Kubernetes

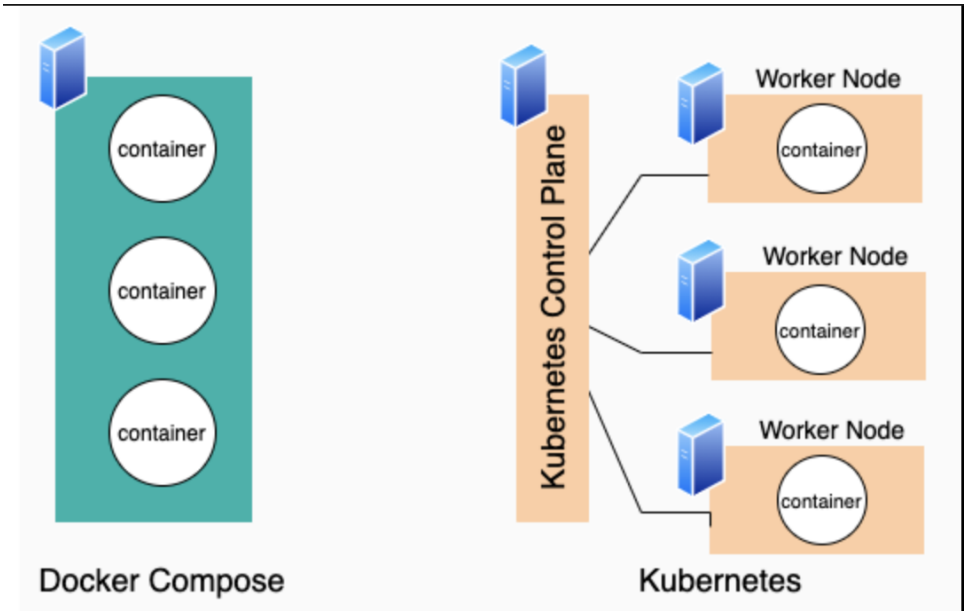
Created	@April 24, 2023 2:23 PM
Tags	Study

Why needed?

to orchestrate containers in a distributed system, where a container instance can die unintentionally but will have multiple replicas (redundancy) to guarantee service runs stably

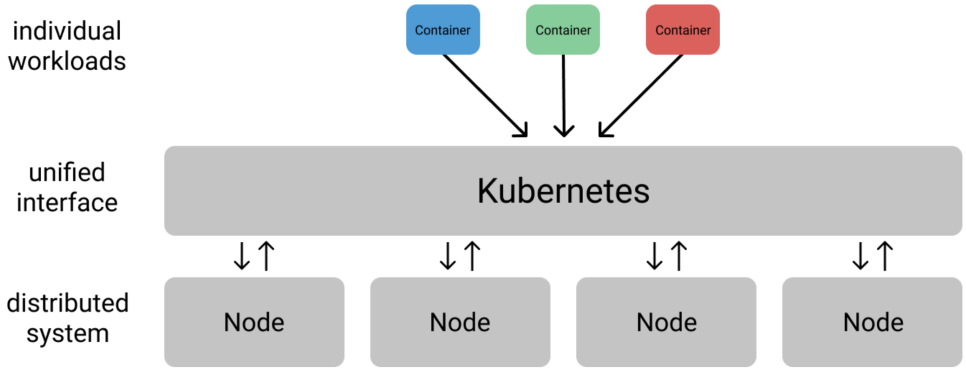
The difference in docker-compose and Kubernetes:

- docker-compose runs on a single machine
- K8S runs on multiple machines (aka distributed system)

