What is Kubernetes? An intro to container orchestration

With Stephen Tollenaar

A background

- DevOps Engineer at CarteNav
- 6 years of Kubernetes experience
- Certified Kubernetes Security Specialist
- Small homelab

Before we dive in

- No expectations of participation
  - But if you want use one of these on your laptop
    - Minikube
    - Microk8s
    - DO NOT USE DOCKER SWARM
- Feel free to ask questions

Quick reminder

```
FROM node:22 (last pushed 6 hours ag
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 5000
CMD ["node", "index.js"]
```

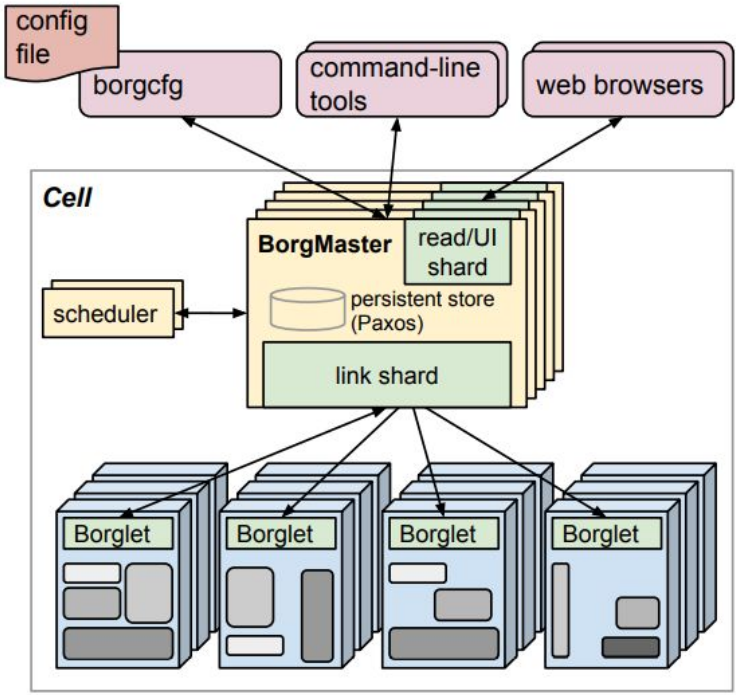
So we know what Docker is

```
stollenaar@spicetop:~/.../personal/kubetalk$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED         STATUS         PORTS                                       NAMES
f56b4d1338c1   fullstack-frontend             "docker-entrypoint.s…"   3 seconds ago   Up 3 seconds   0.0.0.0:3000->80/tcp, [::]:3000->80/tcp     fullstack-frontend-1
7a80e55ceb46   fullstack-backend              "docker-entrypoint.s…"   3 seconds ago   Up 3 seconds   0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp fullstack-backend-1
4cdf50862ebd   postgres:15                    "docker-entrypoint.s…"   2 minutes ago   Up 3 seconds   5432/tcp                                    fullstack-db-1
4760aef03187   moby/buildkit:buildx-stable-1  "buildkitd"              9 months ago    Up 4 days                                                  buildx_buildkit_competent_wescoff0
stollenaar@spicetop:~/.../personal/kubetalk$
```

