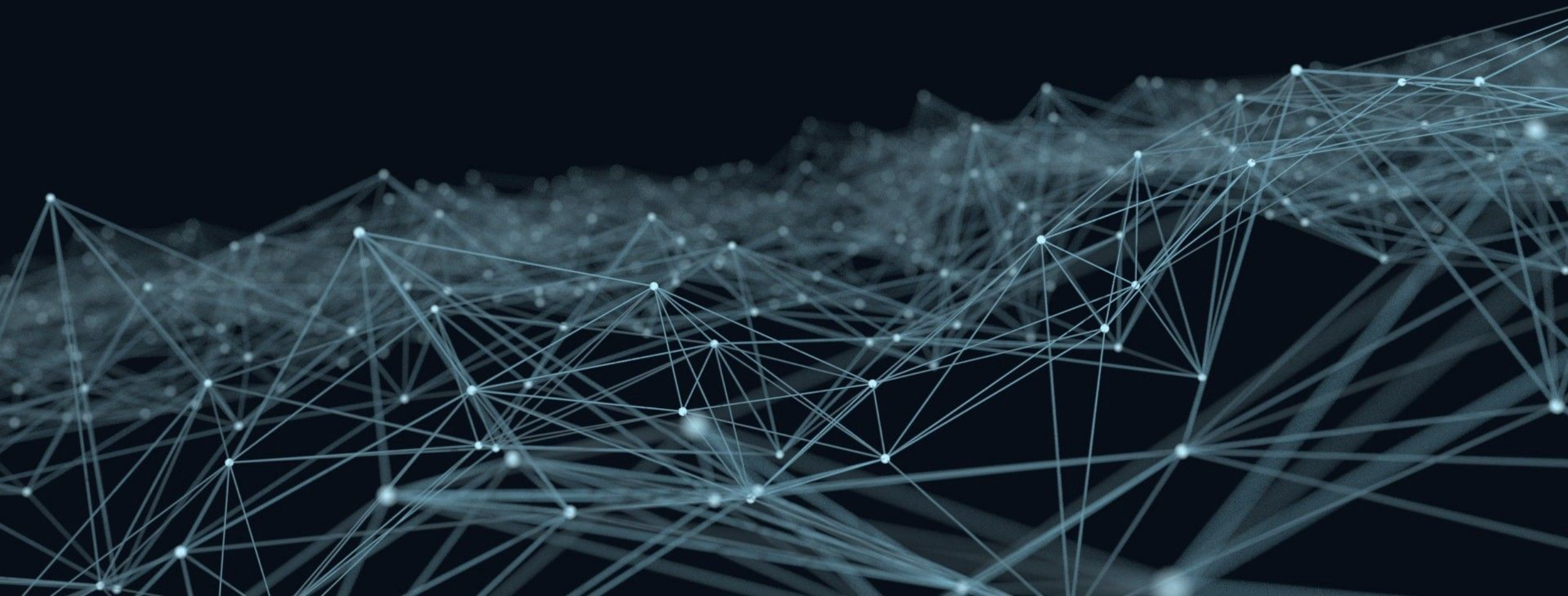
Microservices Architecture

Orchestration with Kubernetes

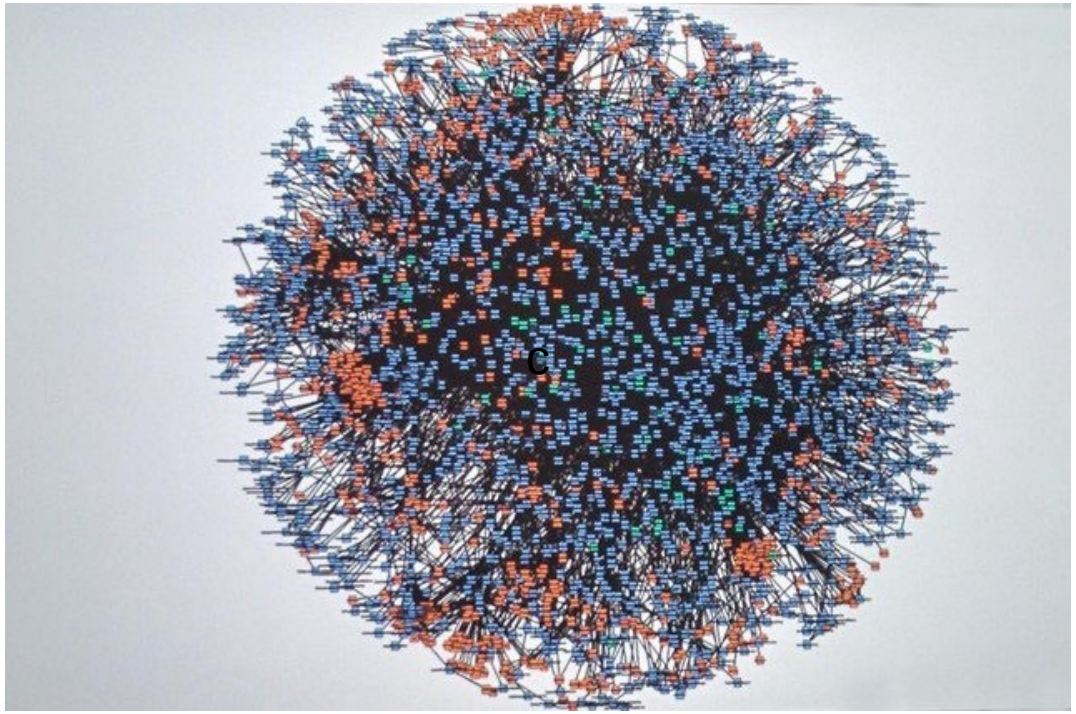


The Operations Challenge

- In another module, we looked at containerization from a development perspective
- In production, we have to be concerned with
  - Coordinating the activity of possibly thousand of containers that need to work together
  - Creating and maintaining connections between containers
  - Ensure the whole system operates well enough to meet Service Level Agreements (SLAs)
- We need to deal with non-functional requirements
  - Loading, throughput, stress, response times
  - Disaster recovery
  - Security
- The lack of an effective way to do this was a major impediment to the deployment of microservice based applications

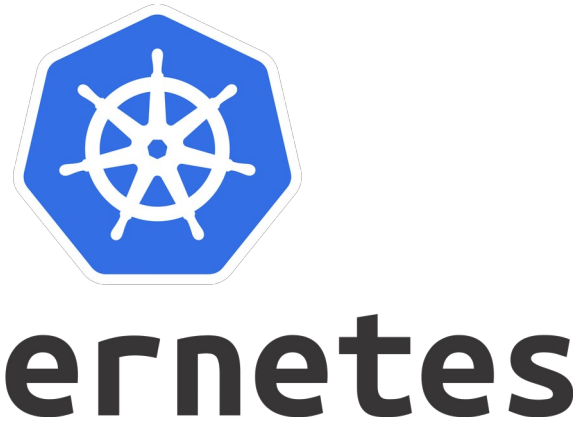
Site Reliability Engineering

- Practices designed to ensure large systems are operational
- Continuously checking for potential problems
- Manages a set of mitigation responses to react to problems
- Recent examples
  - Rogers Canada 2022 network failure
  - Facebook October 2021 upgrade failure
  - Check out risks.org
- As applications scale, this becomes increasingly difficult