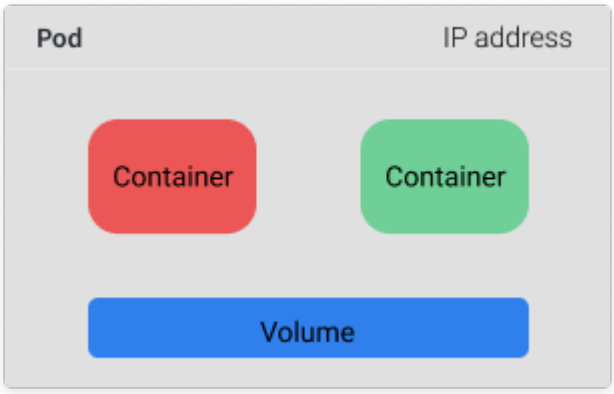


Notes and highlights from https://www.jeremyjordan.me/kubernetes/

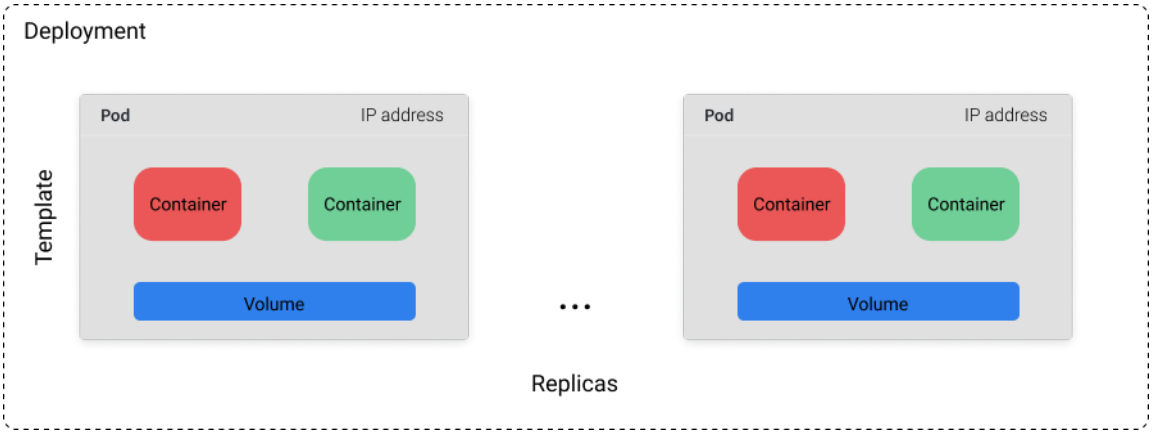
- Kubernetes will detect when the actual state of the system doesn't meet these expectations and it will intervene on your behalf to fix the problem. This enables our systems to be **self-healing**
  and react to problems without the need for human intervention.
- Each Kubernetes object has (1) a *specification* in which you provide the desired state and (2) a *status*
  which reflects the current state of the object.
- Kubernetes provides a *unified* interface for interacting with this cluster such that you don't have to worry about communicating with each machine individually.



- Kubernetes objects can be defined using either YAML or JSON files; these files defining objects are commonly referred to as *manifests*
- Kubernetes components
  1. **Pod**: The **Pod** object is the fundamental building block in Kubernetes, comprised of **one or more (tightly related) containers, a shared networking layer, and shared filesystem volumes**.



2. **Deployment:** A **Deployment** object encompasses a collection of pods defined by a template and a replica count