Full Stack App

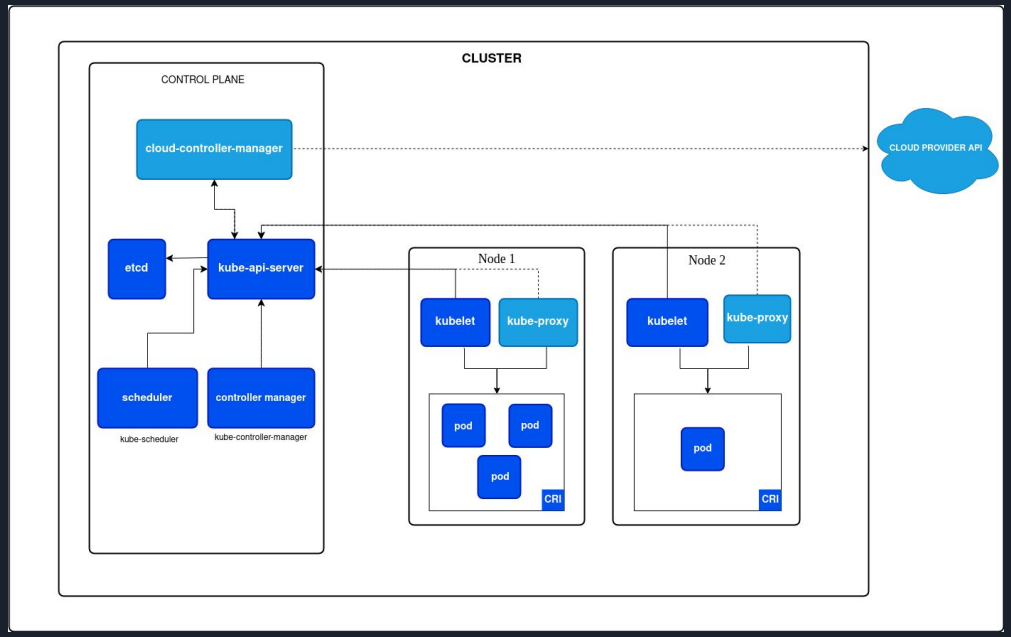
```yaml
version: "3.9"

services:
  frontend:
    build: ./frontend
    ports:
      - "3000:80"
    depends_on:
      - backend

  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - DATABASE_URL=postgres://postgres:postgres@db:5432/postgres
    depends_on:
      - db

  db:
    image: postgres:15
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: postgres
    volumes:
      - db_data:/var/lib/postgresql/data

volumes:
  db_data:
```

Borg



- The first iteration of container orchestration
- Does feature “pods”
- Borg tasks are run inside a Linux cgroup-based resource container
- Still very open to accessing host resources

Introducing Kubernetes



- Removes the headache of managing containers
  - Your sanity might falter
- You specify what you want, Kubernetes takes care of the rest
- Scaling up to an insane number of resources*
  - No more than 110 pods per node
  - No more than 5,000 nodes
  - No more than 150,000 total pods
  - No more than 300,000 total containers

What is a Pod?

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.29.1
      ports:
        - containerPort: 8080
```

- The lowest of building blocks
- The container array is “similar” to the docker compose services
- Can hold more containers, and even init containers
- We apply/delete these manifests using kubectl

```yaml
version: "3.9"
services:
  nginx:
    image: nginx:1.29.1
    ports:
      - "8080:8080"
```

Pod Deployment methods

But a pod is just a singular throwaway thing, we need something more “persistent”

So we got:

- Deployment
  - ReplicaSet
- StatefulSet
- DaemonSet
- Job
- CronJob

What is a Deployment?

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.29.1
          ports:
            - containerPort: 8080
```

- A Deployment manages a set of Pods to run an application workload, usually one that doesn't maintain state.
- The template resembles the pod from before
- Each version of a Deployment creates a ReplicaSet

```
stollenaar@spicetop:~/.../personal/kubetalk$ kubectl get replicasets.apps
NAME               DESIRED   CURRENT   READY   AGE
nginx-844c99f596   3         3         0       2s
stollenaar@spicetop:~/.../personal/kubetalk$
```

```
stollenaar@spicetop:~/.../personal/kubetalk$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-844c99f596-cdbb4   1/1     Running   0          76s
nginx-844c99f596-lxq5d   1/1     Running   0          76s
nginx-844c99f596-rm9l6   1/1     Running   0          76s
stollenaar@spicetop:~/.../personal/kubetalk$
```

What is a StatefulSet?

