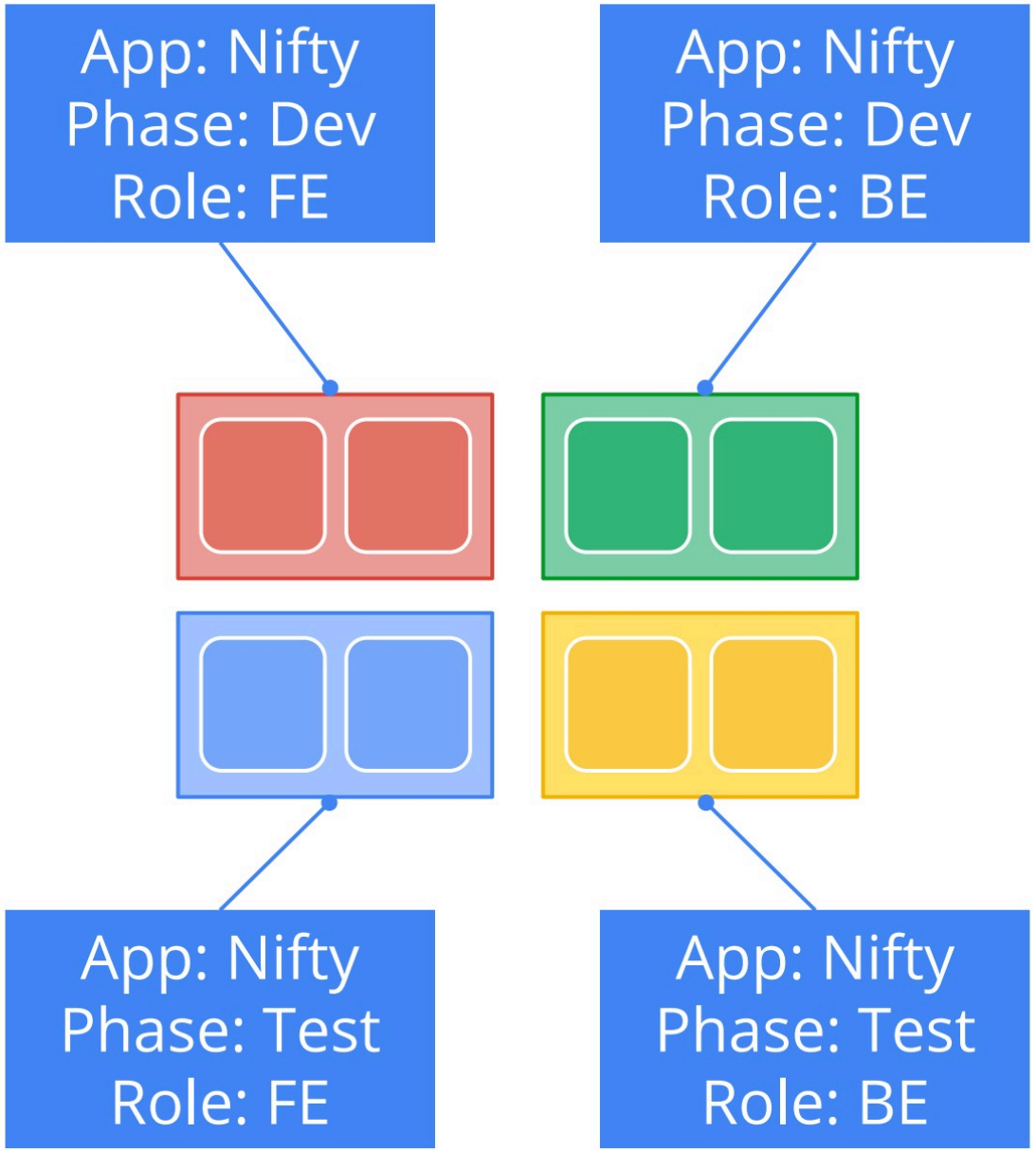
Queryable by **selectors**

- think SQL '*select ... where ...*'

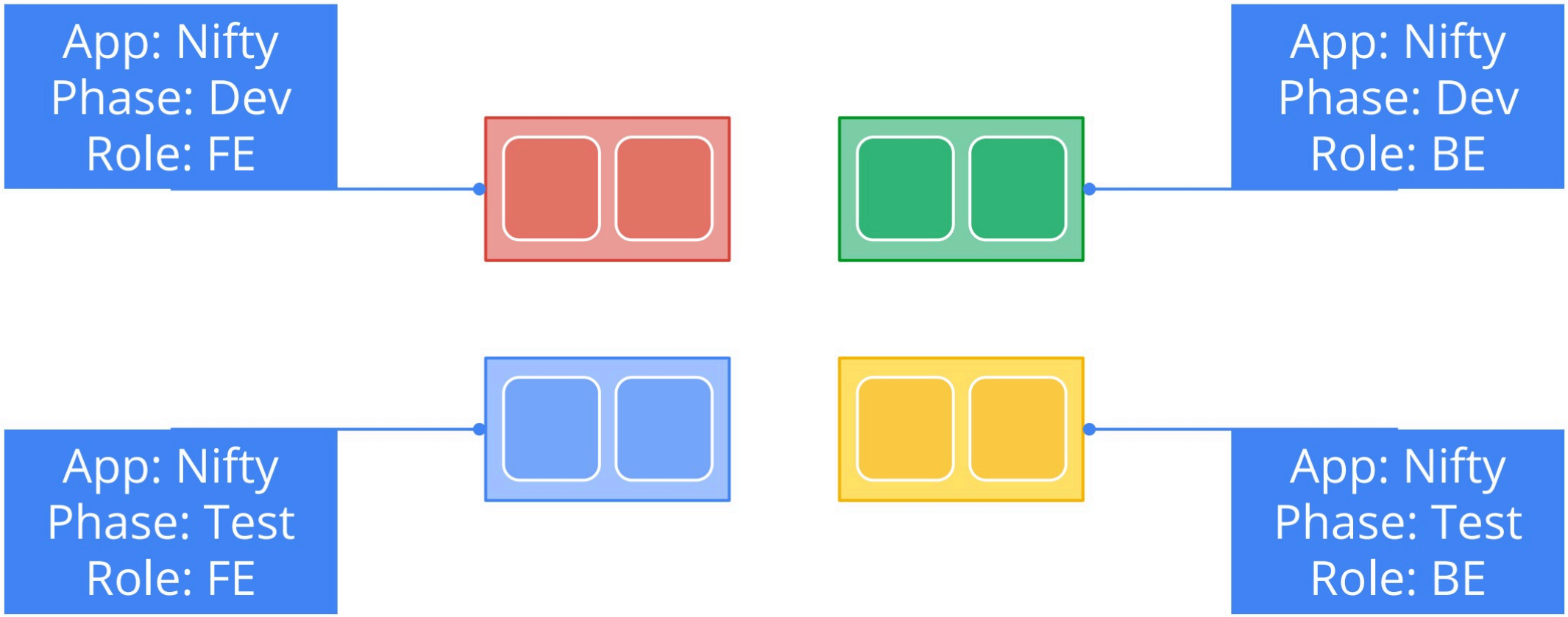
The **only** grouping mechanism

Use to determine which objects to apply an operation to

- pods under a ReplicationController
- pods in a Service
- capabilities of a node (scheduling constraints)



Selectors


App: Nifty
Phase: Dev
Role: FE
App: Nifty
Phase: Dev
Role: BE
App: Nifty
Phase: Test
Role: FE
App: Nifty
Phase: Test
Role: BE

Pod lifecycle

Once scheduled to a node, pods do not move

- restart policy means restart **in-place**

Pods can be observed *pending*, *running*, *succeeded*, or *failed*

- *failed* is **really** the end - no more restarts
- no complex state machine logic

Pods are **not rescheduled** by the scheduler or apiserver

- even if a node dies
- controllers are responsible for this
- keeps the scheduler **simple**

Apps should consider these rules

- Services hide this
- Makes pod-to-pod communication more formal

Persistent Volumes

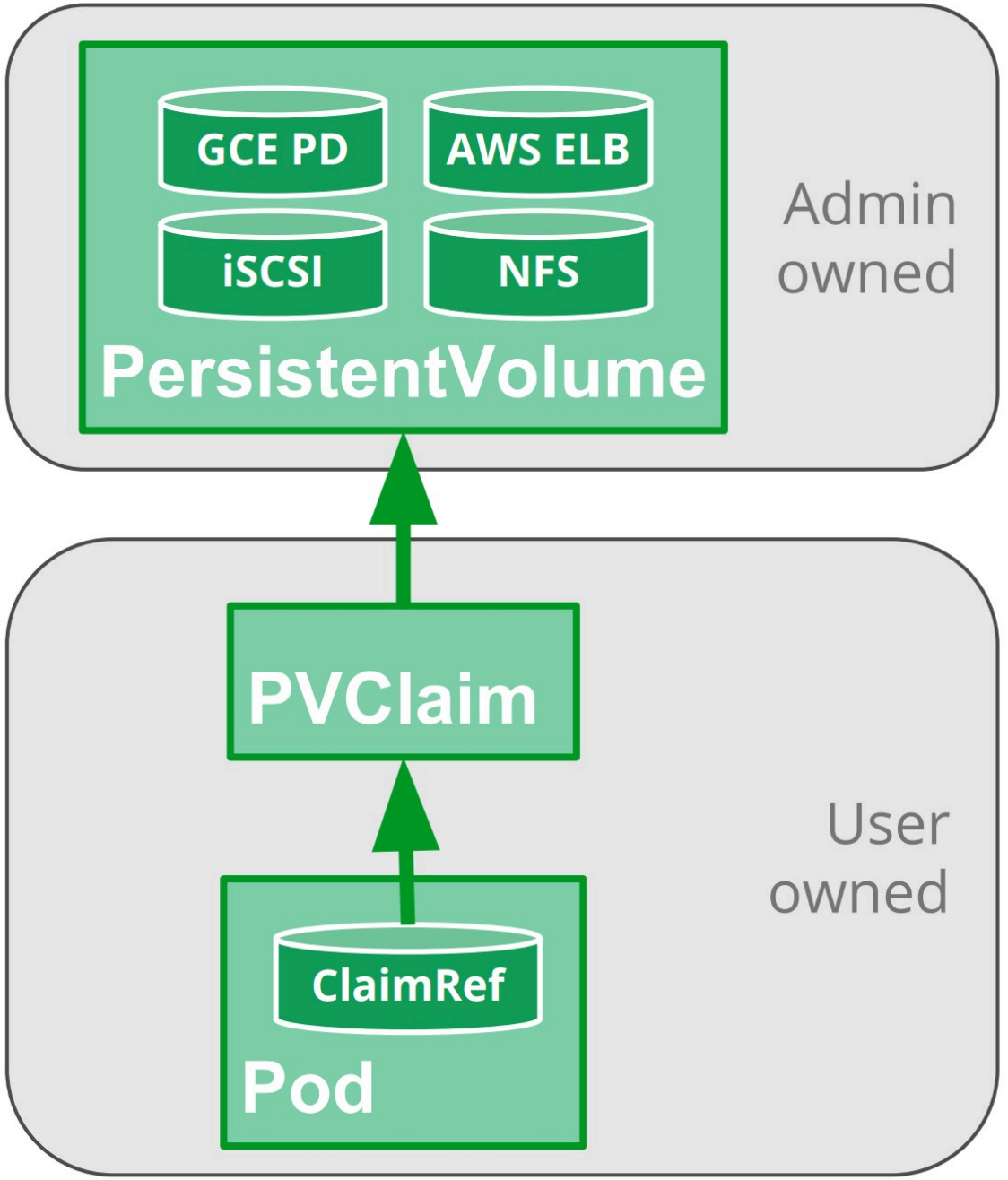
A higher-level abstraction - insulation from any one cloud environment