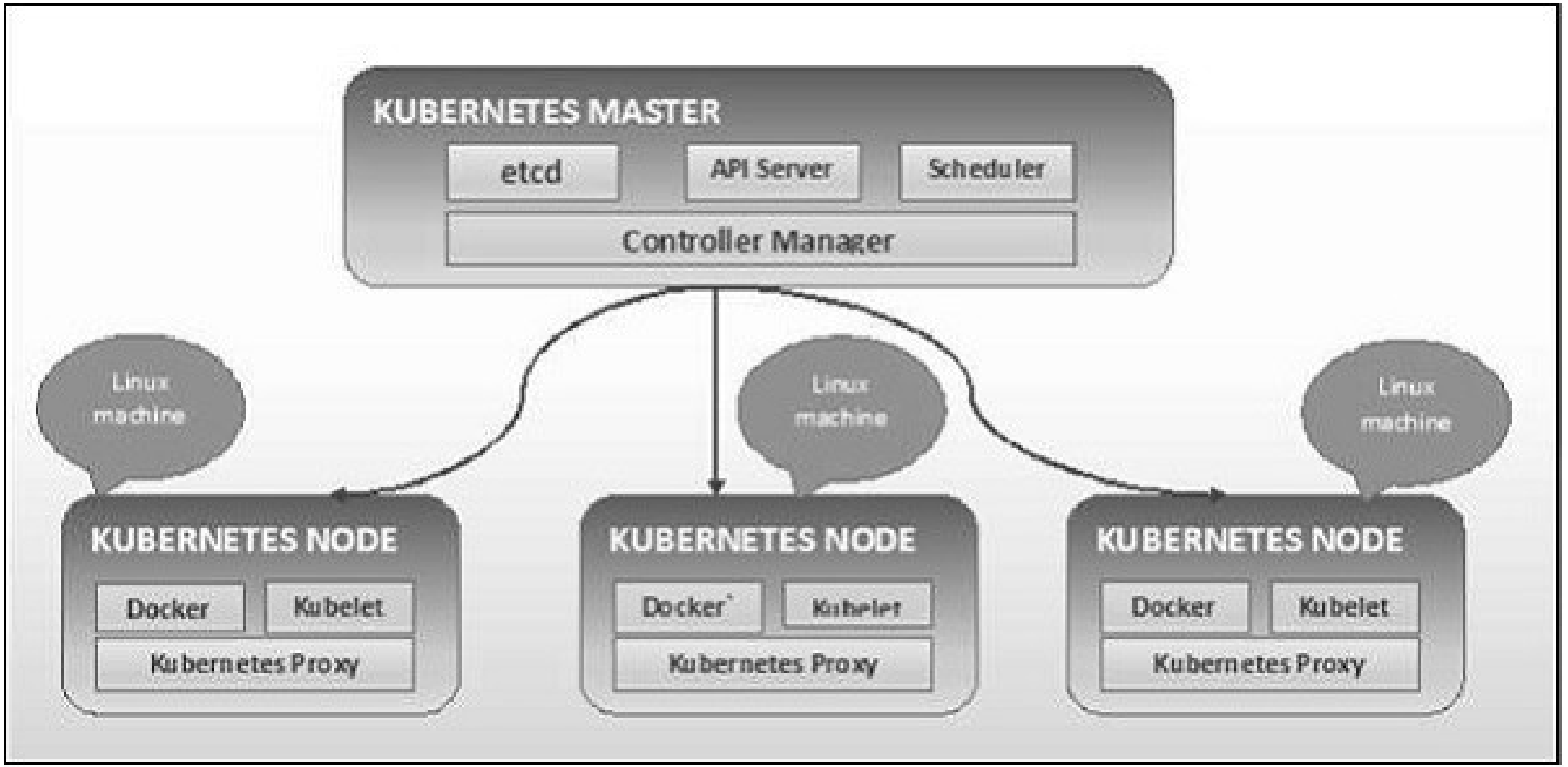


Kubernetes

- Kubernetes is a container orchestration manager
  - Not the only manager
  - Docker Swarm does the same
  - Kubernetes is “industrial strength”
- Orchestration:
  - Manages “clusters” of containers
  - Provides service discovery
  - Manages scaling and failover
  - Works at the Ops level
  - Infrastructure as Code

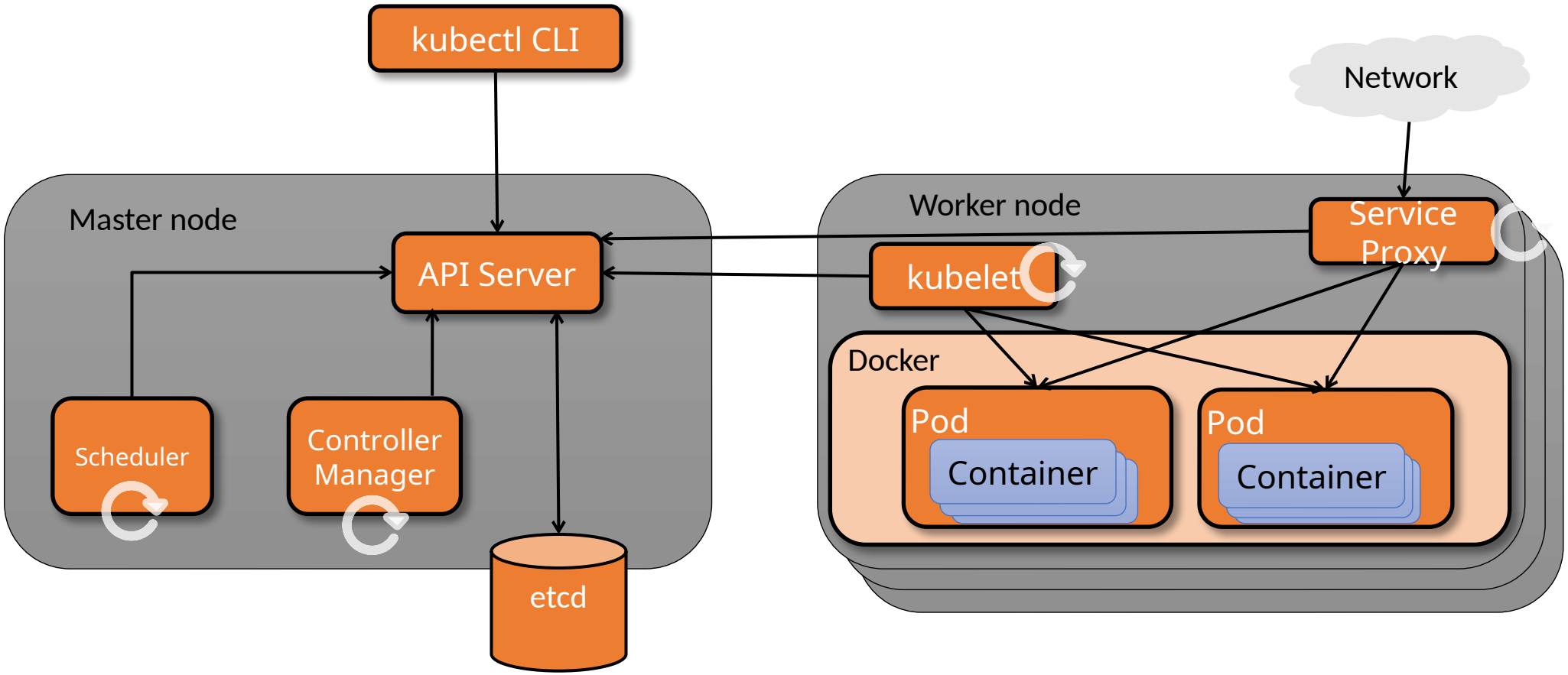


Kubernetes Cluster



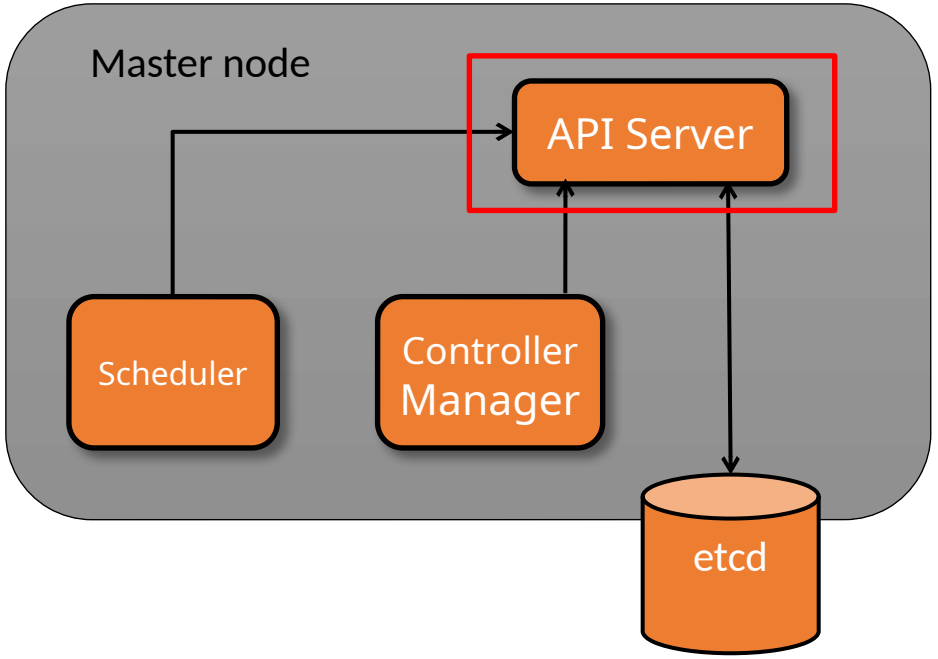
Kubernetes Architecture

- Kubernetes nodes can be physical hosts or VM's running a container-friendly Linux



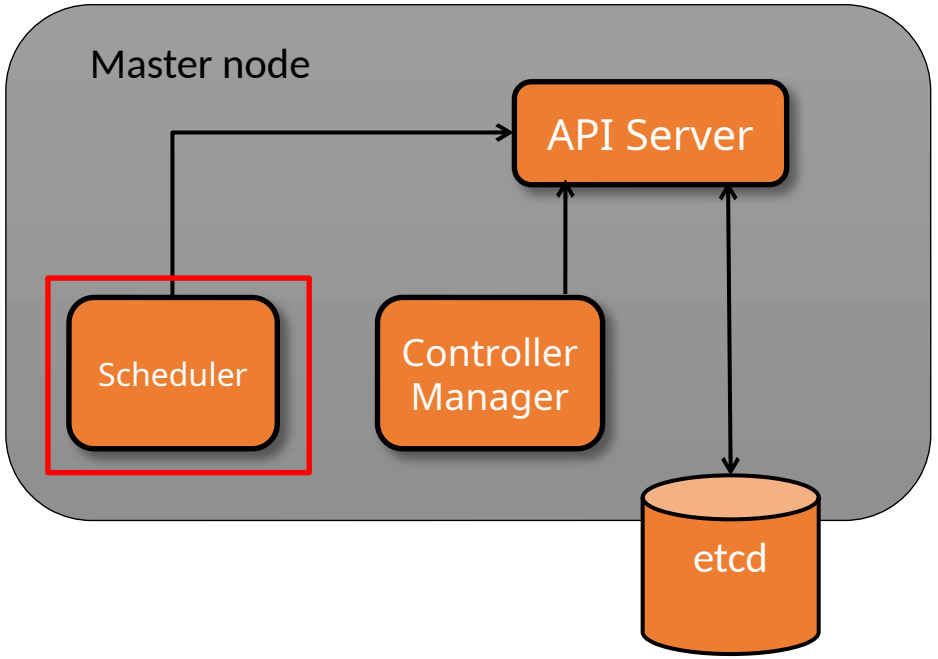
Master Node Components

- API Server (kube-apiserver): exposes the Kubernetes REST API, and can be scaled horizontally