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
spec:
  serviceName: "postgres"
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:17
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_PASSWORD
              value: password
          volumeMounts:
            - name: pgdata
              mountPath: /var/lib/postgresql/
  volumeClaimTemplates:
    - metadata:
        name: pgdata
      spec:
        accessModes: ["ReadWriteOnce"]
        resources:
          requests:
            storage: 1Gi
```

- A StatefulSet runs a group of Pods, and maintains a sticky identity for each of those Pods.
- Useful for databases.
- Same building block from the Pod
- No ReplicaSet
- Has the replica number, instead of hash

```
stollenaar@spicetop:~/.../personal/kubetalk$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
postgres-0   1/1     Running   0          89s
```

What is a DaemonSet?

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: node-exporter
  labels:
    app: node-exporter
spec:
  selector:
    matchLabels:
      app: node-exporter
  template:
    metadata:
      labels:
        app: node-exporter
    spec:
      containers:
        - name: node-exporter
          image: prom/node-exporter:latest
          ports:
            - containerPort: 9100
```

- A DaemonSet ensures that all (or some) Nodes run a copy of a Pod

Some typical uses of a DaemonSet are:

- Running a cluster storage daemon on every node
- Running a logs collection daemon on every node
- Running a node monitoring daemon on every node

What is a Job?

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: hello-job
spec:
  template:
    spec:
      containers:
        - name: hello
          image: busybox
          command: ["echo", "Hello, Kubernetes!"]
      restartPolicy: Never
```

- Jobs represent one-off tasks that run to completion and then stop.

What is a CronJob?

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: postgres-backup
spec:
  schedule: "0 2 * * *" # Runs daily at 2 AM
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: pg-backup
              image: postgres:16
              env:
                - name: PGPASSWORD
                  value: examplepassword
              command:
                - /bin/sh
                - -c
                - pg_dump -h postgres -U postgres -F c -b -v -f /backup/backup.dump postgres
              volumeMounts:
                - name: backup-volume
                  mountPath: /backup
          restartPolicy: OnFailure
```

- A regular recurring Job
- Builds on top of the Job template from before

Where in Kubernetes do we run these things?

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: kubetalk
```

```
stollenaar@spicetop:~/.../personal/kubetalk$ kubectl get ns
NAME                              STATUS   AGE
cert-manager                      Active   406d
cmstate-operator                  Active   406d
copypastabotv2                    Active   258d
default                           Active   406d
diplomacy                         Active   214d
external-dns                      Active   23d
github-arc                        Active   128d
jellyfin                          Active   406d
kube-node-lease                   Active   406d
kube-public                       Active   406d
kube-system                       Active   406d
kubelet-serving-cert-approver     Active   367d
kubetalk                          Active   4d22h
metallb-system                    Active   406d
monitoring                        Active   361d
ollama                            Active   227d
ollamabot                         Active   41d
openebs                           Active   406d
renovate                          Active   102d
statisticsbot                     Active   406d
tailscale                         Active   406d
uptime-kuma                       Active   205d
vault                             Active   406d
stollenaar@spicetop:~/.../personal/kubetalk$
```

- Used for isolating groups of resources within a single cluster
- Not everything can be namespaced

The Kubernetes Node

- Runs your work loads
- Is either a physical or virtual workload

```
stollenaar@spicetop:~/.../personal/kubetalk$ kubectl get nodes
NAME           STATUS   ROLES           AGE    VERSION
talos-7zr-i5q   Ready    control-plane   197d   v1.34.0
talos-e5t-zk5   Ready    worker          406d   v1.34.0
talos-iso-cgi   Ready    worker          266d   v1.34.0
stollenaar@spicetop:~/.../personal/kubetalk$
```

Putting it all Together

```
stollenaar@spicetop:~$ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/kubetalk-app-6bb5d66f99-5scfb       1/1     Running   0          2m34s
pod/kubetalk-backend-664c48448c-zwg5w   1/1     Running   0          2m34s
pod/postgres-0                          1/1     Running   0          3m13s

NAME                        TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/kubetalk-app        ClusterIP   10.104.95.93     <none>        80/TCP     3m13s
service/kubetalk-backend    ClusterIP   10.106.152.120   <none>        4000/TCP   3m13s
service/postgres            ClusterIP   10.106.230.46    <none>        5432/TCP   3m13s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/kubetalk-app        1/1     1            1           2m34s
deployment.apps/kubetalk-backend    1/1     1            1           2m34s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/kubetalk-app-6bb5d66f99        1         1         1       2m34s
replicaset.apps/kubetalk-backend-664c48448c    1         1         1       2m34s