Admin provisions them, users claim them

Independent lifetime and fate

Can be handed-off between pods and lives until user is done with it

Dynamically “scheduled” and managed, like nodes and pods



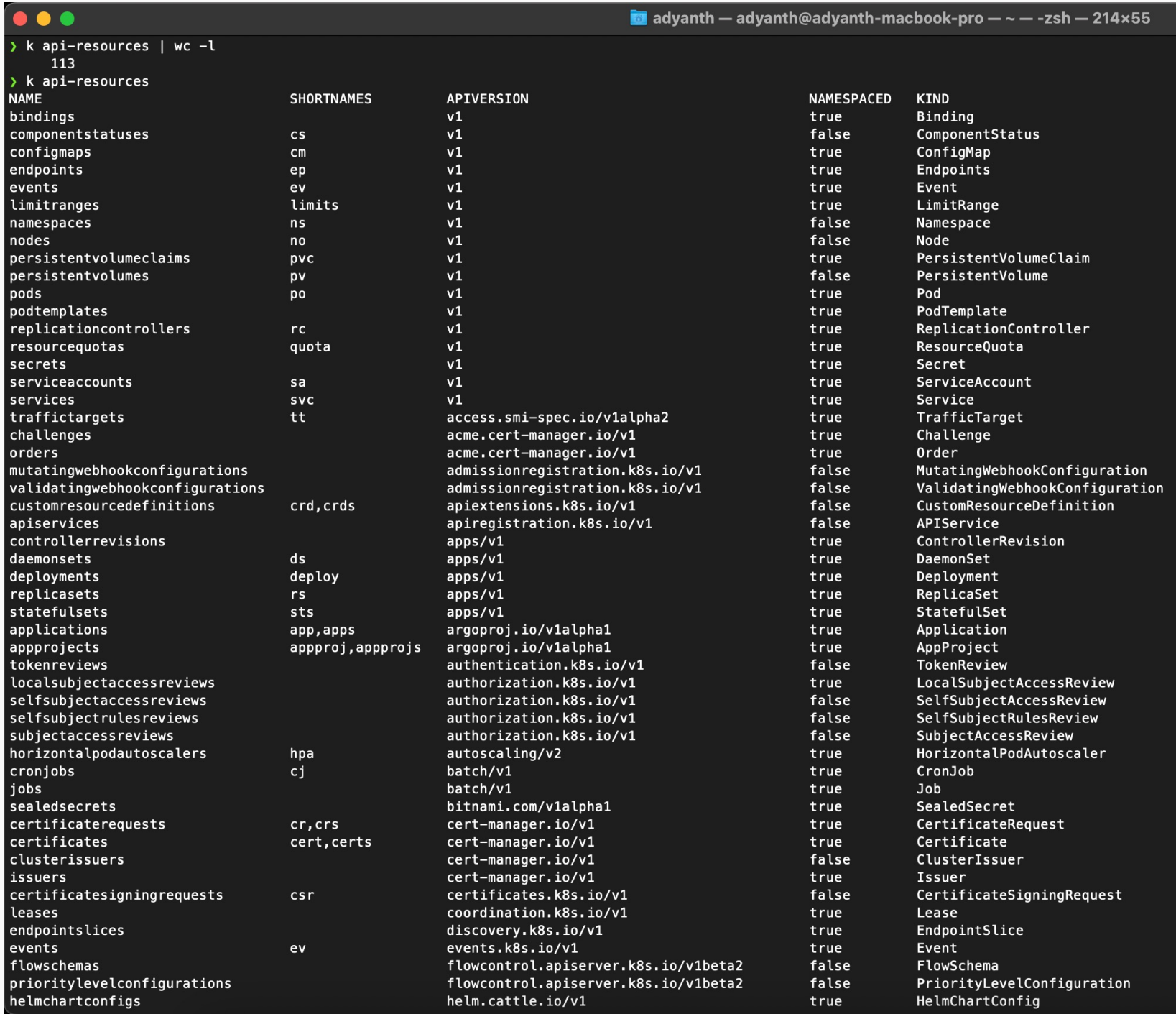
There are many API resources in Kubernetes.

Built in and custom resources

Input to the API server

Pods, Nodes, Deployments, Services, Jobs, CronJobs, ConfigMaps, Secrets, PersistentVolumes are all API resources

- kubectl api-resources



Run the Ghost micro-blogging platform

kubectl run ghost --image=ghost:0.9

kubectl expose deployment ghost --port=2368 --type=NodePort

kubectl get svc

kubectl get deploy

```
NAME                     READY     STATUS              RESTARTS   AGE
ghost-7cbd79df7d-6shhh   0/1       ContainerCreating   0          2s
ghost-7cbd79df7d-7vtjw   1/1       Running             0          1m
ghost-7cbd79df7d-fd7b9   1/1       Running             0          11m
```

kubectl edit deploy ghost (change the nr. of replicas to 3)