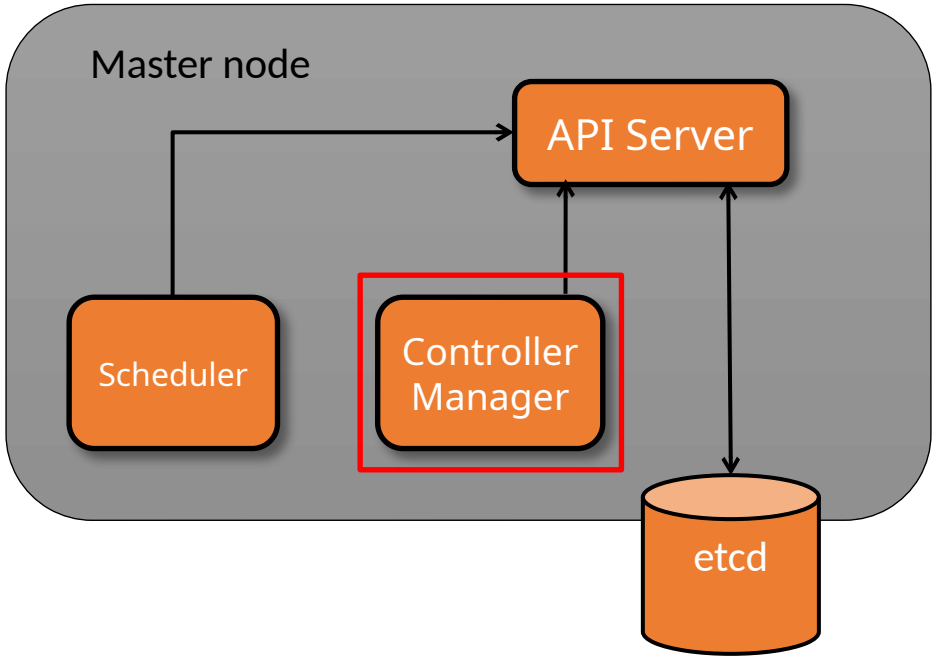
Master Node Components

- Scheduler (kube-scheduler): selects nodes for newly created pods to run on



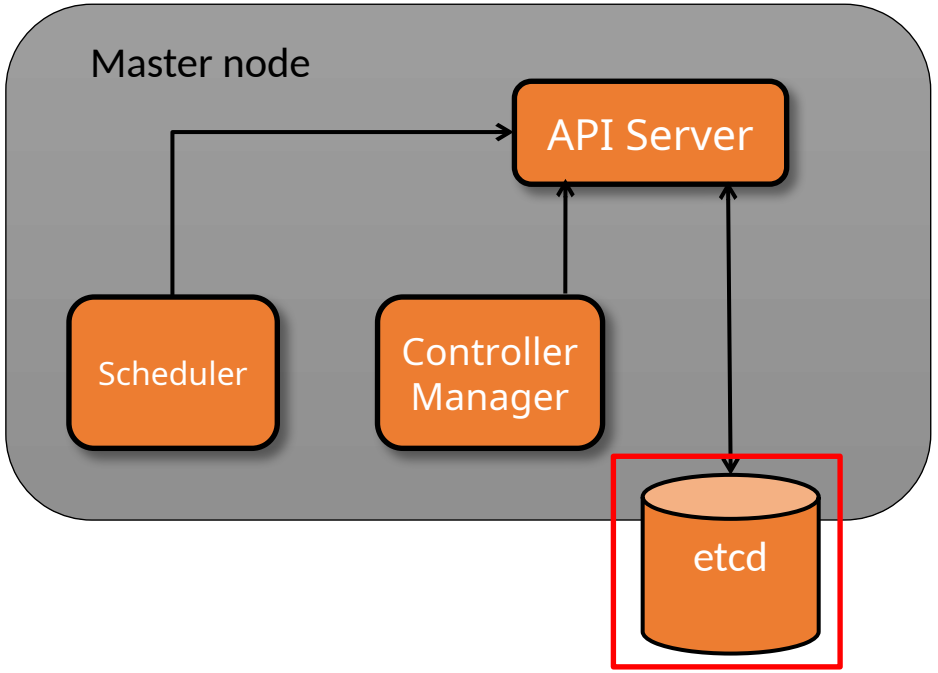
Master Node Components

- Controller manager (kube-controller-manager): runs background controller processes for the system to enforce declared object states, e.g. Node Controller, Replication Controller



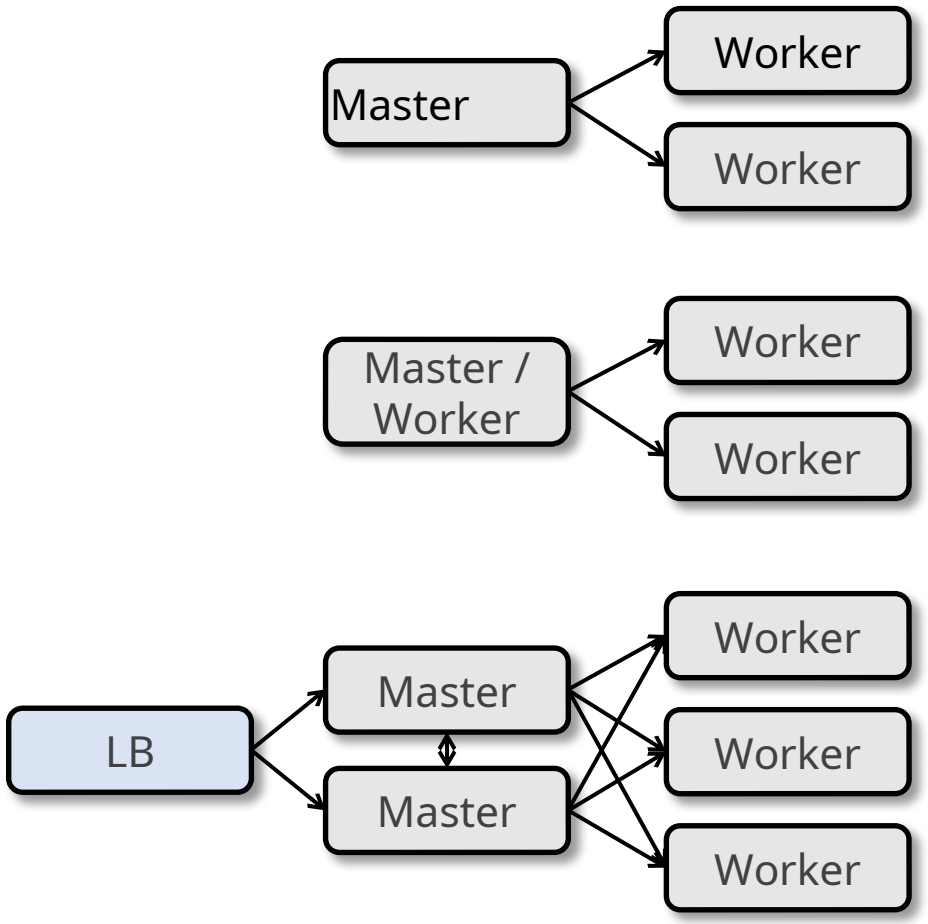
Master Node Components

- Persistent data store (etcd): all K8s system data is stored in a distributed, reliable key-value store. etcd may run on separate nodes from the master



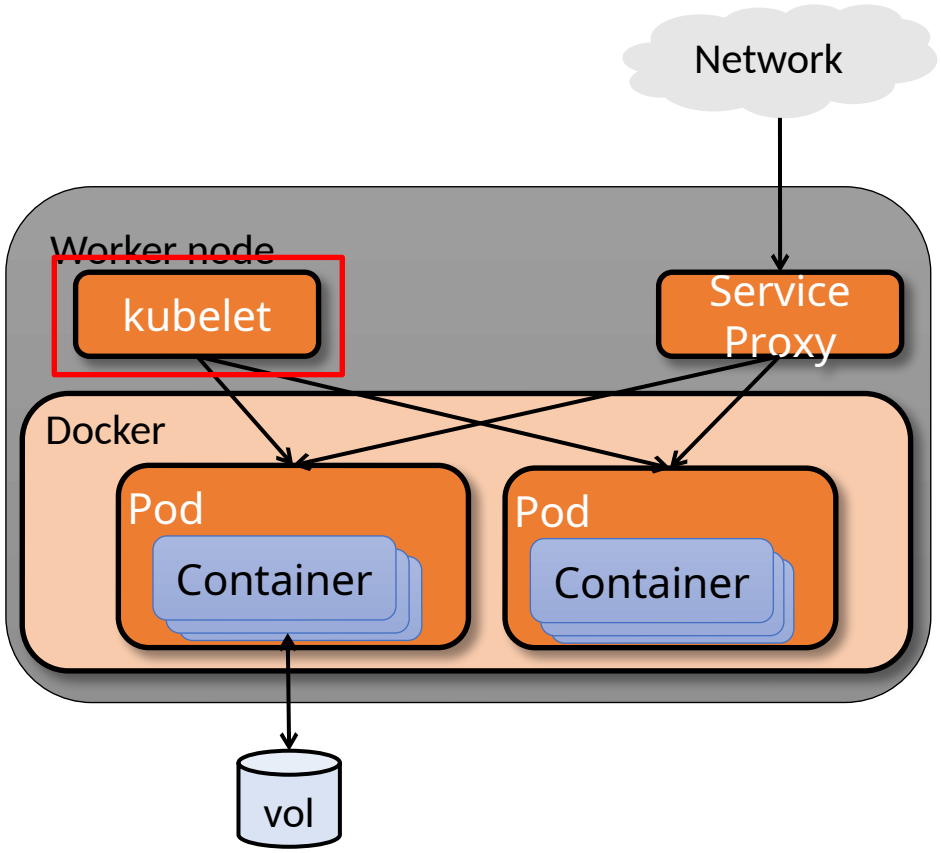
Master Node Deployment Options

- Simple cluster has a single master node
  - At small scale, master may also be a worker node
- Cluster of master node replicas behind a loadbalancer
  - Kube-apiserver and etcd scale out horizontally
  - Kube-scheduler and kube-controller-manager use master election to run single instances at a time