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-model-serving
  labels:
    app: ml-model
spec:
  replicas: 10
  selector:
    matchLabels:
      app: ml-model
  template:
    metadata:
      labels:
        app: ml-model
    spec:
      containers:
      - name: ml-rest-server
        image: ml-serving:1.0
        ports:
        - containerPort: 80
```

How many Pods should be running?

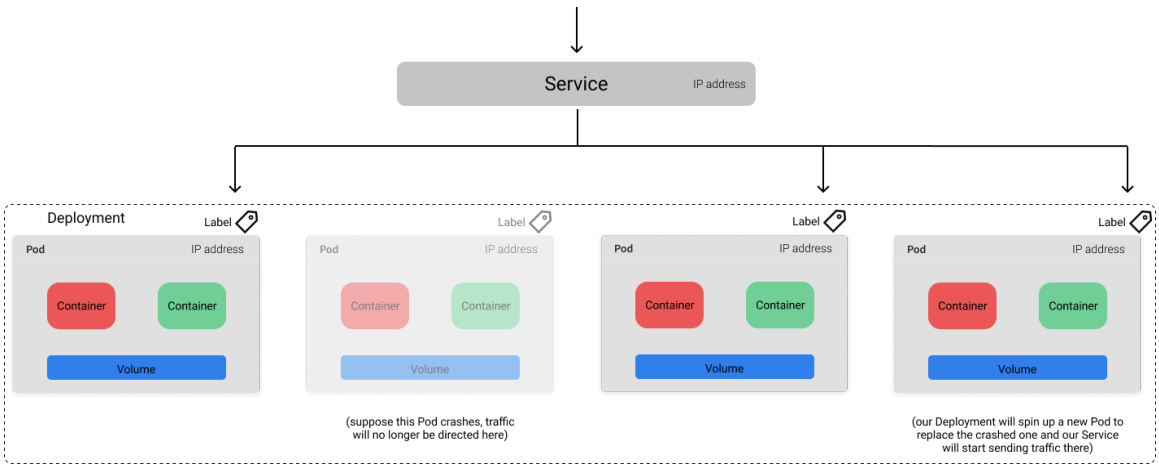
How do we find Pods that belong to this Deployment?

What should a Pod look like?

Add a label to the Pods so our Deployment can find the Pods to manage.

What containers should be running in the Pod?

3. **Service**: A Kubernetes Service provides you with a stable endpoint that can be used to direct traffic to the desired Pods even as the exact underlying Pods change due to updates, scaling, and failures. Services know which Pods they should send traffic to based on *labels* (key-value pairs) which we define in the Pod metadata.