NAME                          READY   AGE
statefulset.apps/postgres     1/1     3m13s
stollenaar@spicetop:~$
```

- The fullstack app in a namespace

Another Example

```
stollenaar@spicetop:~/.../personal/kubetalk$ kubectl get pods,deployments,statefulset
NAME                                 READY   STATUS    RESTARTS       AGE
pod/bazarr-768c7d8d78-lthf6          1/1     Running   1 (3d8h ago)   4d
pod/byparr-6c9c5b8749-tlxts          1/1     Running   3 (10h ago)    5d8h
pod/jellyfin-5847f9896d-8b52h        1/1     Running   0              47h
pod/jellyseerr-b644f77d-8gp4r        1/1     Running   0              3d13h
pod/postgres-1                       1/1     Running   0              3d3h
pod/prowlarr-5fffb576bc-2jw47        1/1     Running   1 (3d8h ago)   4d
pod/qbittorrent-5c77b5d657-5q2lj     2/2     Running   0              3d3h
pod/radarr-d7c9bc8d5-v674v           1/1     Running   1 (3d8h ago)   4d
pod/sonarr-df775568c-fkkbz           1/1     Running   1 (3d8h ago)   4d

NAME                                 READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/bazarr               1/1     1            1           121d
deployment.apps/byparr               1/1     1            1           75d
deployment.apps/decluttarr           0/0     0            0           127d
deployment.apps/jellyfin             1/1     1            1           416d
deployment.apps/jellyseerr           1/1     1            1           416d
deployment.apps/prowlarr             1/1     1            1           416d
deployment.apps/qbittorrent          1/1     1            1           253d
deployment.apps/radarr               1/1     1            1           416d
deployment.apps/sonarr               1/1     1            1           416d
deployment.apps/tdarr-node           0/0     0            0           16d
deployment.apps/tdarr-server         0/0     0            0           16d
```

But what about extra config?

What is a ConfigMap?

v1@configmap+v1@configmap.json

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-env
data:
  DB_HOST: localhost
  DB_PORT: "5432"
  NODE_ENV: production
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-file
data:
  user-interface.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

- A ConfigMap is used to store non-confidential data in key-value pairs.
- Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume.
- When values are updated, it automatically propagates

What is a Secret?

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secret
  namespace: kubetalk
type: Opaque
stringData:
  POSTGRES_PASSWORD: examplepassword
```

- A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key
- Are similar in config as a ConfigMap
- When viewing the values, they are encoded

```
stollenaar@spicetop:~$ kubectl get secret app-secret -o yaml
apiVersion: v1
data:
  POSTGRES_PASSWORD: ZXhhbXBsZXBhc3N3b3Jk
kind: Secret
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"Secret","metadata":{"annotations":{},"nam
  creationTimestamp: "2025-09-16T23:28:34Z"
  name: app-secret
  namespace: kubetalk
  resourceVersion: "89063781"
  uid: bd1fae01-b19c-4c57-8cea-d720ff2331dd
type: Opaque
```

But what about storage?

- We have persistent storage and ephemeral

What is a PersistentVolume?