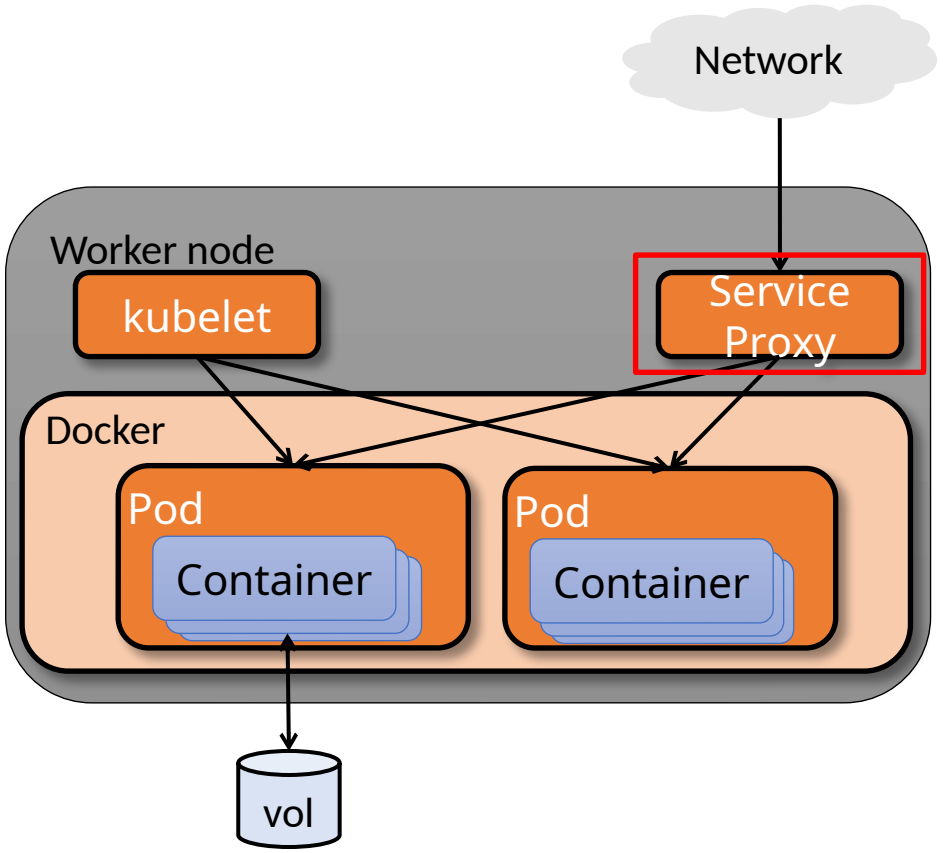
Worker Node Components

- **kubelet**: local K8s agent that is responsible for operations on the node, including
  - Watching for pod assignments
  - Mounting pod required volumes
  - Running a pod's containers
  - Executing container liveness probes
  - Reporting pod status to system
  - Reporting node status to system



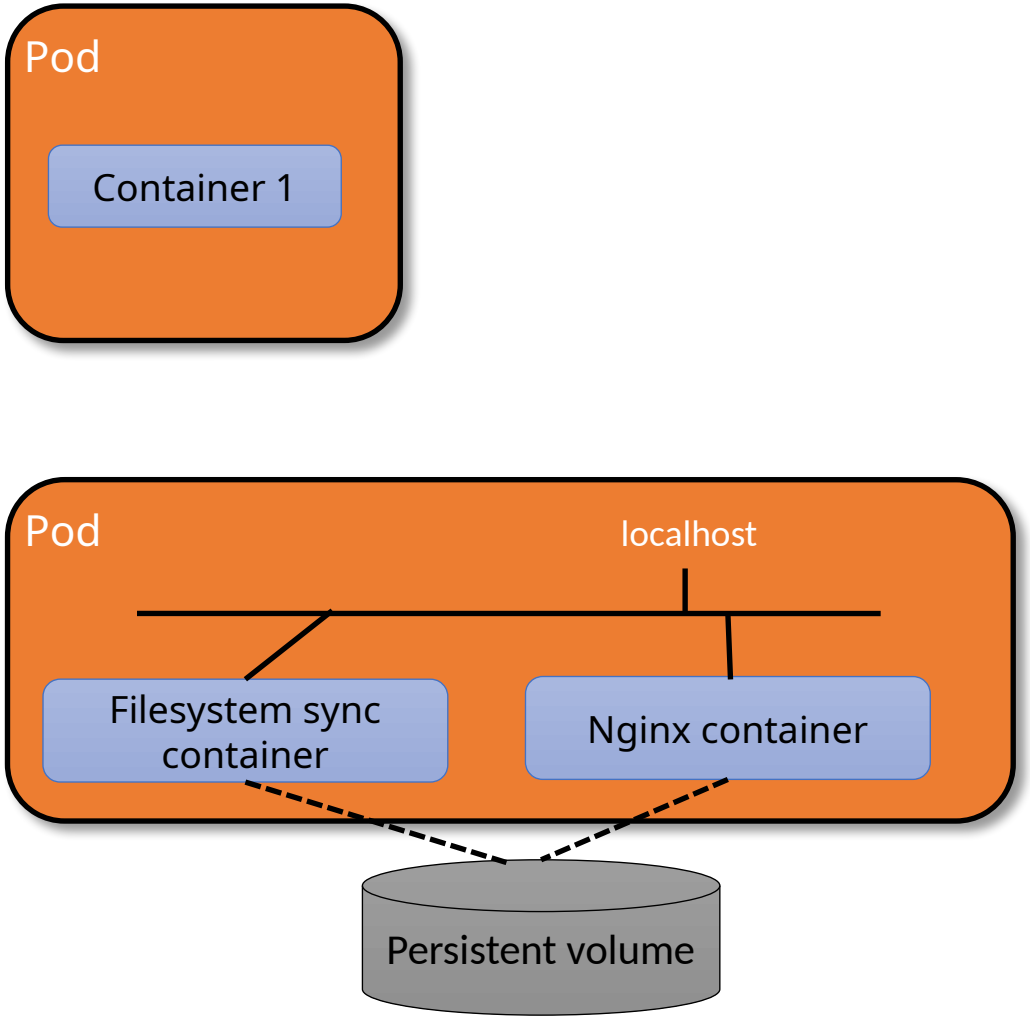
Worker Node Components

- Service proxy (kube-proxy): enables K8s service abstractions by maintaining host network rules and forwarding connections
- Docker: runs the containers



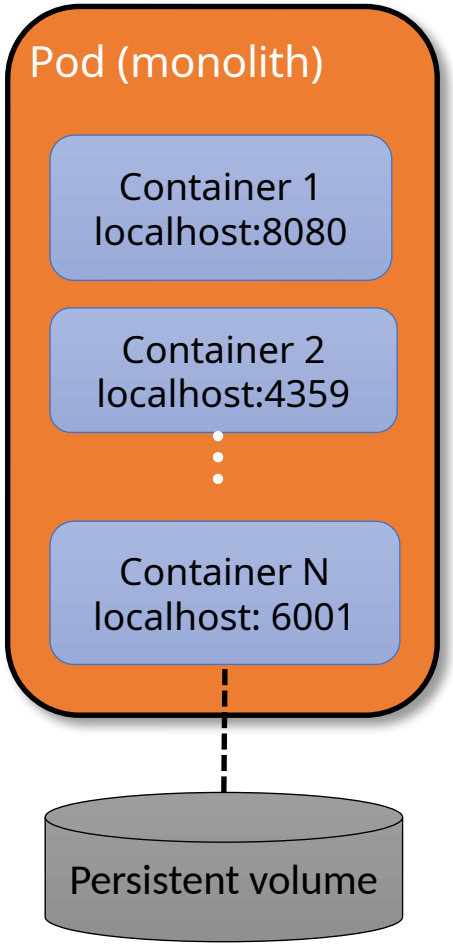
Kubernetes Pod

- Basic unit of deployment is the pod, a set of co-scheduled containers and shared resources
- Pods can include more than one container, for tightly-coupled application components, e.g.
  - Sidecar containers : nginx + filesystem synchronizer to update www from git
  - Content adapter: transform data to common output standard
- Containers in a pod share network namespaces and mounted volumes