kubectl exec -it ghost-xxx bash

kubectl logs ghost-xxx

kubectl delete deploy ghost

- kubectl apply –f <filename>.yaml

Declarative syntax

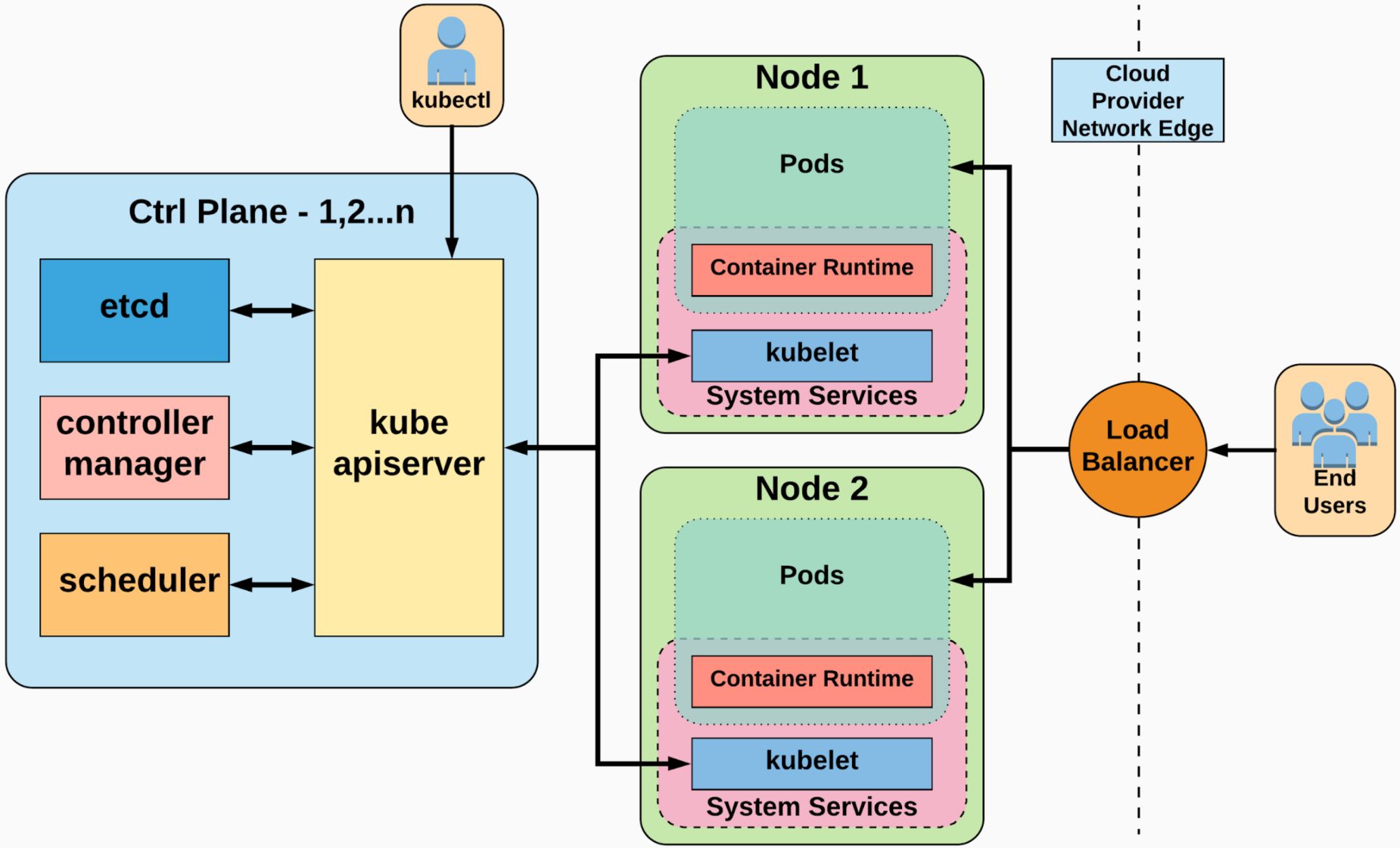
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.23.2-alpine
          volumeMounts:
            - name: config
              subPath: nginx.conf
              mountPath: /etc/nginx/conf.d/default.conf
            - name: data
              mountPath: /usr/share/nginx/html
            - name: certs
              mountPath: /usr/share/nginx/html/certs
          resources:
            requests:
              memory: "8Mi"
              cpu: "10m"
            limits:
              memory: "64Mi"
              cpu: "100m"
          ports:
            - containerPort: 80
      volumes:
        - name: config
          configMap:
            name: nginx
        - name: data
          nfs:
            path: /export/content
            server: nas.domain.lan
        - name: certs
          nfs:
            path: /export/certs
            server: nas.domain.lan
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: static
spec:
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: 80
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx
data:
  nginx.conf: |
    server {
        listen       80;
        server_name  localhost;
        location / {