io.k8s.api.core.v1.PersistentVolume (v1@persistentvolume.json)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  annotations:
    pv.kubernetes.io/provisioned-by: nfs.csi.k8s.io
    volume.kubernetes.io/provisioner-deletion-secret-name: ""
    volume.kubernetes.io/provisioner-deletion-secret-namespace: ""
  finalizers:
    - kubernetes.io/pv-protection
  name: pvc-d7c4c439-916a-4476-88c2-24a84429a485
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 20Gi
  claimRef:
    apiVersion: v1
    kind: PersistentVolumeClaim
    name: postgres-1
    namespace: jellyfin
    resourceVersion: "89451732"
  csi:
    driver: nfs.csi.k8s.io
    volumeAttributes:
      csi.storage.k8s.io/pv/name: pvc-d7c4c439-916a-4476-88c2-24a84429a485
      csi.storage.k8s.io/pvc/name: postgres-1
      csi.storage.k8s.io/pvc/namespace: jellyfin
      server: 192.168.2.113
      share: /mnt/main/kubernetes
      storage.kubernetes.io/csiProvisionerIdentity: 1758195604893-1373-nfs.csi.k8s.io
      subDir: jellyfin/postgres-1
    volumeHandle: 192.168.2.113#mnt/main/kubernetes#jellyfin/postgres-1#pvc-d7c4c439-916a-4476-88c2-24a84429a485#
  mountOptions:
    - nolock
    - vers=4
  persistentVolumeReclaimPolicy: Retain
```

- Describes in detail what your storage for your volume is

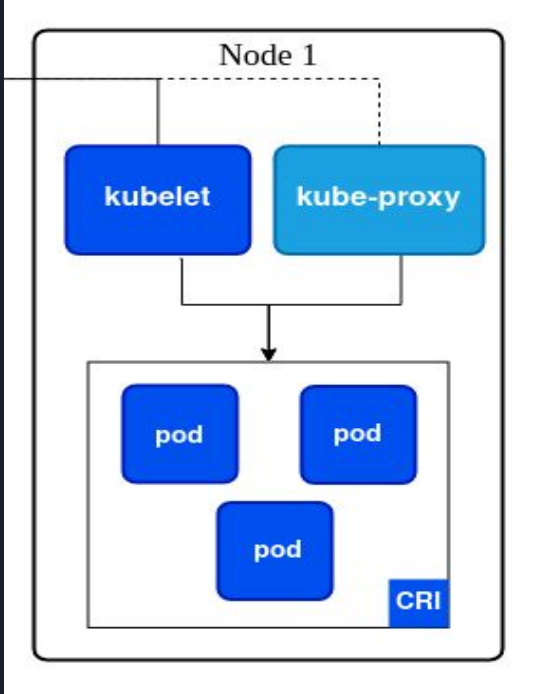
What is a PersistentVolumeClaim?

io.k8s.api.core.v1.PersistentVolumeClaim (v1@persistentvolumeclaim.json)

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    cnpg.io/nodeSerial: "1"
    cnpg.io/operatorVersion: 1.27.0
    cnpg.io/pvcStatus: ready
    pv.kubernetes.io/bind-completed: "yes"
    pv.kubernetes.io/bound-by-controller: "yes"
    volume.beta.kubernetes.io/storage-provisioner: nfs.csi.k8s.io
    volume.kubernetes.io/storage-provisioner: nfs.csi.k8s.io
  creationTimestamp: "2025-09-18T19:50:43Z"
  finalizers:
    - kubernetes.io/pvc-protection
  labels:
    cnpg.io/cluster: postgres
    cnpg.io/instanceName: postgres-1
    cnpg.io/instanceRole: primary
    cnpg.io/pvcRole: PG_DATA
    role: primary
  name: postgres-1
  namespace: jellyfin
  ownerReferences:
    - apiVersion: postgresql.cnpg.io/v1
      controller: true
      kind: Cluster
      name: postgres
      uid: 90afed56-3b1e-4c9c-92df-1cc4252ad04b
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: nfs-csi-main
  volumeMode: Filesystem
```

- Tells kubernetes what you want, how large, and from which provider
- Interacts with the StorageClass, which handles the volume requests

But How do they talk to each other?



- The easy: magic
- The complicated: Container Network Interfaces
- The popular:
  - Flannel
  - Calico
  - Cillium
- The cloud platforms usually use their own
- Runs on each node, (Daemonset)

The Services

```
io.k8s.api.core.v1.Service (v1@service.json)
apiVersion: v1
kind: Service
metadata:
  name: sonarr
  namespace: jellyfin
spec:
  ports:
    - port: 8989
      protocol: TCP
      targetPort: 8989
  selector:
    app: sonarr
  sessionAffinity: None
  type: ClusterIP
```