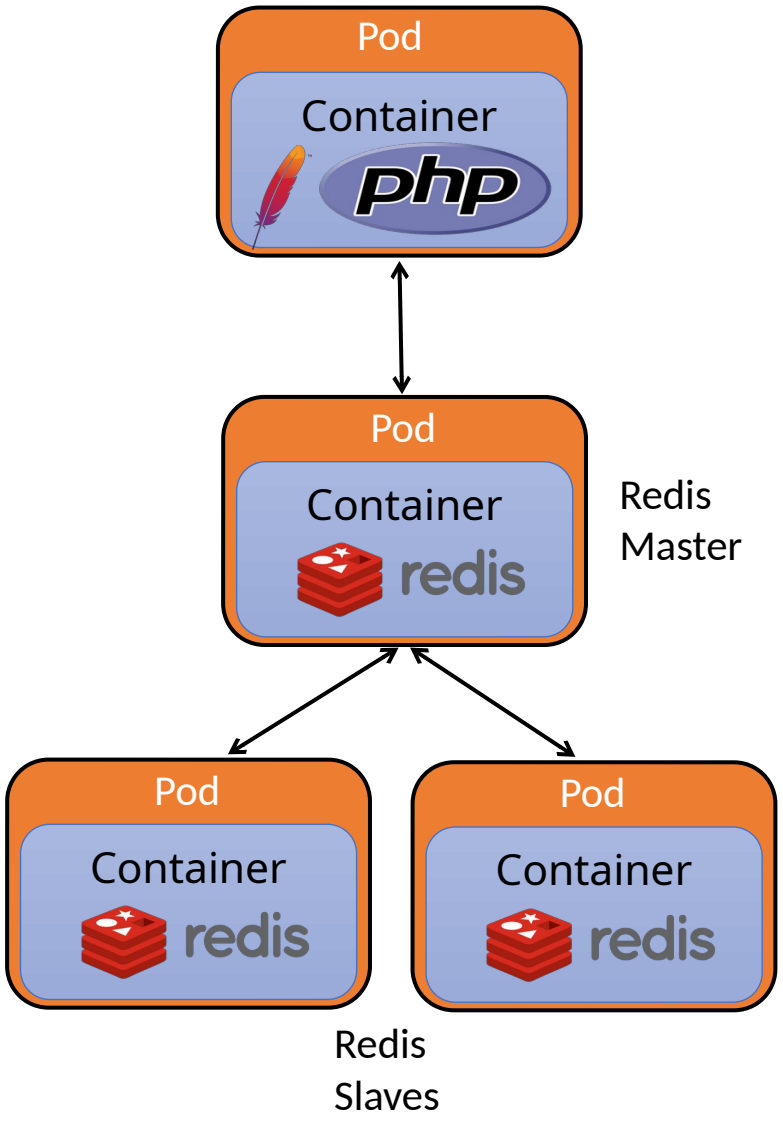
Pod Deployment

- Possible to use a single pod to run a monolithic application
- Each application process can be built as a container
- All containers can access each other's ports on localhost



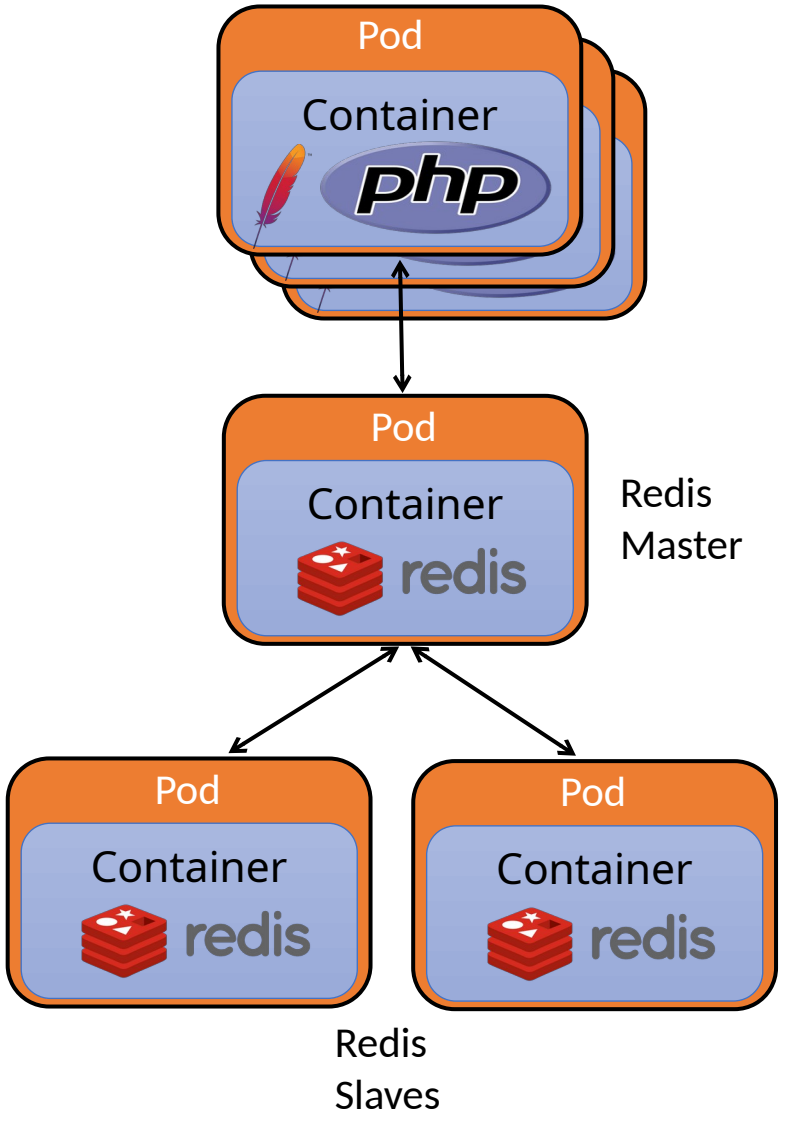
Pod Deployment

- More advanced features of the K8s system available if application built instead from assemblages of pods, e.g.
- Web tier: Apache pods
- Data tier: Redis master/slave pods



Pod Deployment

- Pods provide scale and elasticity via replication – not possible in the monolith
- Best practice: assume every pod is mortal



Defining a Pod via a Manifest File

simplepod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: test-tomcat-pod
  labels:
    app: web
spec:
  containers:
  - image: tomcat
    name: test-tomcat
    ports:
    - containerPort: 8080
```

Type of this K8s object (service, pod, deployment, …)

User defined labels on pod

Set of containers, each With name, Docker image, and resources

Port configuration for this container

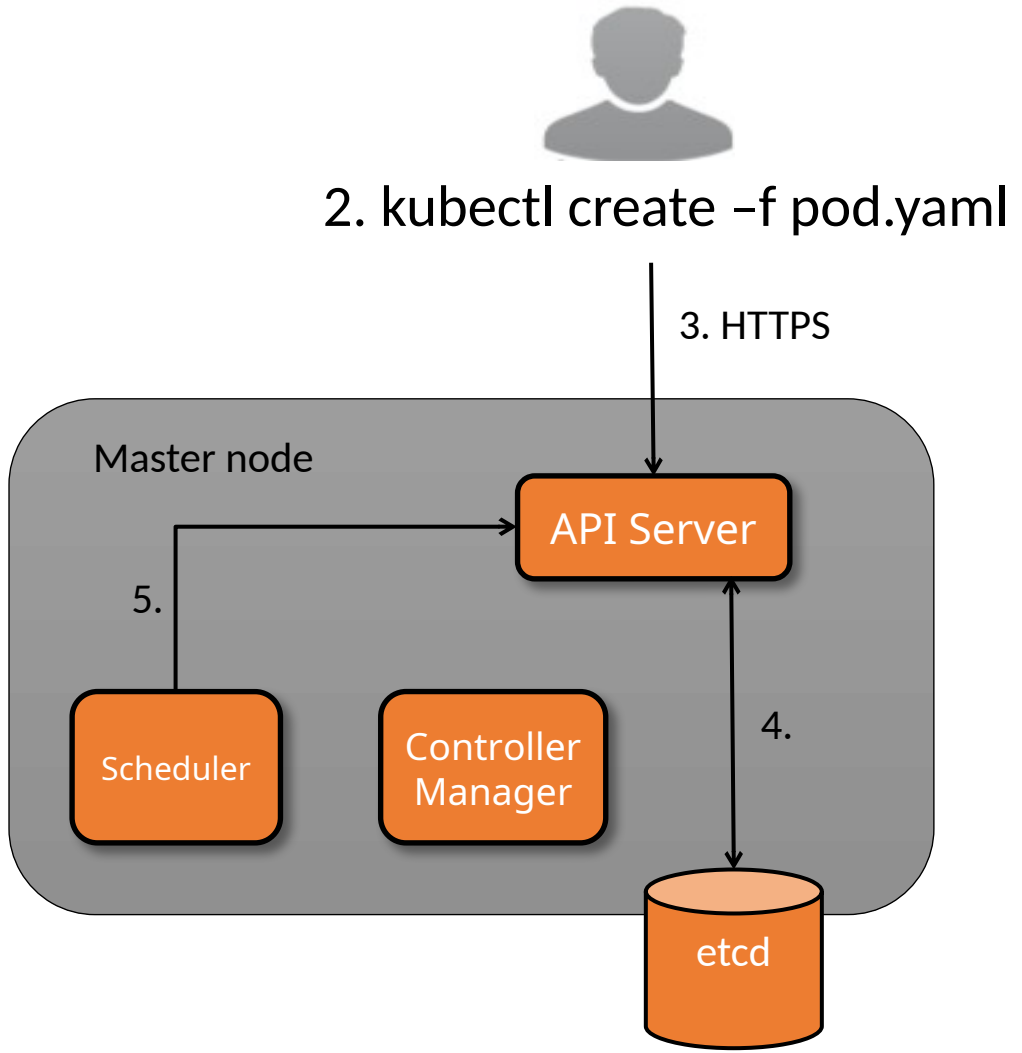
Defining a Pod with Multiple Containers

- Pod spec can contain multiple containers from different images
- Containers in pod share local network context and cluster IP for pod