            root   /usr/share/nginx/html;
            autoindex on;
            index  index.html index.htm;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
```

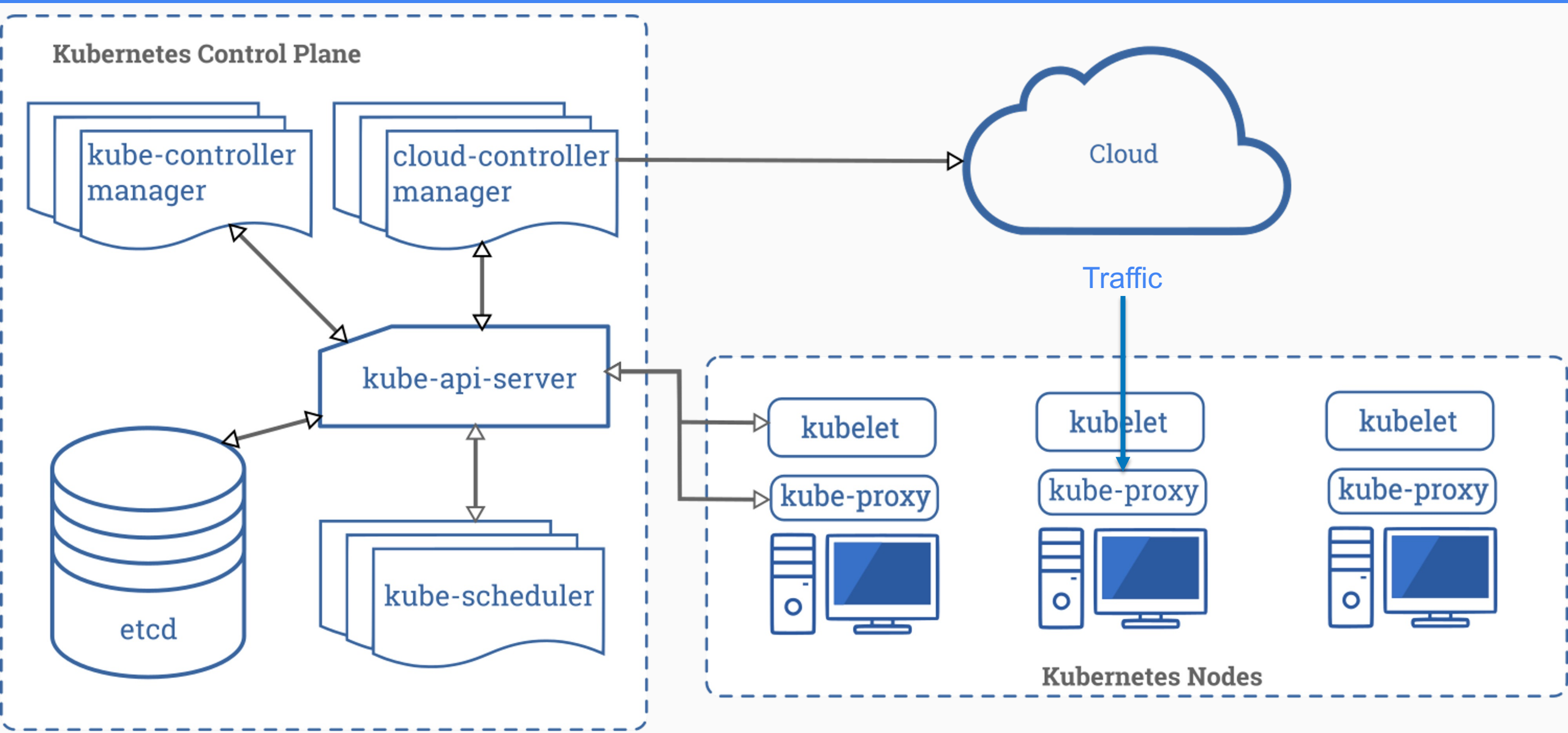
```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: static
  annotations:
    cert-manager.io/cluster-issuer: le
  labels:
    name: nginx
spec:
  rules:
    - host: static.domain.lan
      http: &http
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: static
                port:
                  number: 80
    - host: static.domain.site
      http: *http
  tls:
    - hosts:
        - static.domain.lan
        - static.domain.site
      secretName: static-cert
```

Kubernetes Architecture Overview



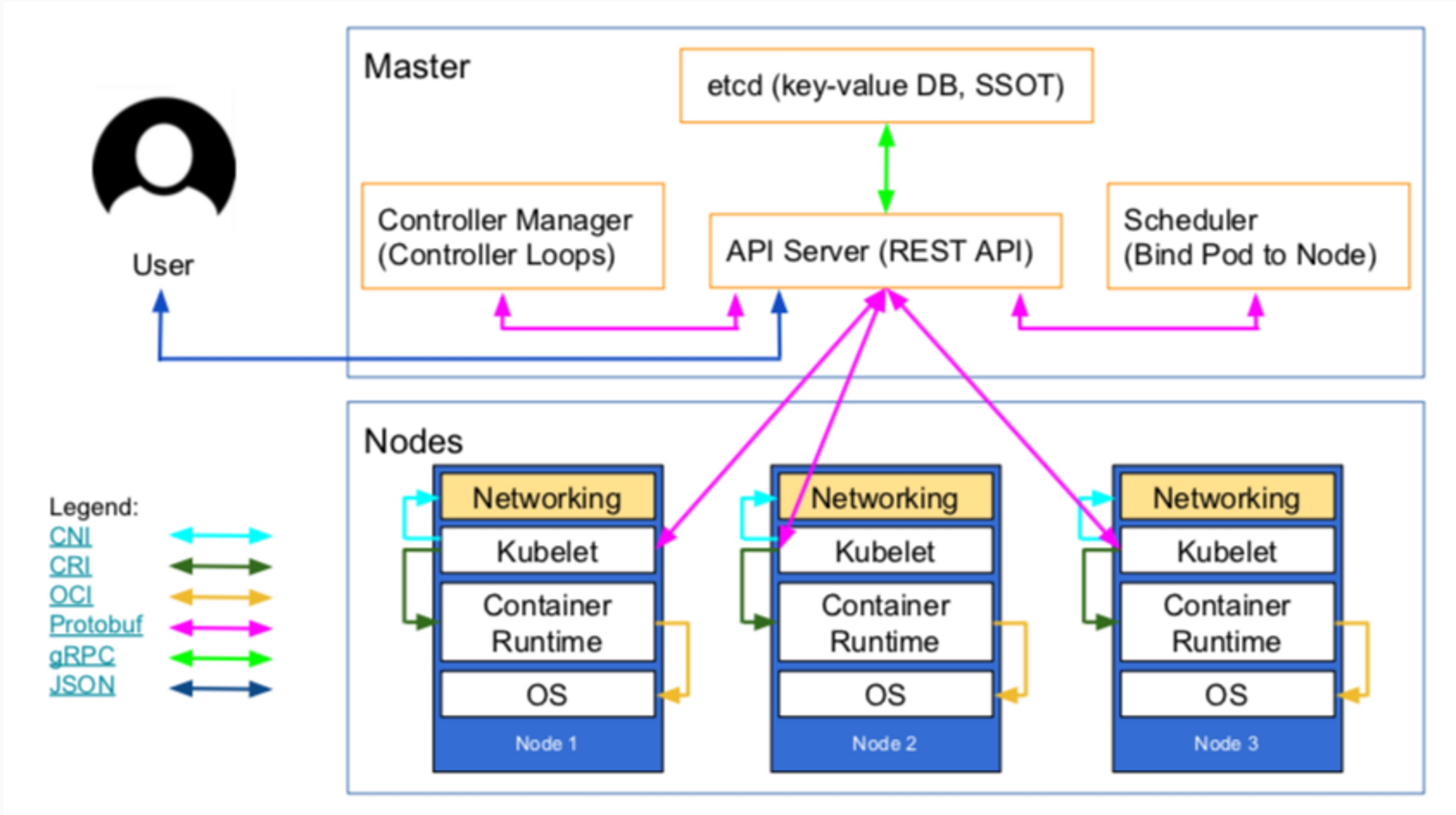
Source: Introduction to Kubernetes

Kubernetes Components



Source: https://kubernetes.io/docs/concepts/overview/components/#master-components

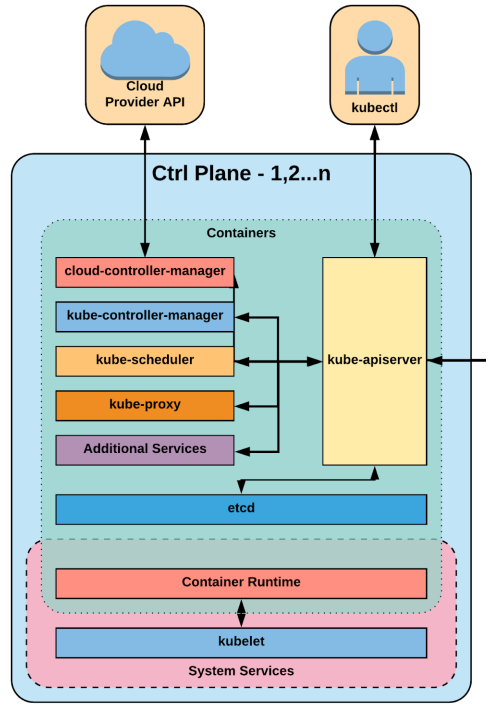
Kubernetes' High-Level Architecture Overview



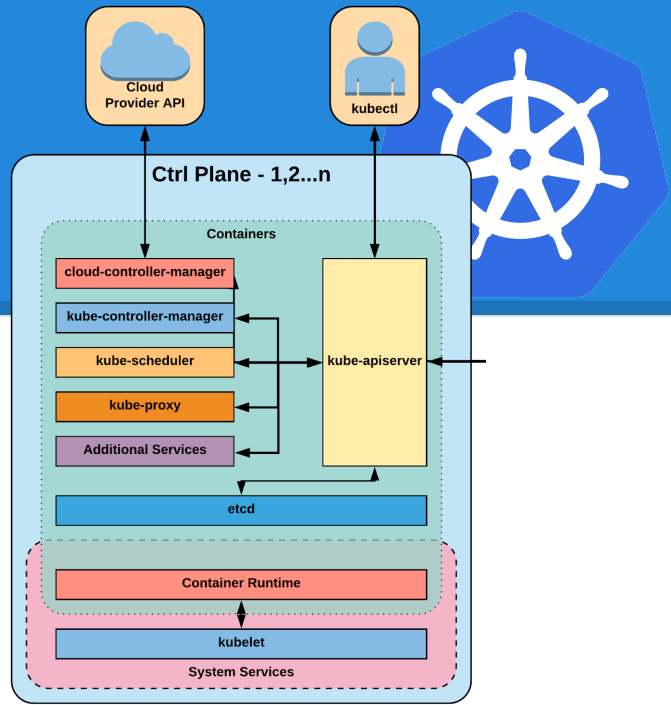
Source: https://www.weave.works/blog/what-does-production-ready-really-mean-for-a-kubernetes-cluster

kube-apiserver

- Provides a forward facing REST interface into the Kubernetes control plane and datastore
- All clients and other applications interact with Kubernetes **strictly** through the API Server