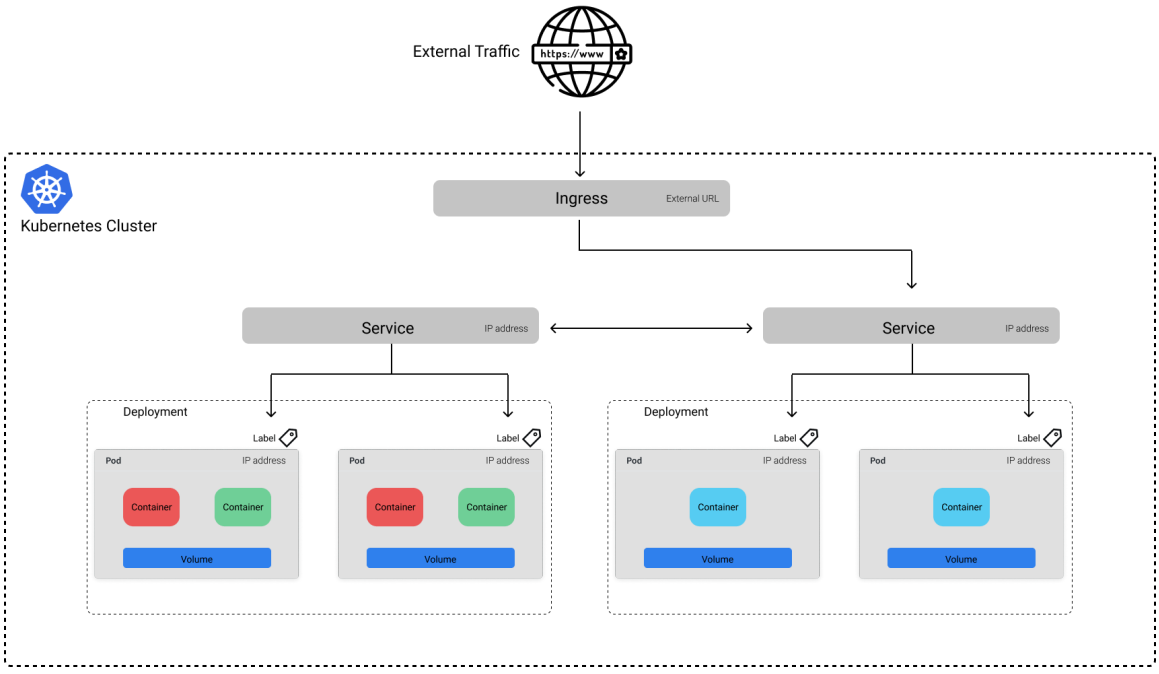
```yaml
apiVersion: v1
kind: Service
metadata:
  name: ml-model-svc
  labels:
    app: ml-model
spec:
  type: ClusterIP
  selector:
    app: ml-model
  ports:
  - protocol: TCP
    port: 80
```

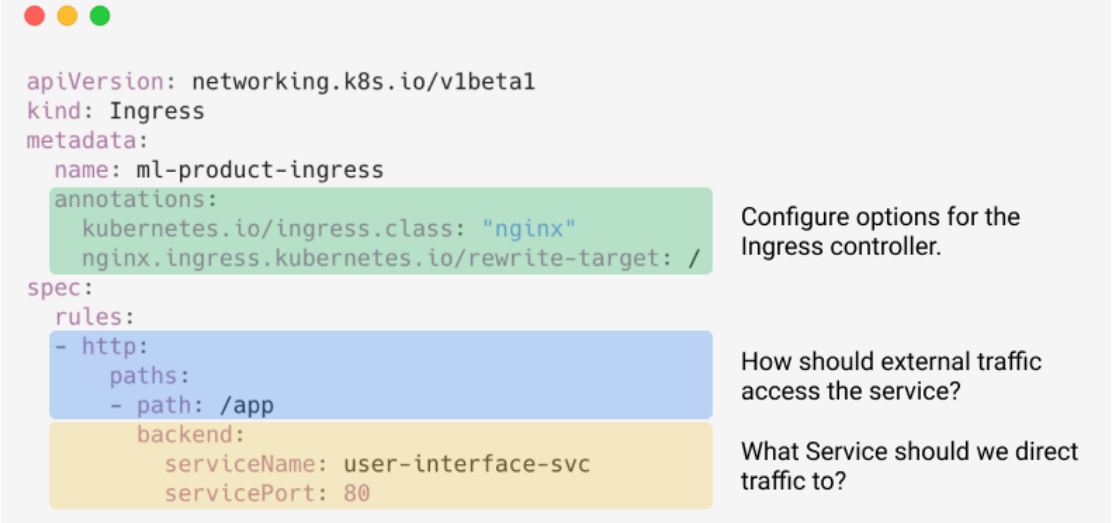
How do we want to expose our endpoint?

How do we find Pods to direct traffic to?

How will clients talk to our Service?

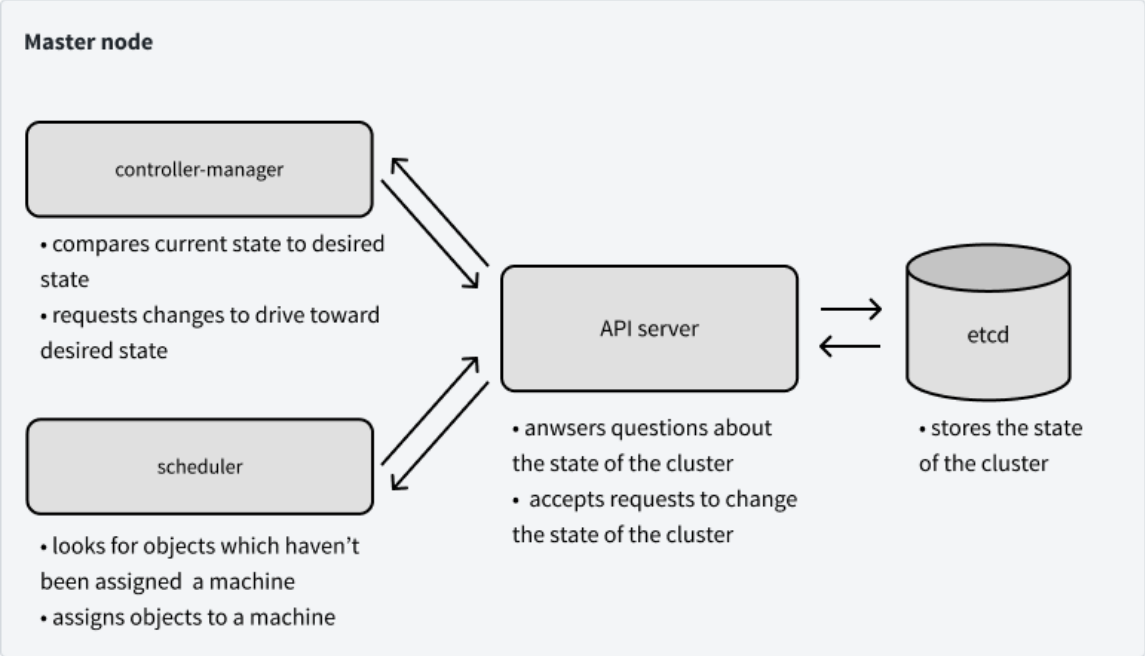
4. **Ingress**: To make a component (such as UI publicly available and keep others unavailable through endpoints [e.g. ML model])



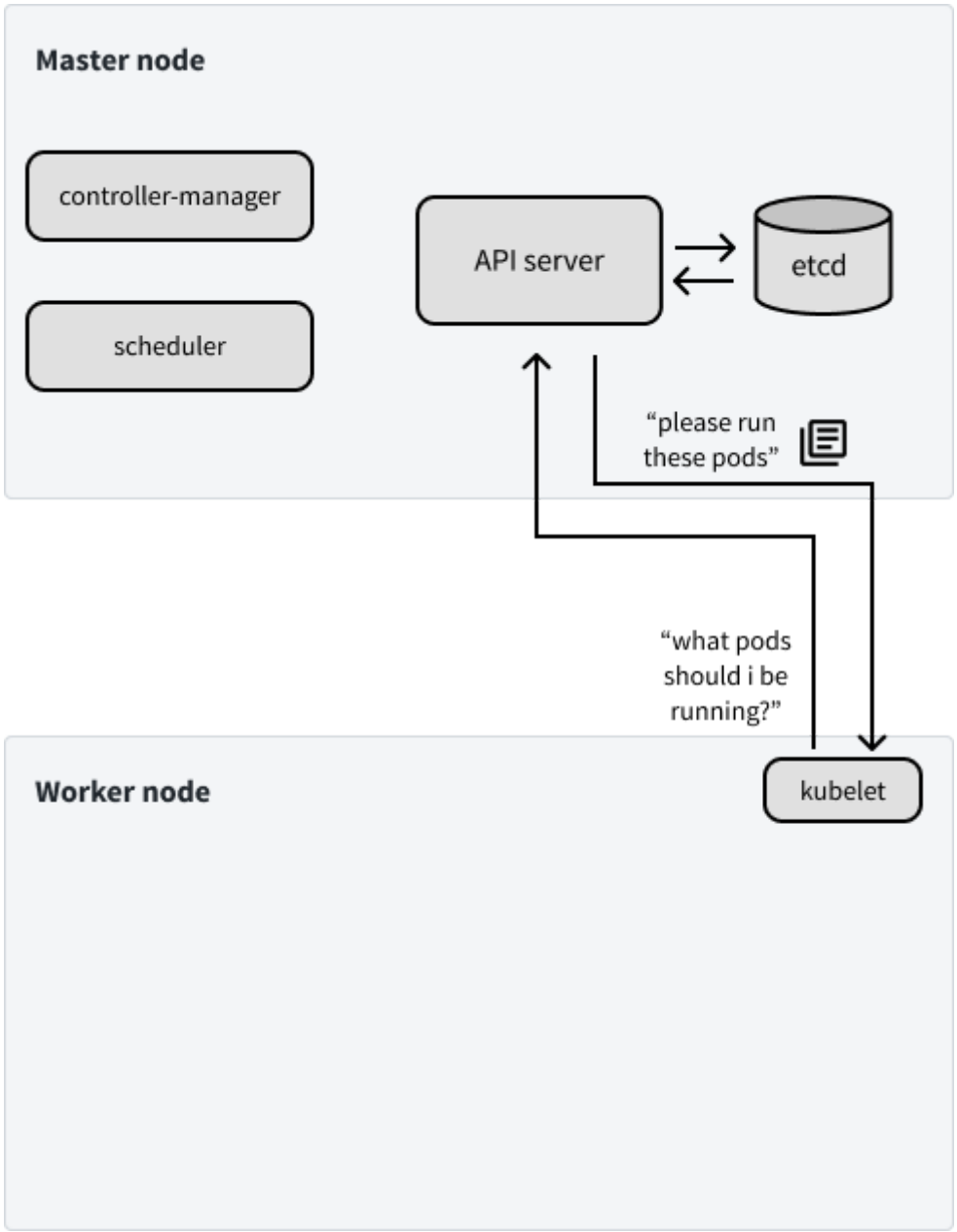