```
apiVersion: v1
kind: Pod
metadata:
  name: test-tomcat-pod
spec:
  containers:
  - image: tomcat
    name: test-tomcat
    ports:
    - containerPort: 8080
  - image: mysql
    name: test-mysql
    ports:
    - containerPort: 3306
```

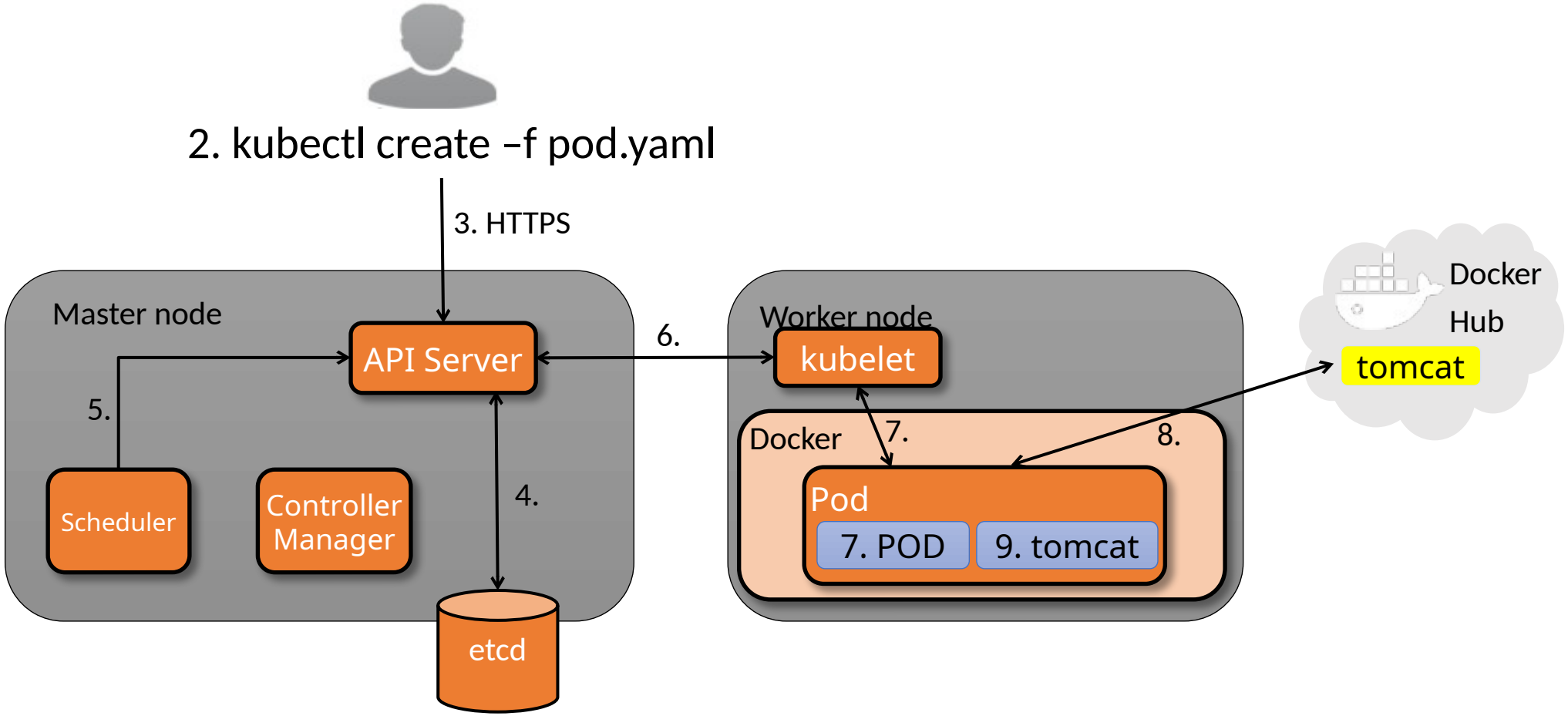
Pod Creation Process

- User writes a pod manifest file
- User requests creation of pod from manifest via CLI
- CLI tool marshals parameters into K8s RESTful API request (HTTP POST)
- kube-apiserver creates new pod object record in etcd, with no node assignment
- kube-scheduler notes new pod via API
- Selects node for pod to run on
- Updates pod record via API with node assignment



Pod Creation Process

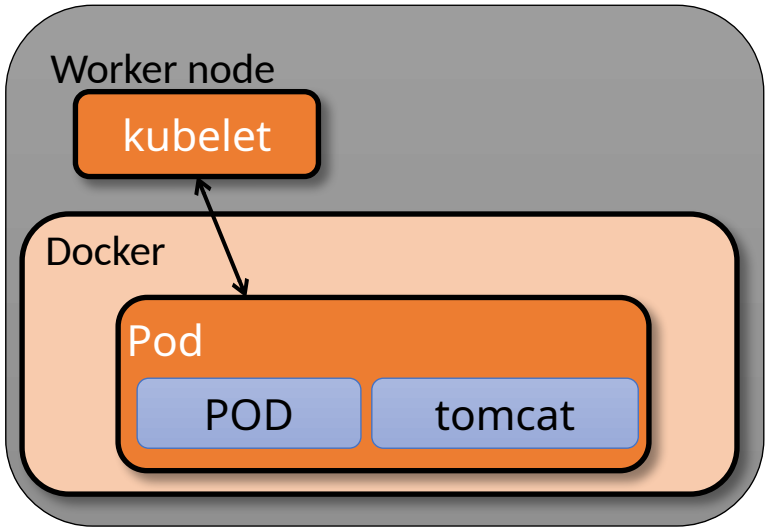
- Kubelet pulls Docker image(s) for pod workload containers
- Container(s) started and running on worker node



Pod Lifecycles

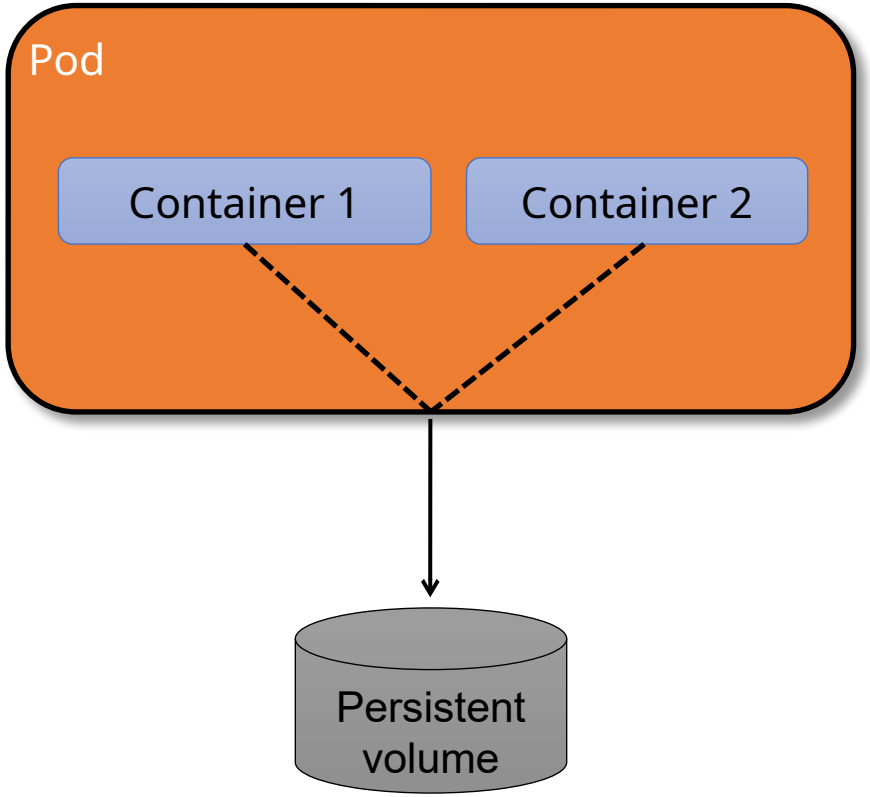
- By default, K8s Pods have an indefinite lifetime, which is not immortality
  - restartPolicy of Always by default
  - restartPolicy of Never or OnFailure also available for terminating jobs
- Node's kubelet will create and keep running containers for pods assigned to node, per the pod specs
- If a Pod container fails to start, or unexpectedly exits, kubelet will restart it
  - Can see container lifecycle events via 'kubectl describe pod <PODNAME>'
- If node is lost, its Pods are also lost – K8s will not rebind Pods to another node