- Acts as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore



kube-controller-manager



Monitors the cluster state via the apiserver and **steers the cluster towards the desired state**

- **Node Controller**: Responsible for noticing and responding when nodes go down.
- **Replication Controller**: Responsible for maintaining the correct number of pods for every replication controller object in the system.
- **Endpoints Controller**: Populates the Endpoints object (that is, joins Services & Pods).
- **Service Account & Token Controllers**: Create default accounts and API access tokens for new namespaces.

Control loops

Drive **current state** -> **desired state**

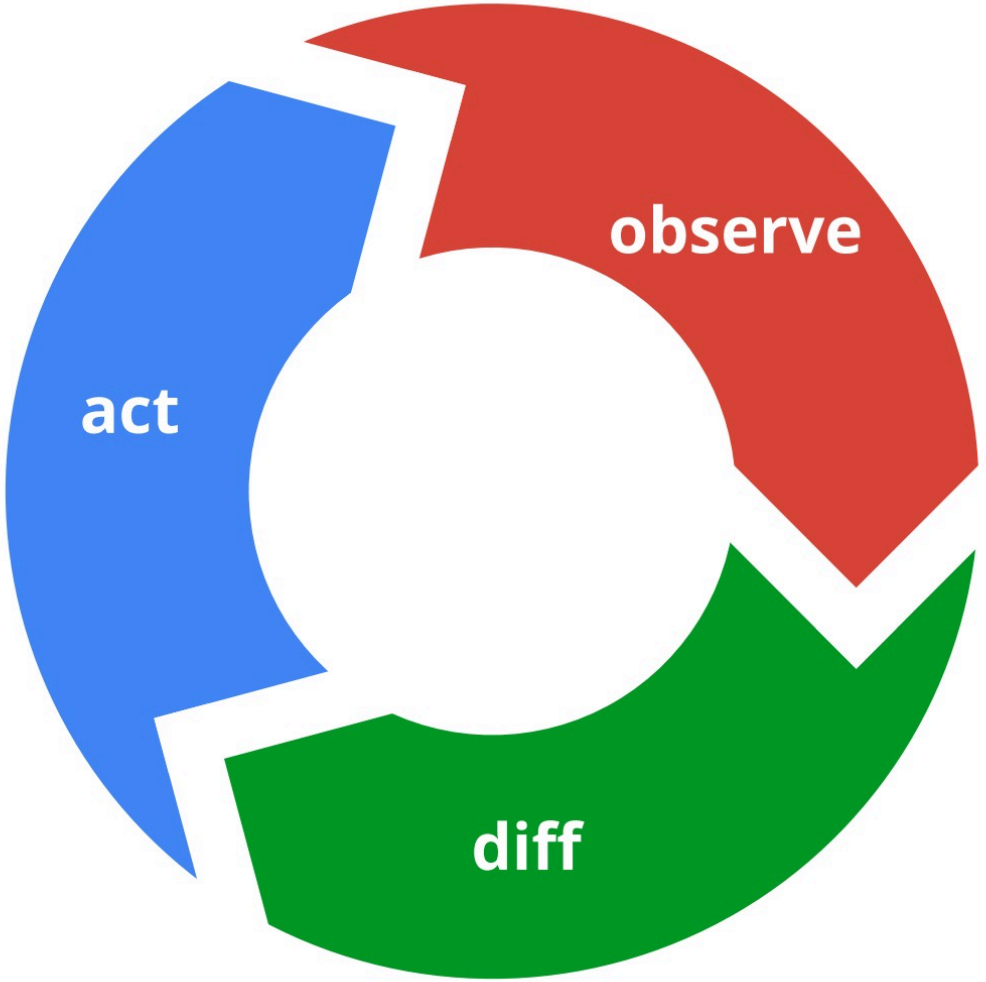
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth

Recurring pattern in the system

Example: ReplicationController



Replication Controllers

A type of *controller* (control loop)

Ensure N copies of a pod always running

- if too few, start new ones
- if too many, kill some
- group == selector

Cleanly layered on top of the core

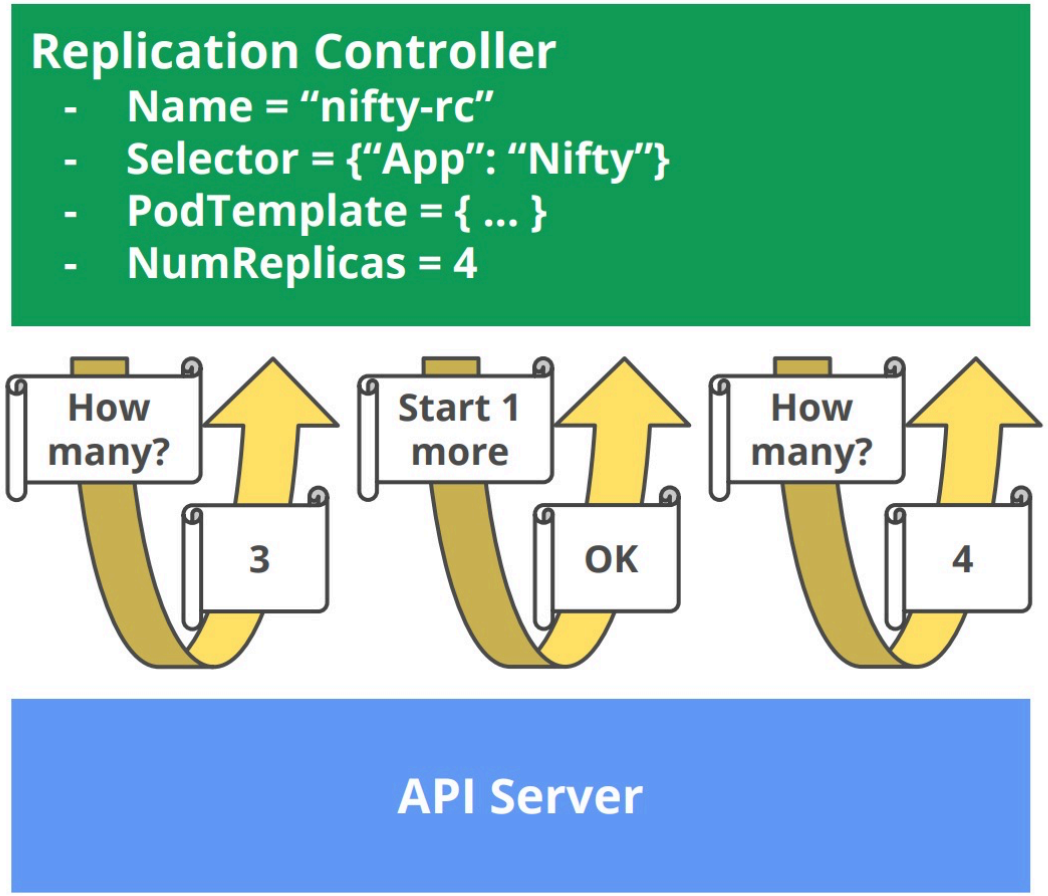
- all access is by public APIs

Replicated pods are fungible

- No implied ordinality or identity

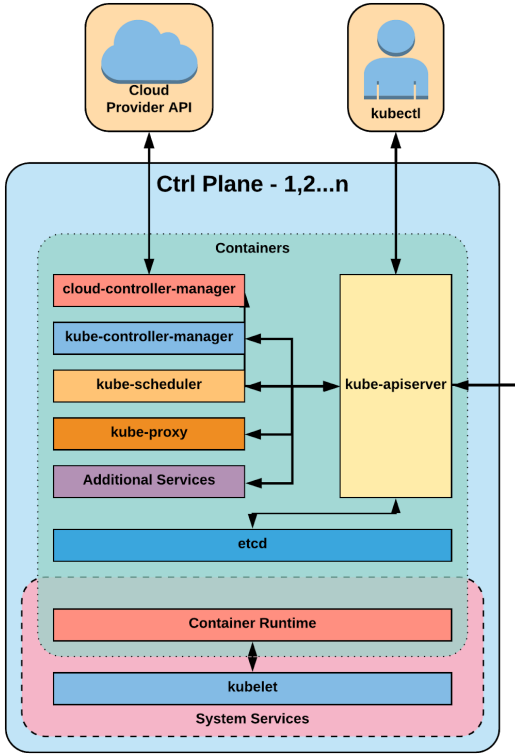
Other kinds of controllers coming

- e.g. job controller for batch



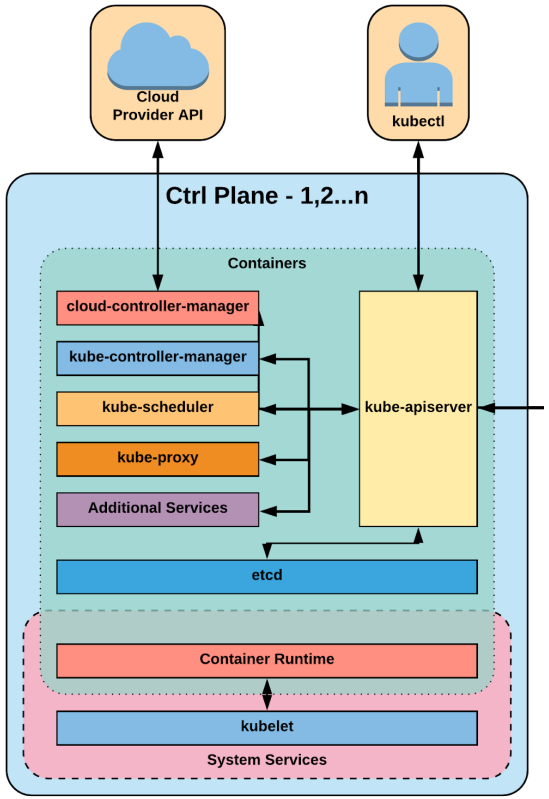
kube-scheduler

- Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on
- Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines



cloud-controller-manager

- **Node Controller**: For checking the cloud provider to determine if a node has been deleted in the cloud after it stops responding
- **Route Controller**: For setting up routes in the underlying cloud infrastructure
- **Service Controller**: For creating, updating and deleting cloud provider load balancers
- **Volume Controller**: For creating, attaching, and mounting volumes, and interacting with the cloud provider to orchestrate volumes

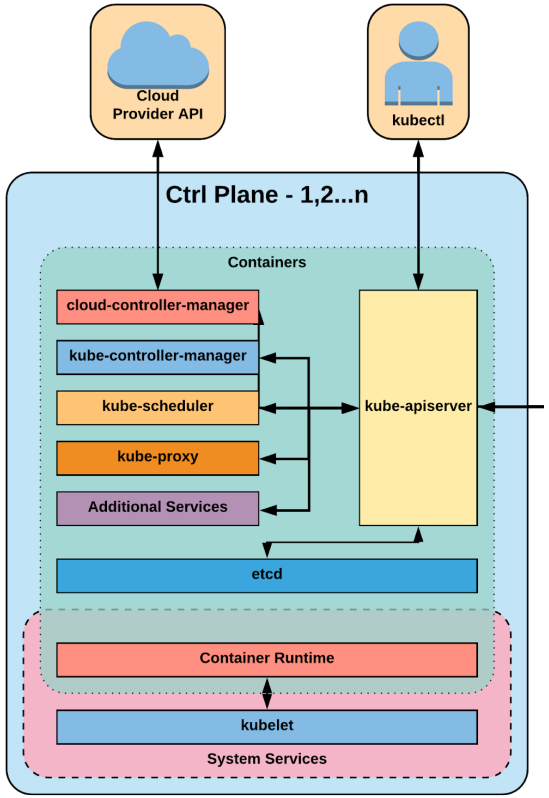


etcd

- etcd: an atomic key-value store that uses Raft consensus

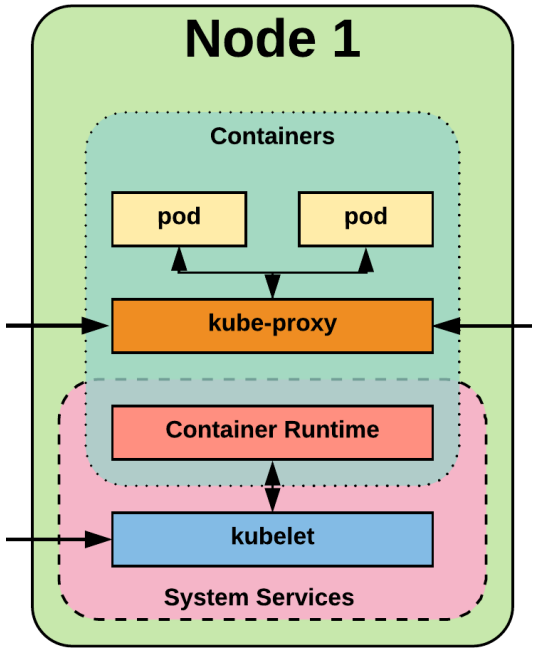
https://raft.github.io

- Backing store for all control plane metadata
- Provides a strong, consistent and highly available key-value store for persisting cluster state
- Stores objects and config information



kubelet

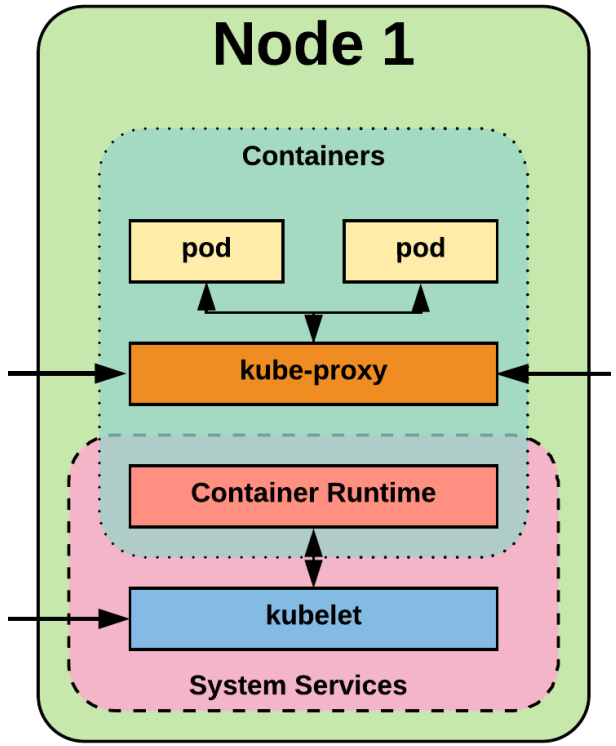
- An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.
- The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.



kube-proxy

- Manages the network rules on each node.
- Performs connection forwarding or load balancing for Kubernetes cluster services.

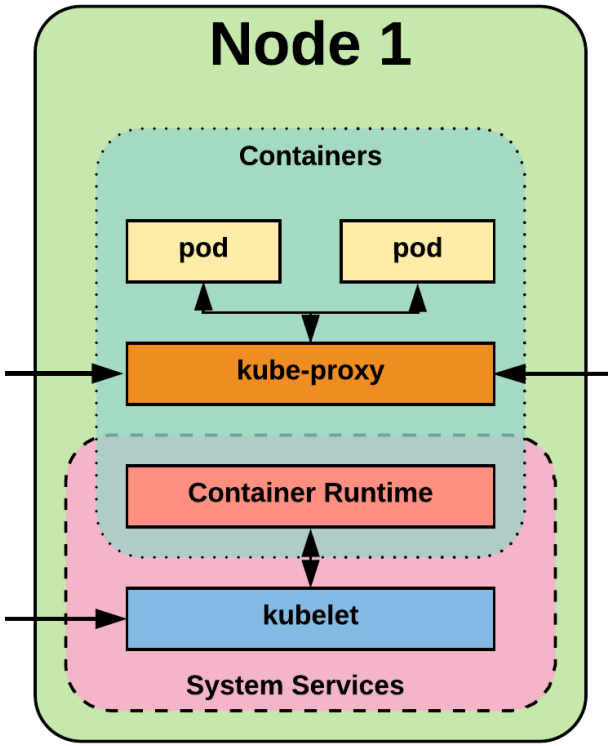
CoreDNS



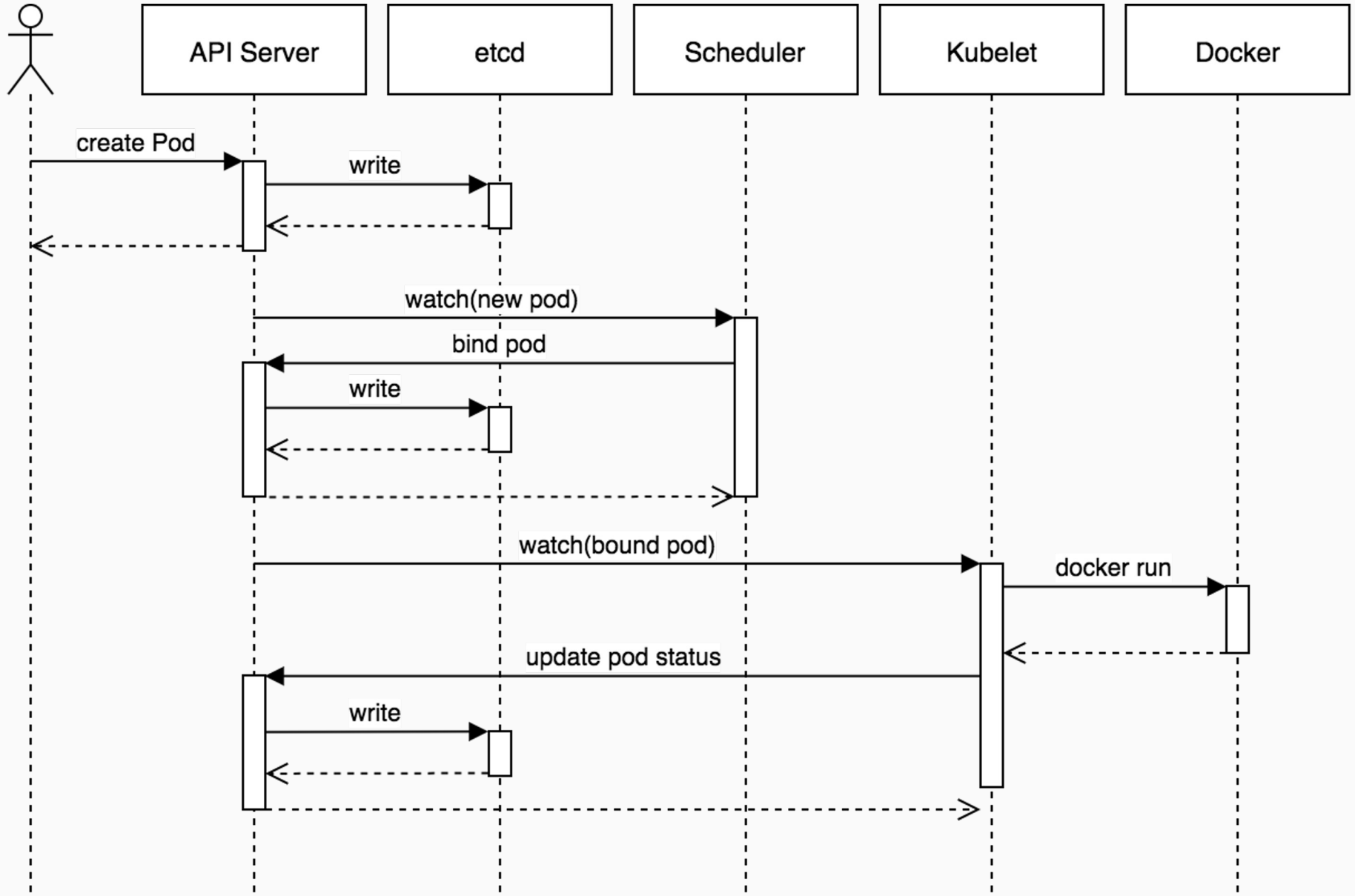