5. **Job: or a task.** For example, suppose we want to retrain our model daily based on the information collected from the previous day. Each day, we want to spin up a container to execute a predefined workload (eg. a `train.py` script) and then shut down when the training finishes.

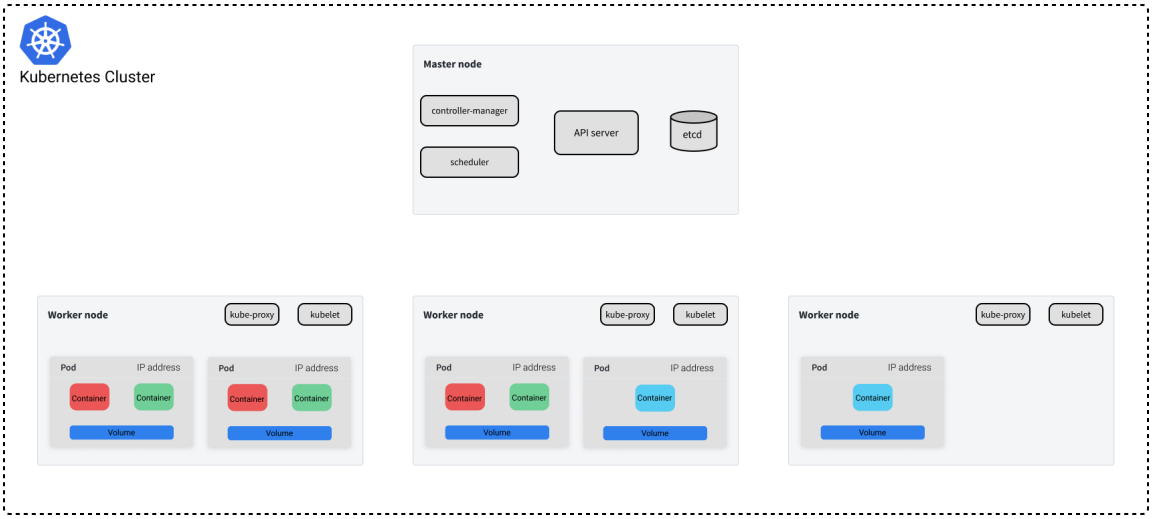
- K8S control plane
  - the *state* of our cluster is stored in **etcd**, a distributed key-value store.
  - **scheduler** is in charge of determining where objects should be run
  - **controller-manager** monitors the state of a cluster through the API server to see if the current state of the cluster aligns with our desired state
    - defined by a collection of **controllers**, each of which are responsible for managing objects of a specific resource type on the cluster



- worker node control plane:
  - The **kubelet** acts as a node's "agent" which communicates with the API server to see which container workloads have been assigned to the node. It is then responsible for spinning up pods to run these assigned workloads. When a node first joins the cluster, kubelet is responsible for announcing the node's existence to the API server so the scheduler can assign pods to it.
  - **kube-proxy** enables containers to be able to communicate with each other across the various nodes on the cluster. This component handles all the networking concerns such as how to forward traffic to the appropriate pod.



- Overall picture


Kubernetes Cluster
Master node
controller-manager
scheduler
API server
etcd
Worker node
kube-proxy
kubelet
Pod
IP address
Container
Container
Volume
Pod
IP address
Container
Container
Volume
Worker node
kube-proxy
kubelet
Pod
IP address
Container
Container
Volume
Pod
IP address
Container
Volume
Worker node
kube-proxy
kubelet
Pod
IP address
Container
Volume