```
apiVersion: v1
kind: Pod
metadata:
  name: test-tomcat-pod
  labels:
    app: web
spec:
  containers:
  - image: tomcat
    name: test-tomcat
    ports:
    - containerPort: 8080
```



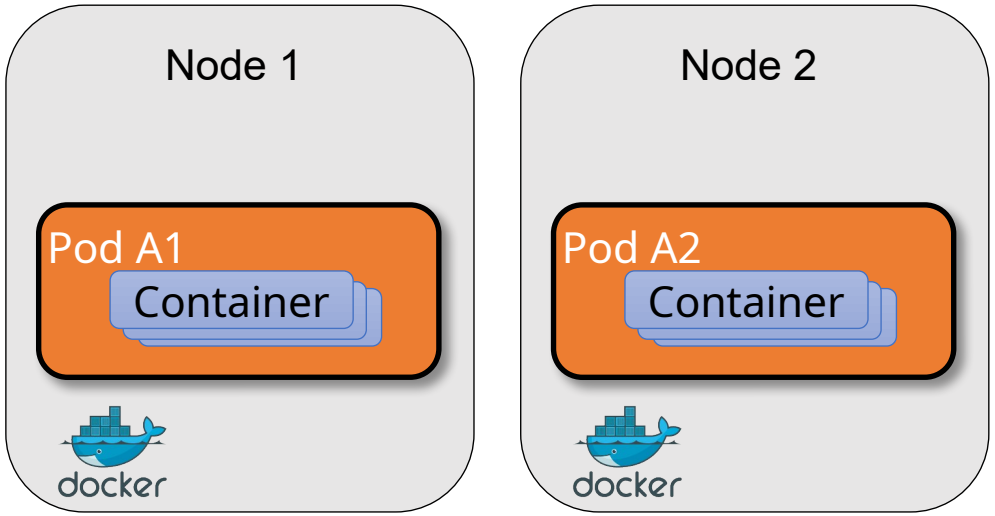
Deleting Pods

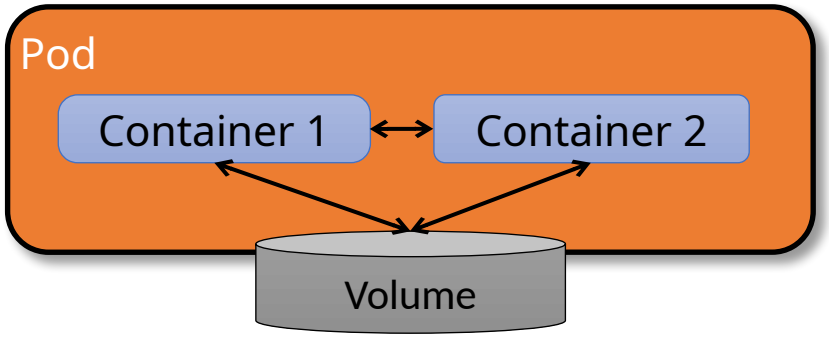
- When deleting a Pod, its containers will be removed and pod IP relinquished
- If an application needs to persist data, its pods must be configured to use persistent volumes for storage
- If a node dies, its local pods are also gone
- Best practice: use controller resources instead of managing pods directly
- Best practice: use service resources to build reliable abstraction layers for clients



Kubernetes Pods

- Pod is an application instance
  - Pods can include more than one container, for tightly-coupled application components
  - Containers in the same Pod share networking and storage resources
- Kubernetes handles efficient placement of Pods across available Nodes

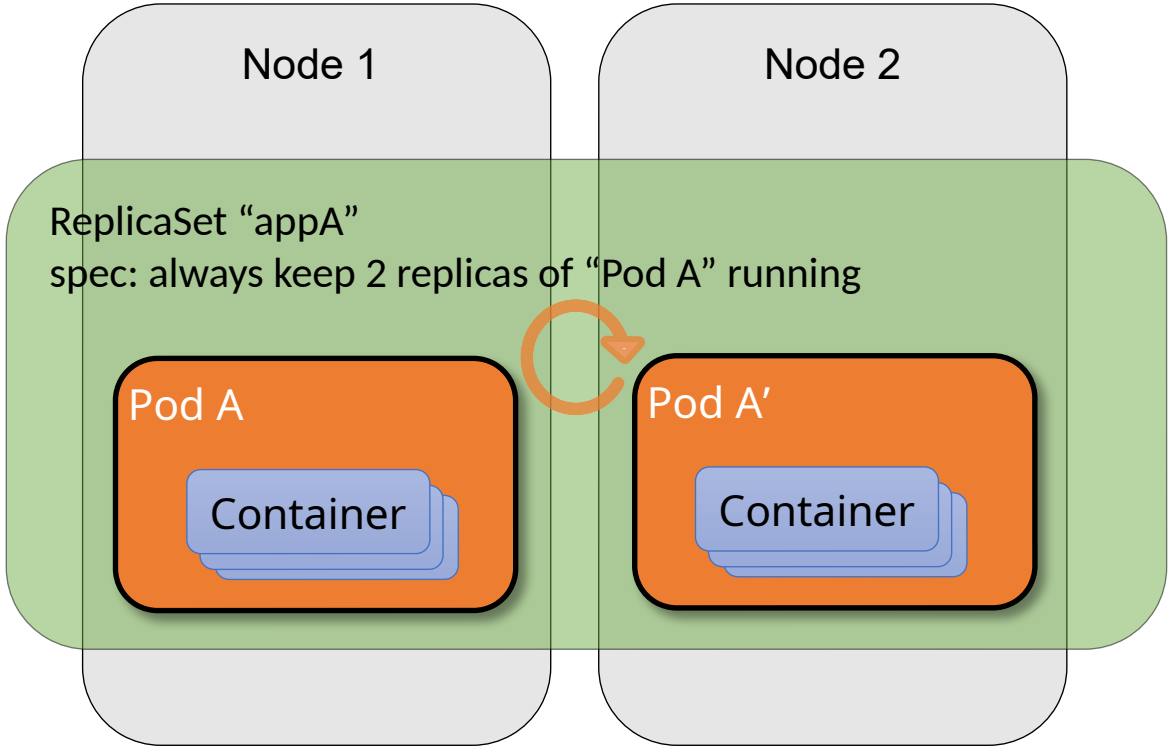




Application Patterns

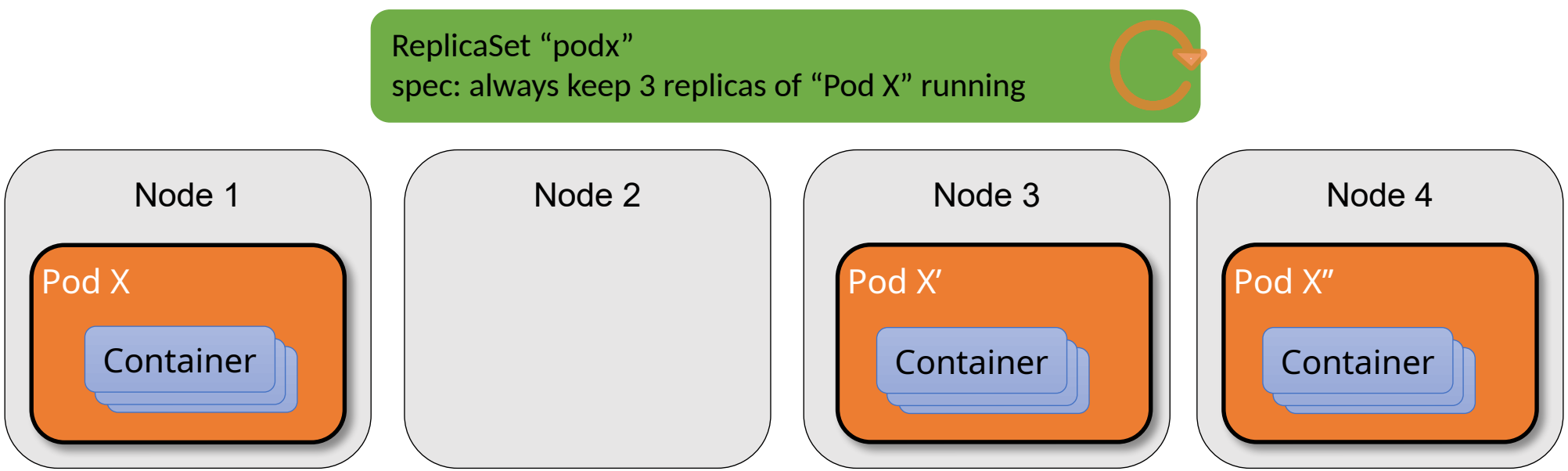