Container Runtime Engine

- A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.
  - Containerd (docker)
  - Cri-o
  - Rkt
  - Kata
  - Virtlet

CRI understands the OCI image spec



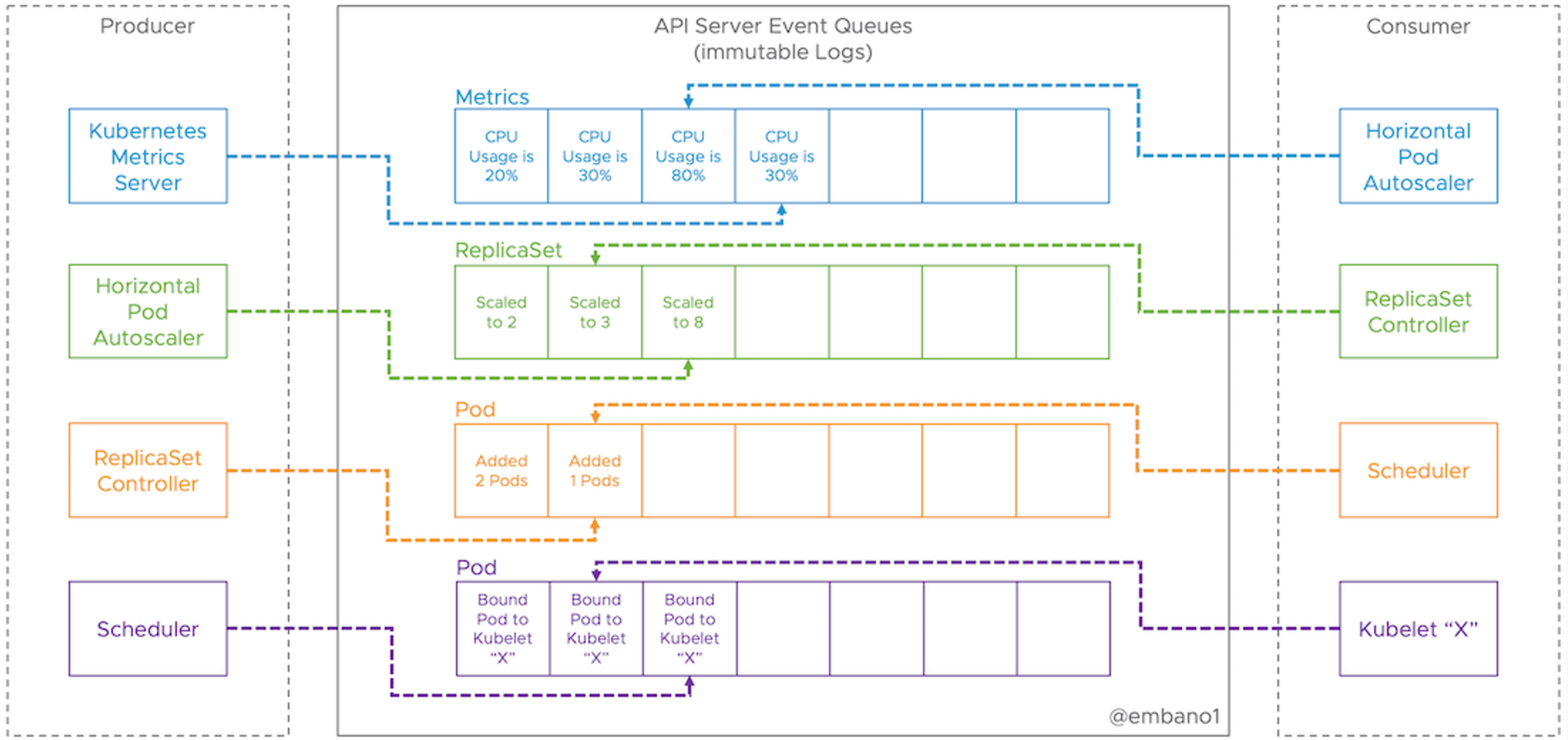
A Typical Flow: How K8s API works



Source: https://blog.heptio.com/core-kubernetes-jazz-improv-over-orchestration-a7903ea92ca

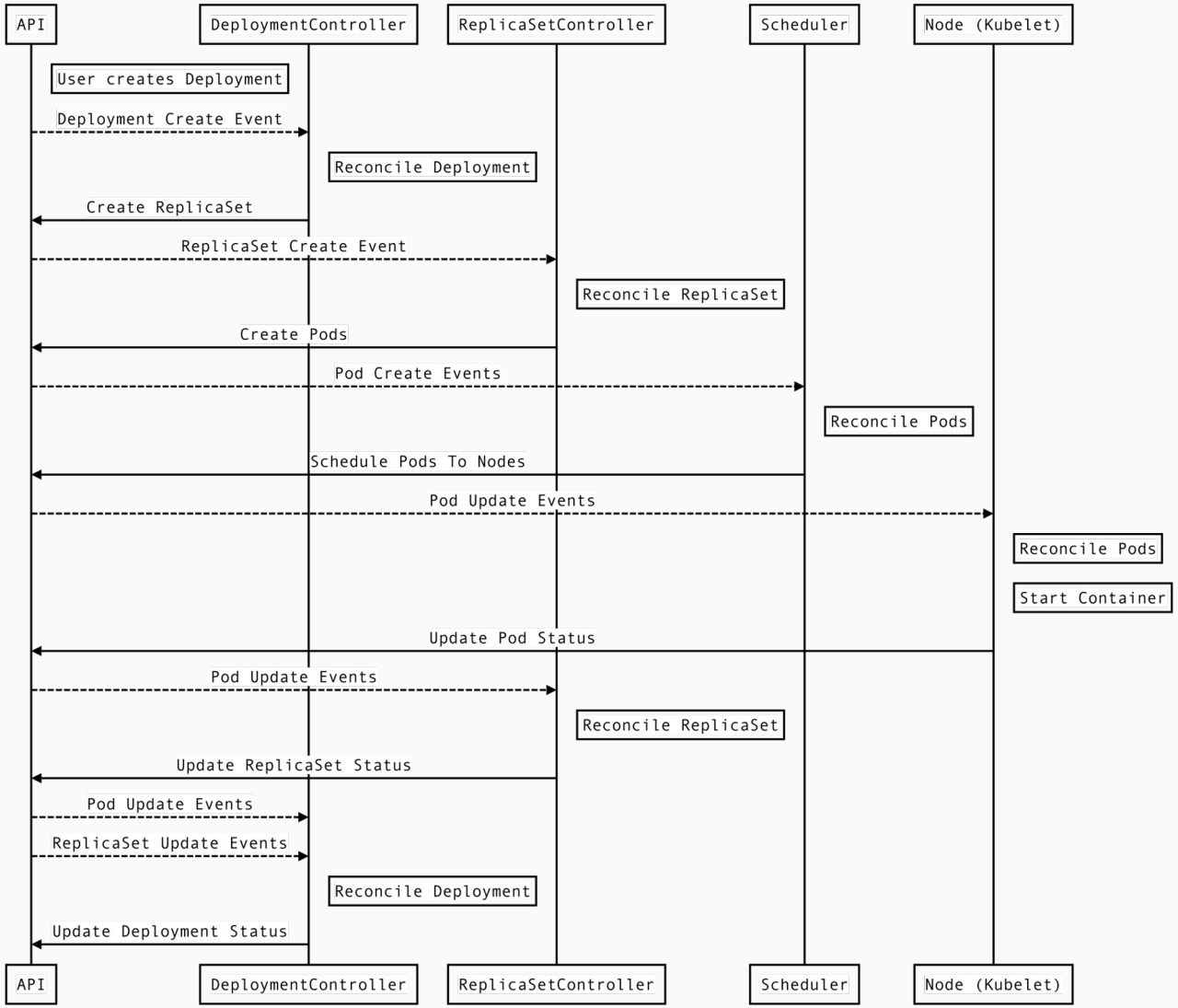
Kubernetes is like Kafka: Event-Driven Architecture

Kubernetes: “Autonomous processes reacting to events from the API server”.



Source: Events, the DNA of Kubernetes

Kubernetes Component Flow



Source: https://medium.com/payscale-tech/imperative-vs-declarative-a-kubernetes-tutorial-4be66c5d8914

Understanding Kubernetes Networking

- Every Pod has a unique IP
- Pod IP is shared by all the containers in this Pod, and it's routable from all the other Pods.
- All containers within a pod can communicate with each other.
- All Pods can communicate with all other Pods without NAT.
- All nodes can communicate with all Pods (and vice-versa) without NAT.
- The IP that a Pod sees itself as, is the same IP that others see it as.

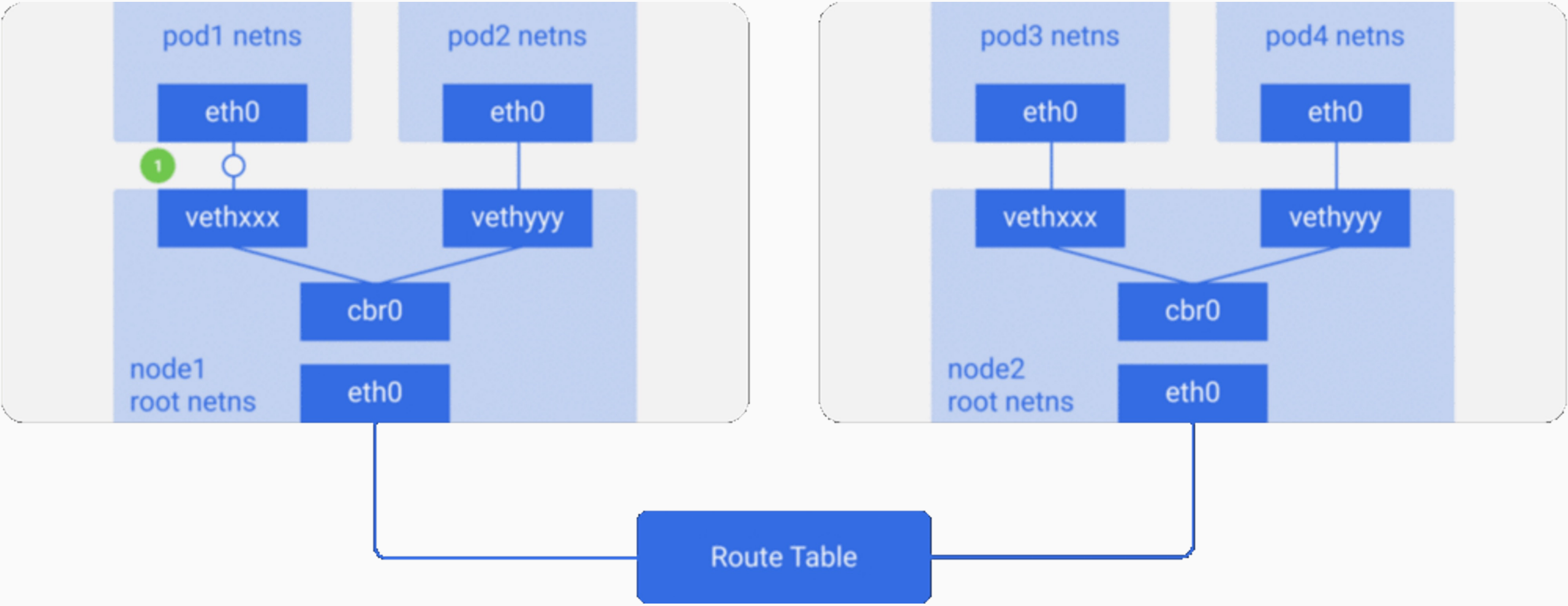
Source: https://itnext.io/an-illustrated-guide-to-kubernetes-networking-part-1-d1ede3322727

- Cluster wide DNS server
- Commonly used: CoreDNS (written in Go)
- Resolves fully and partially qualified domain names for services