- Since Pods come and go, so do their Pod IPs
- We need something that stays the same so other Pods don't crash
- Knows where to send the network traffic through the labels
- 4 types:
  - ClusterIP
    - For internal traffic
  - LoadBalancer
    - To external incoming traffic in a balancing method
  - NodePort
    - To handle incoming traffic if loadbalancing is not an option
  - ExternalName
    - CNAME for a DNS

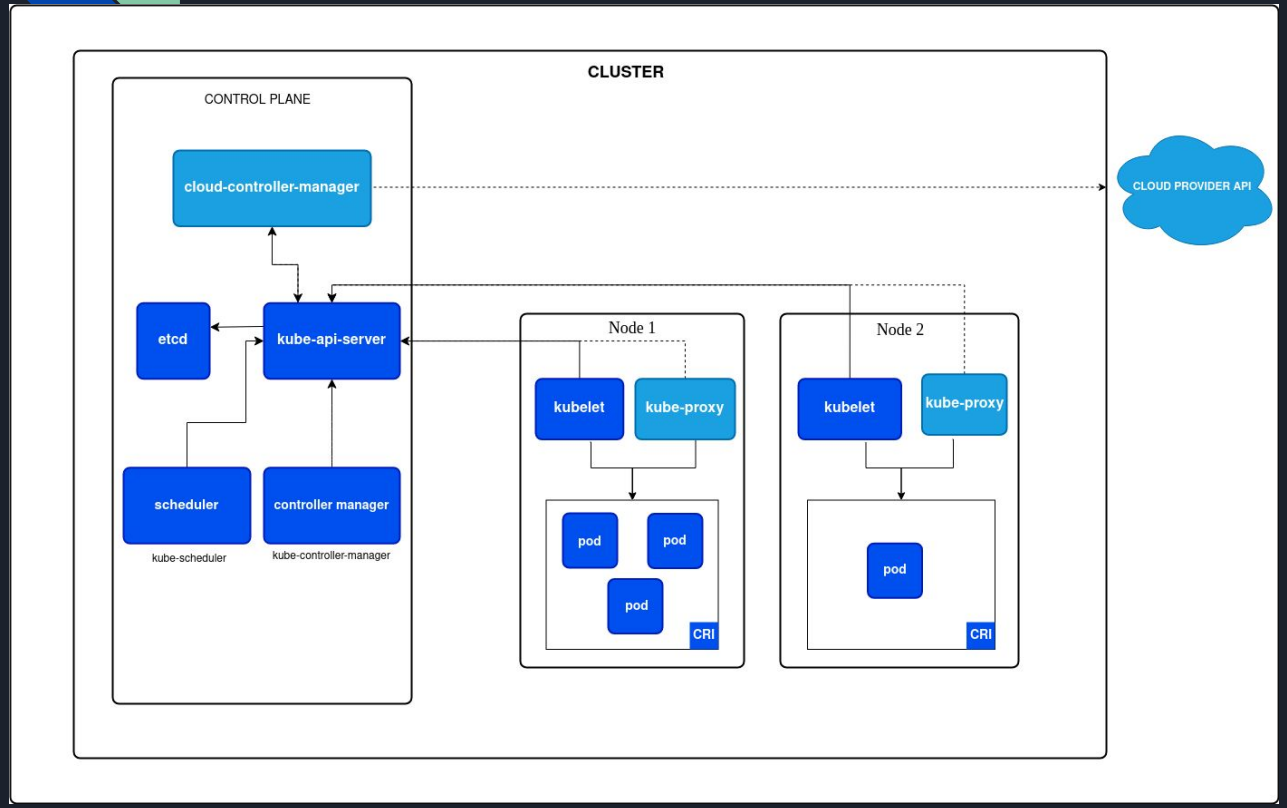
The Ingress

io.k8s.api.networking.v1.Ingress (v1@ingress.json)