- Service: nginx, Namespace: server
  - From same namespace: nginx, nginx.server, nginx.server.cluster.local
  - From other namespaces: nginx.server, nginx.server.cluster.local

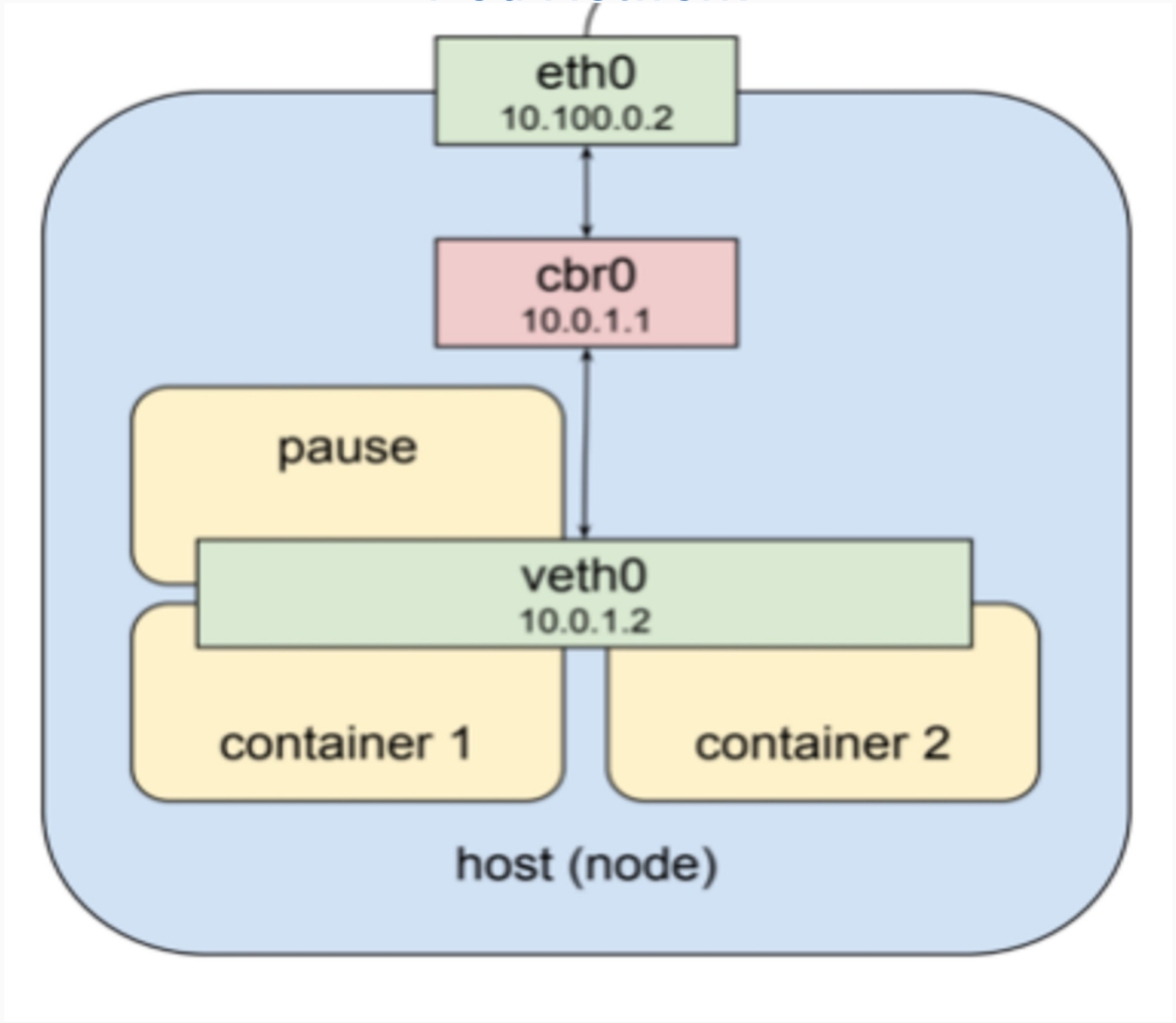
https://kubernetes.io/docs/tasks/administer-cluster/dns-custom-nameservers/

Understanding Kubernetes Networking

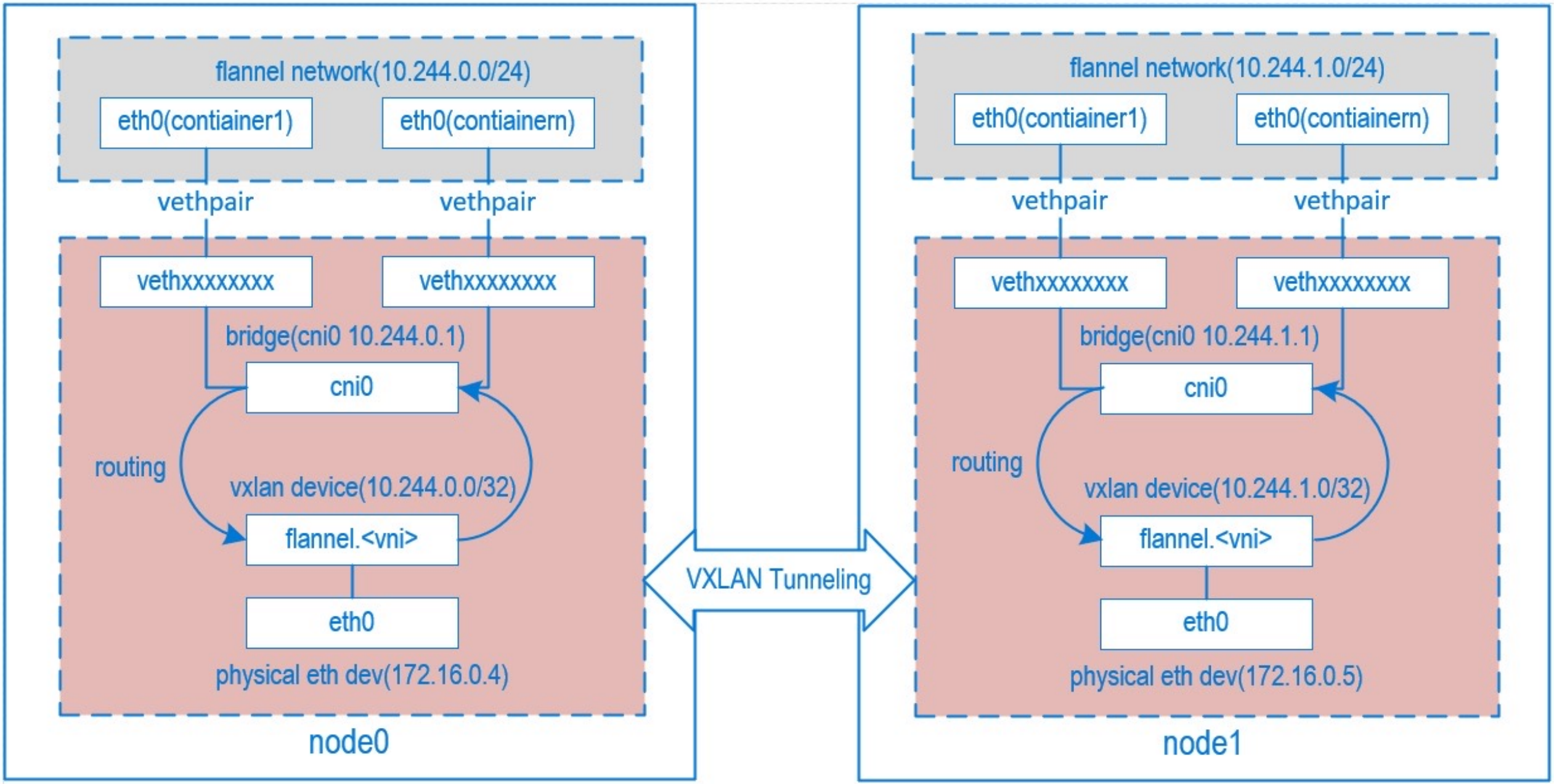


Source: https://itnext.io/an-illustrated-guide-to-kubernetes-networking-part-1-d1ede3322727

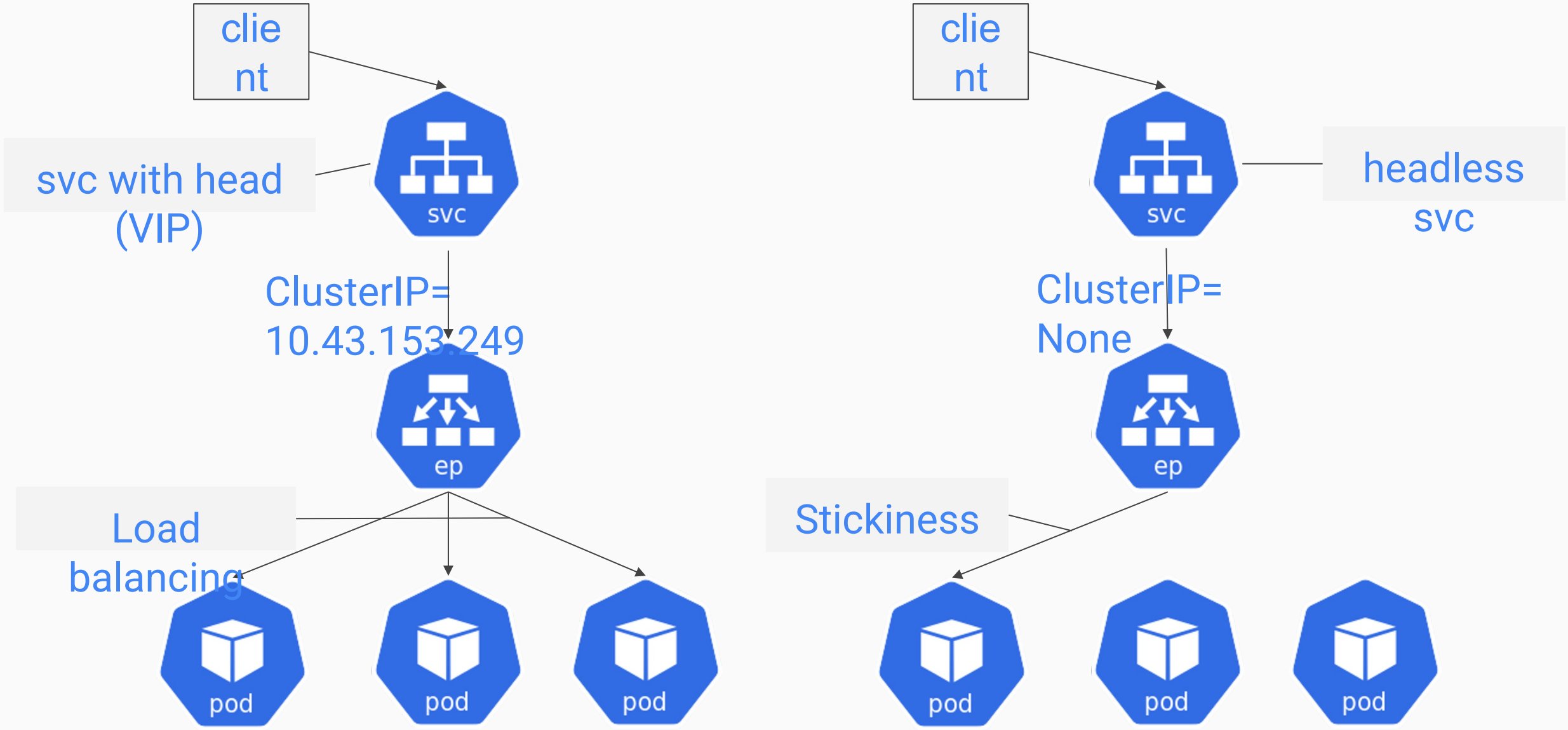
Pod Network


eth0
10.100.0.2
cbr0
10.0.1.1
pause
veth0
10.0.1.2
container 1
container 2
host (node)

Flannel (L2) CNI



Kubernetes Headless vs. ClusterIP and traffic distribution

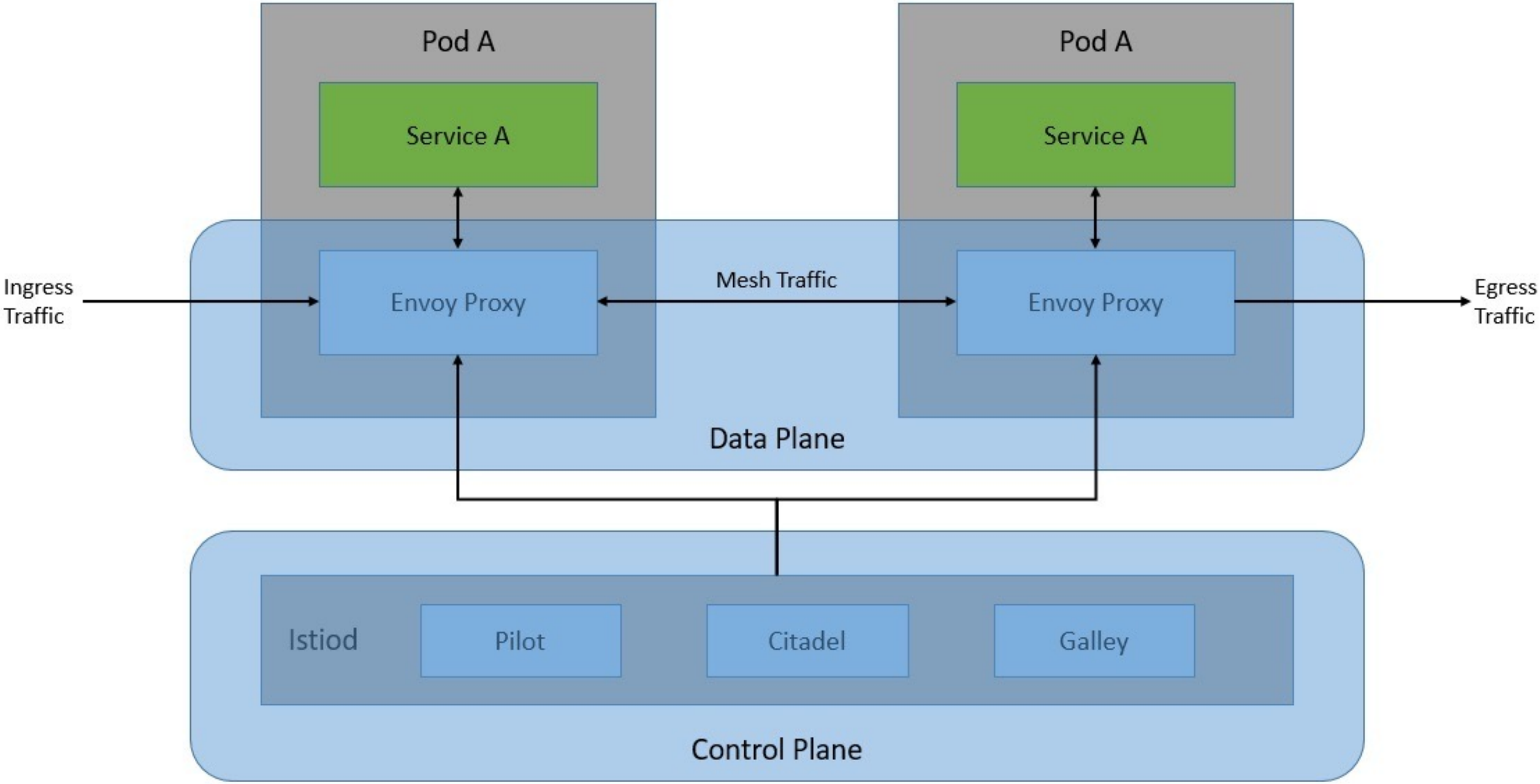


https://github.com/arashkaffamanesh/practical-kubernetes-problems#headless-services-for-stickiness

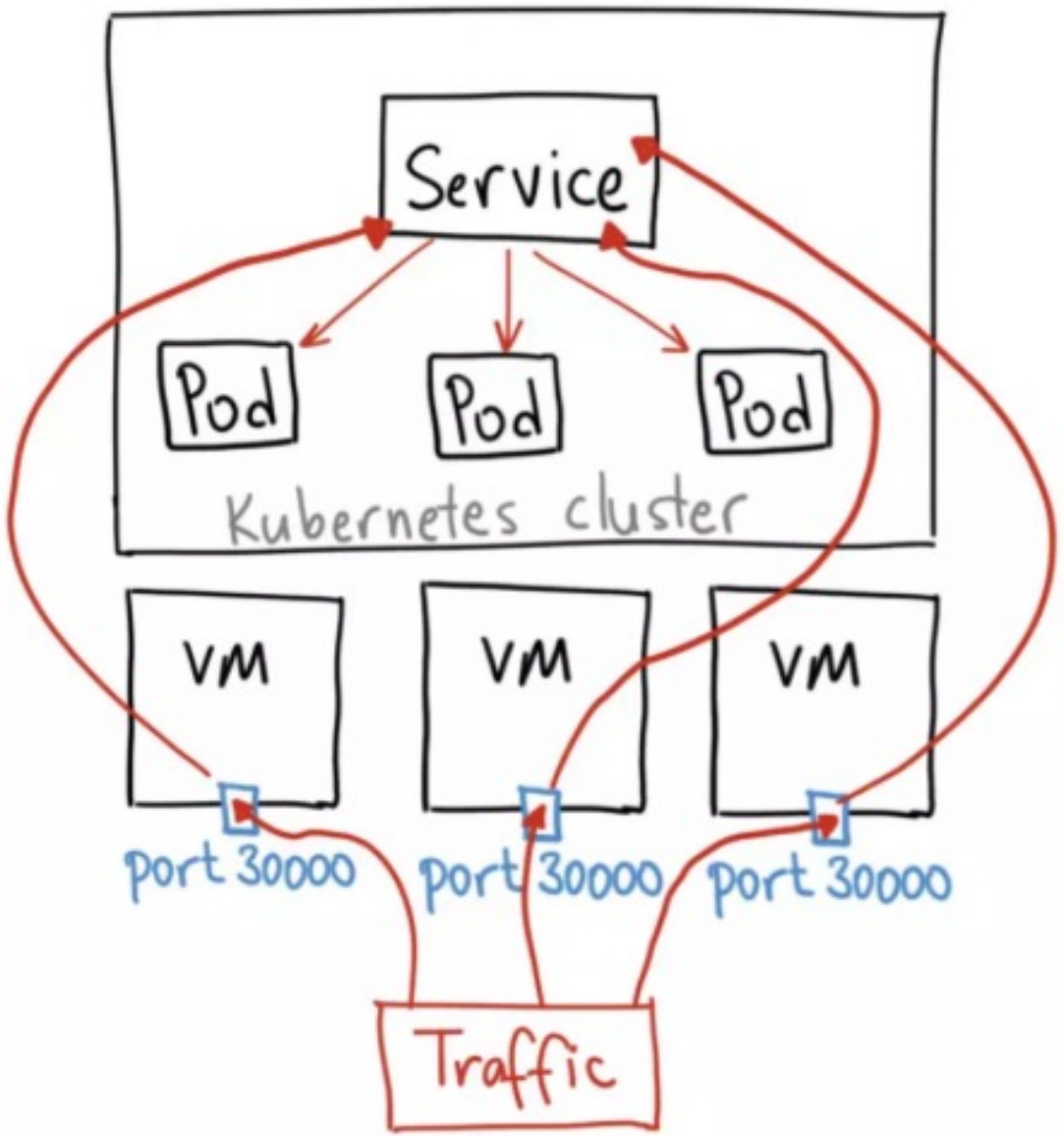
How do you think we can control this communication?

Where would you place firewalls or enforce SDN policies?

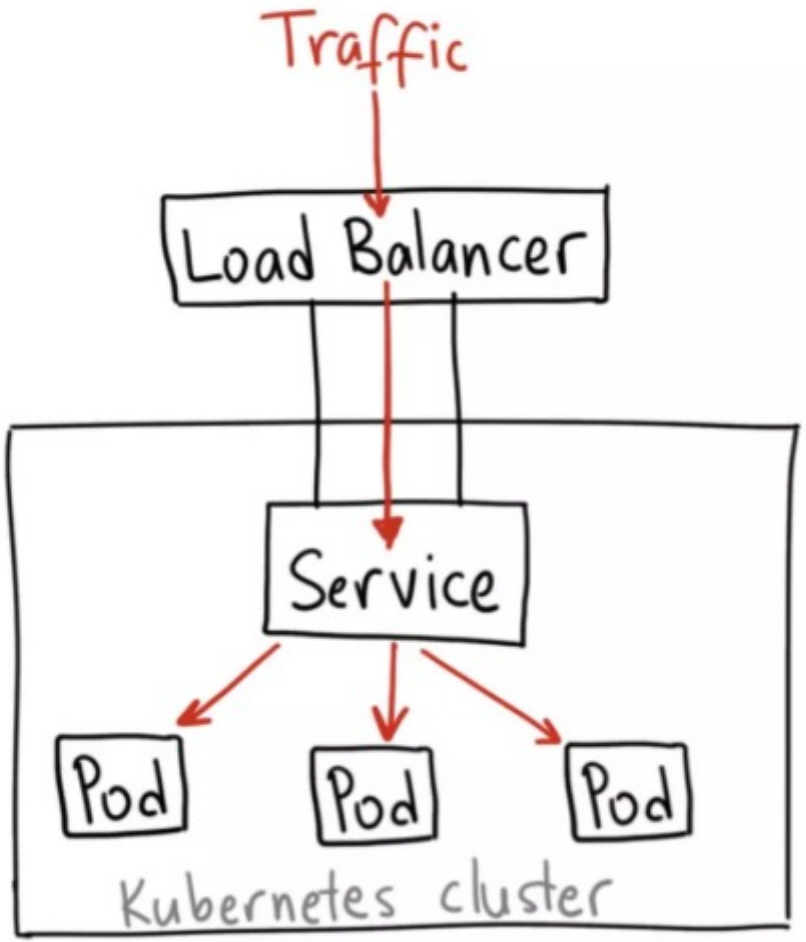
Envoy service mesh: sidecar pattern



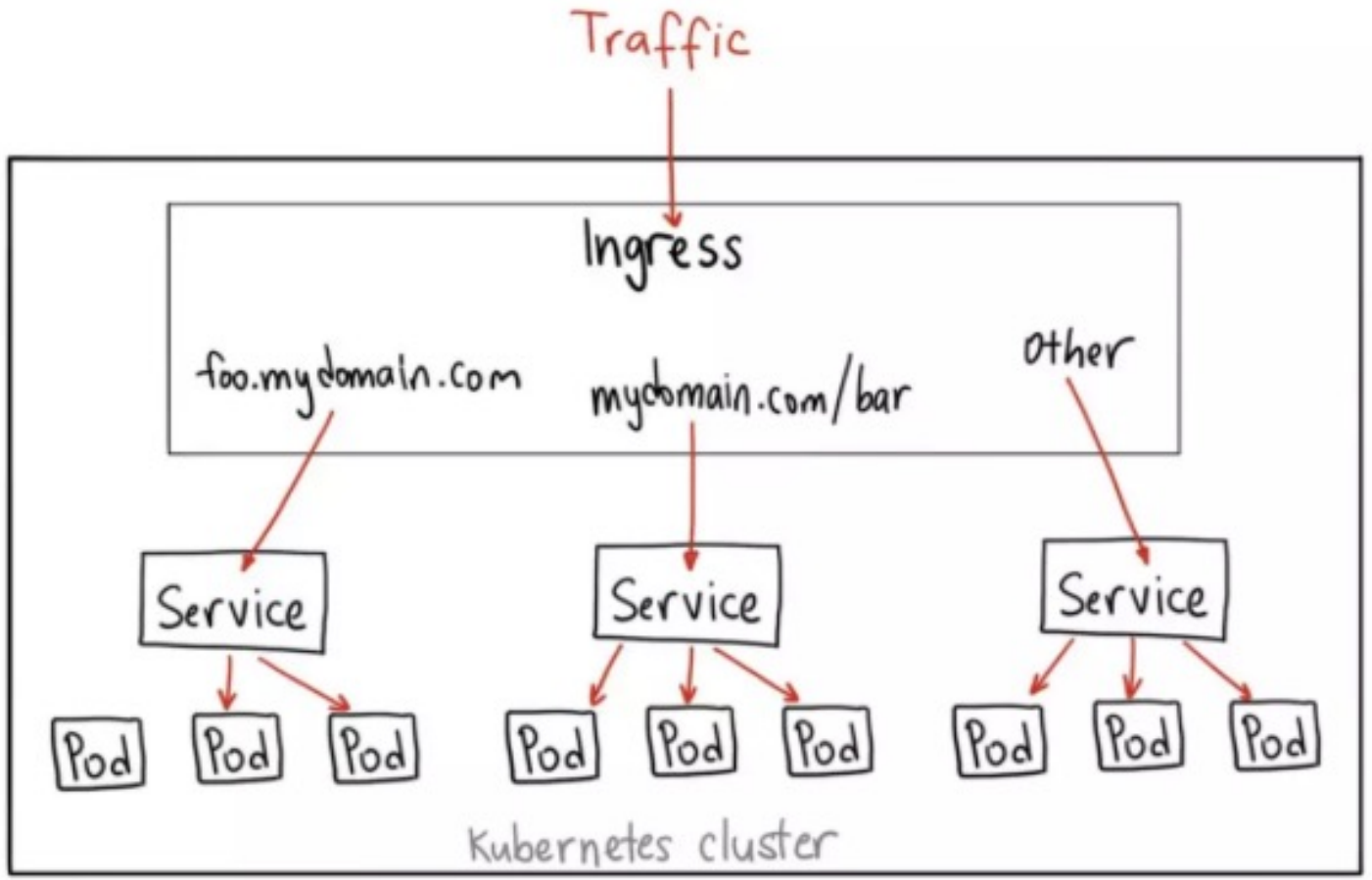
Nodeport



Load balancer



Ingress


Traffic
Ingress
foo.mydomain.com
mydomain.com/bar
Other
Service
Service
Service
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Pod
Kubernetes cluster

Ingress Controller (Traefik)