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    kubernetes.io/ingress.class: nginx
  name: jellyfin
  namespace: jellyfin
spec:
  ingressClassName: nginx
  rules:
  - host: jellyfin.home.spicedelver.me
    http:
      paths:
      - backend:
          service:
            name: jellyfin-web
            port:
              number: 8096
        path: /
        pathType: ImplementationSpecific
  tls:
  - hosts:
    - jellyfin.home.spicedelver.me
    secretName: jellyfin-tls
```

- Make your HTTP (or HTTPS) network service available using a protocol-aware configuration mechanism, that understands web concepts like URIs, hostnames, paths, and more.
- You need a controller to direct this kind of traffic

But what is Kubernetes Actually?

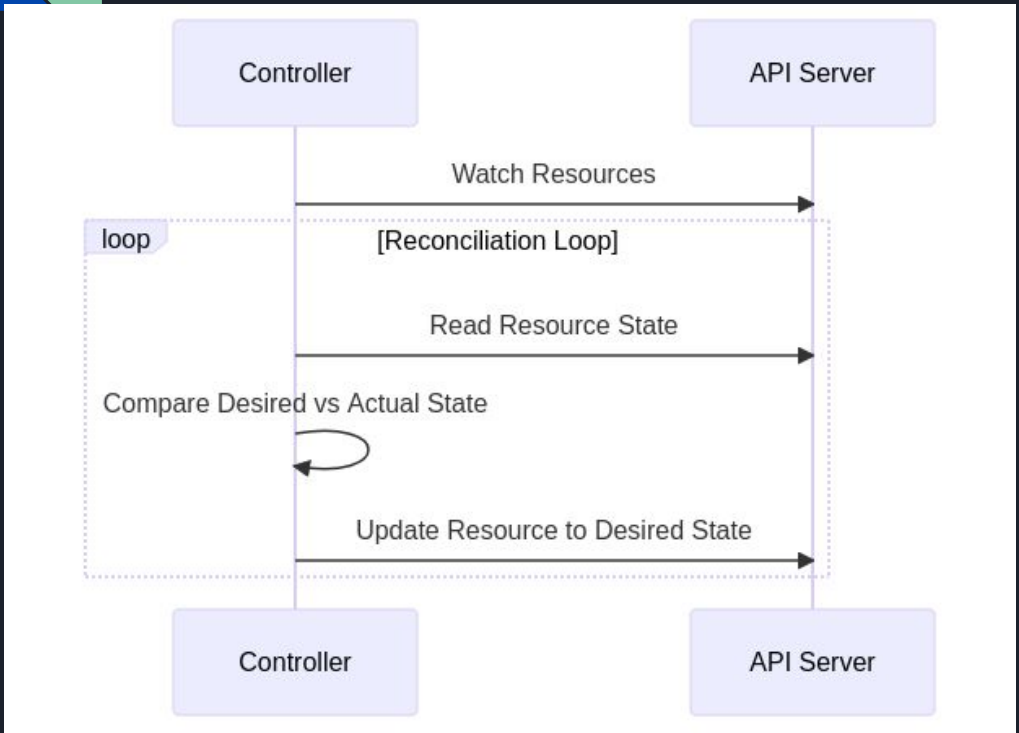


- Everything is a controller, It's controllers all the way down
- Reconciliation loops

The Controllers

- Kube-api-server
  - REST API to manage EVERYTHING in a cluster
- Kubelet
  - Manages the node within a cluster
- Scheduler
  - Makes sure pods can be provisioned and run
- Controller-manager
  - Makes sure the desired state of a cluster is achieved
- ETCD/DQLite
  - The Key Value store of all your cluster resources
- Kube-Proxy
  - Manages your container network, this will different per cluster

How does a controller work



- The Reconciliation loop
- We introduce CustomResourceDefinitions to expand what a Resource is

Custom Controller Example

- Quick demo from my cmstate-operator
  https://github.com/STollenaar/cmstate-injector-operator/tree/main

PizzaKube

Yes, kubernetes could order you a pizza:

https://github.com/grantgumina/provider-pizza/blob/master/examples/order/example-order.yaml

Questions?

Resources

- Find me almost every Thursday evening at CTS
  - 7pm-9pm Jumping Bean Elizabeth Ave
- https://github.com/STollenaar/kubetalk