- K8s controller creates and manages Pods according to different application
  - **ReplicaSets** manage sets of replicas of stateless workloads to ensure availability
  - **StatefulSets** manage stateful workloads on stable storage to ensure consistency
  - **DaemonSets** manage workloads that must run on every node, or set of nodes
- Jobs manage parallel batch processing workloads

Controller example: ReplicaSet



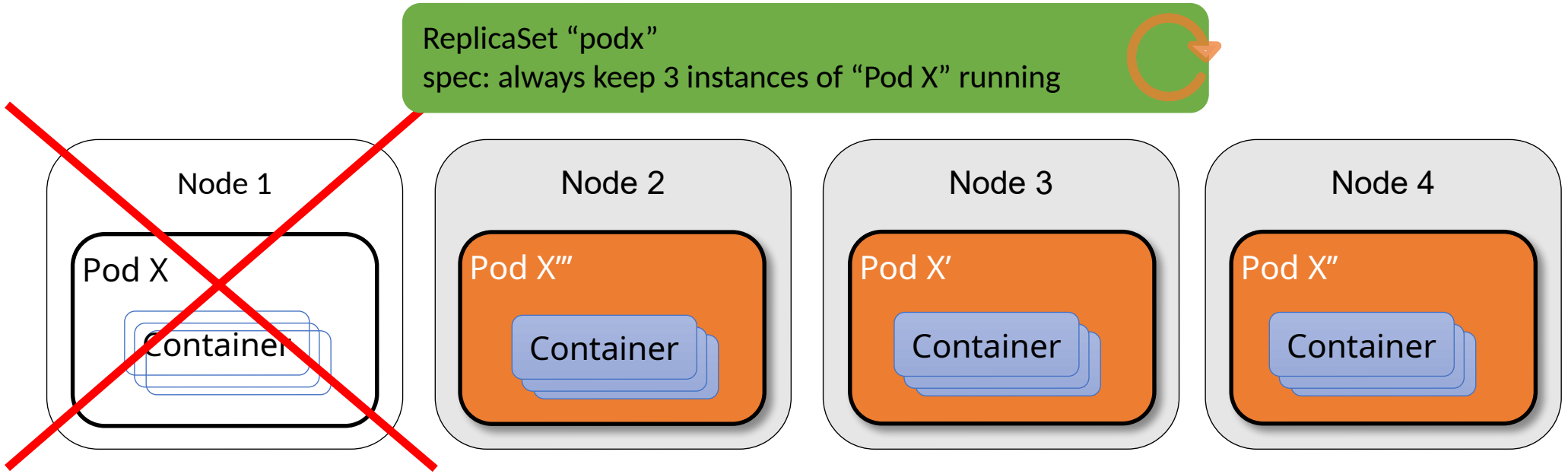
ReplicaSets

- ReplicaSet configuration specifies how many instances of given Pod exist
- ReplicaSet used for web applications, mobile back-ends, APIs

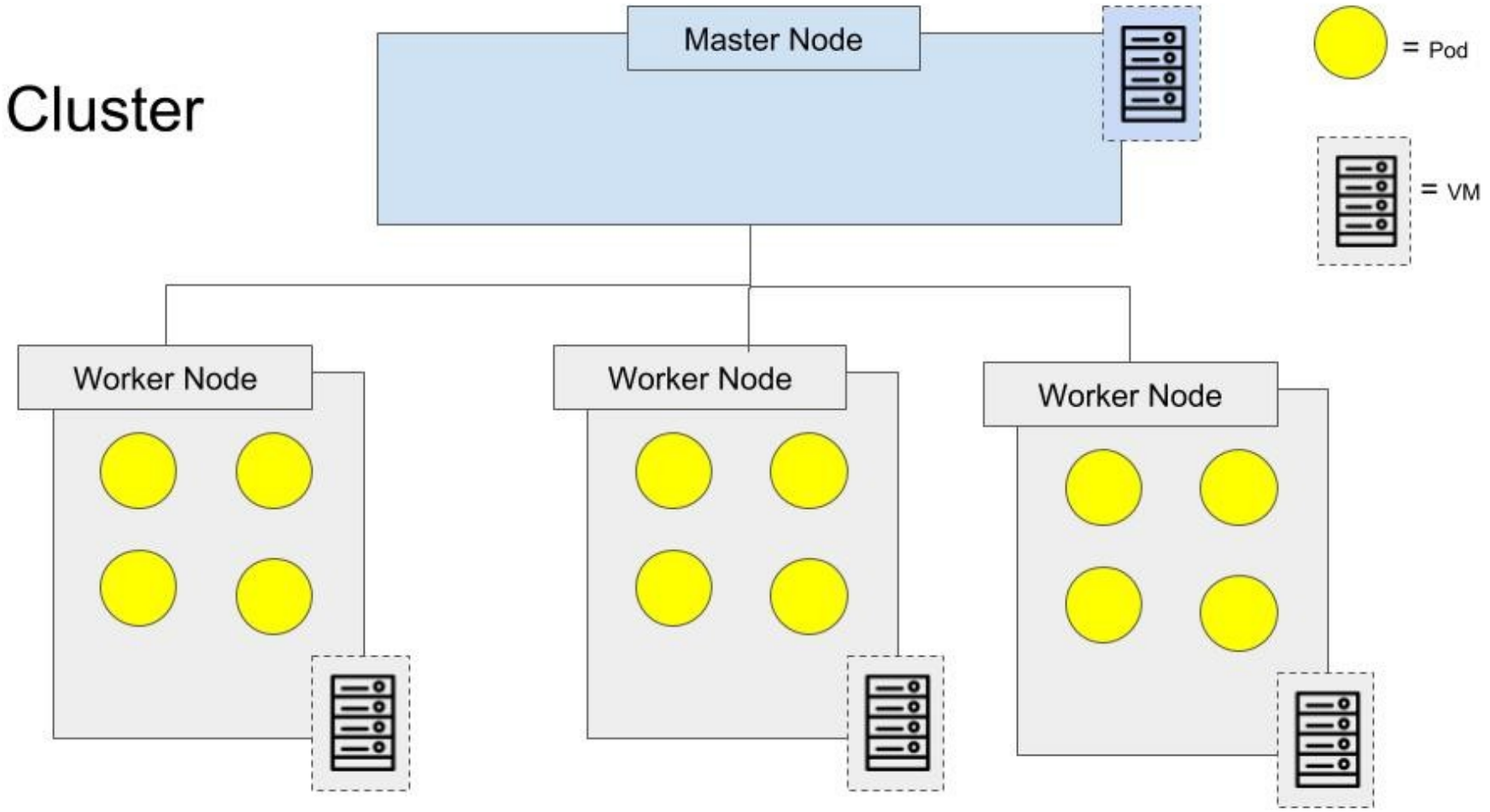


Replication Ensures Application Availability

- When a Node fails, its Pods are lost
- K8s system manages the state of the ReplicaSet back to the declared configuration
- Changing the configuration will result in management to new state, e.g. scale out

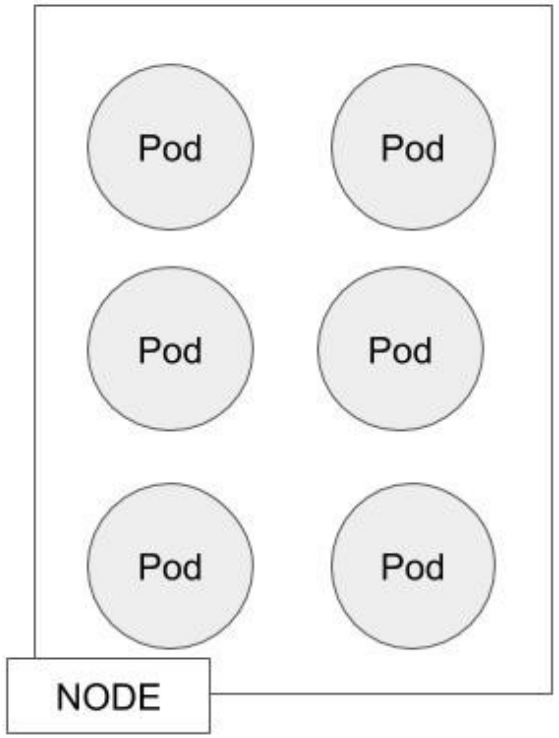


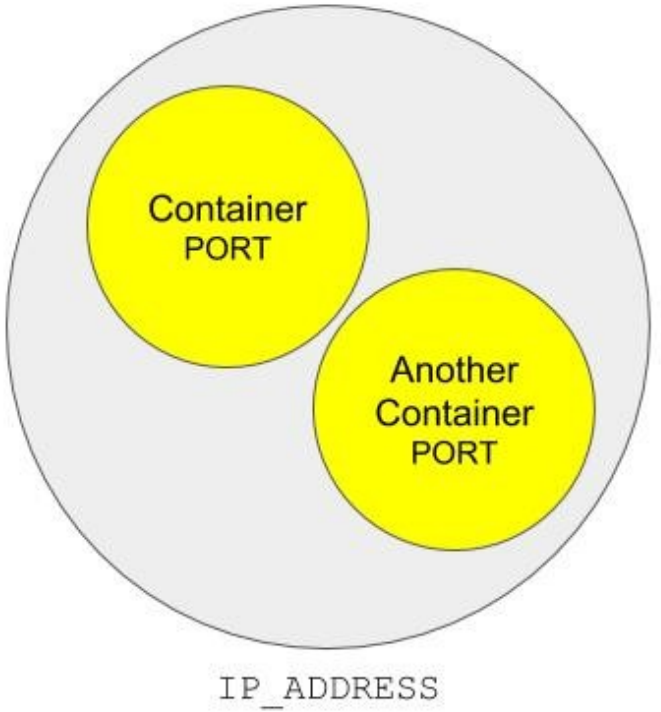
The Essential Architecture Review



The Essential Architecture Review

Pod





Kubernetes Services

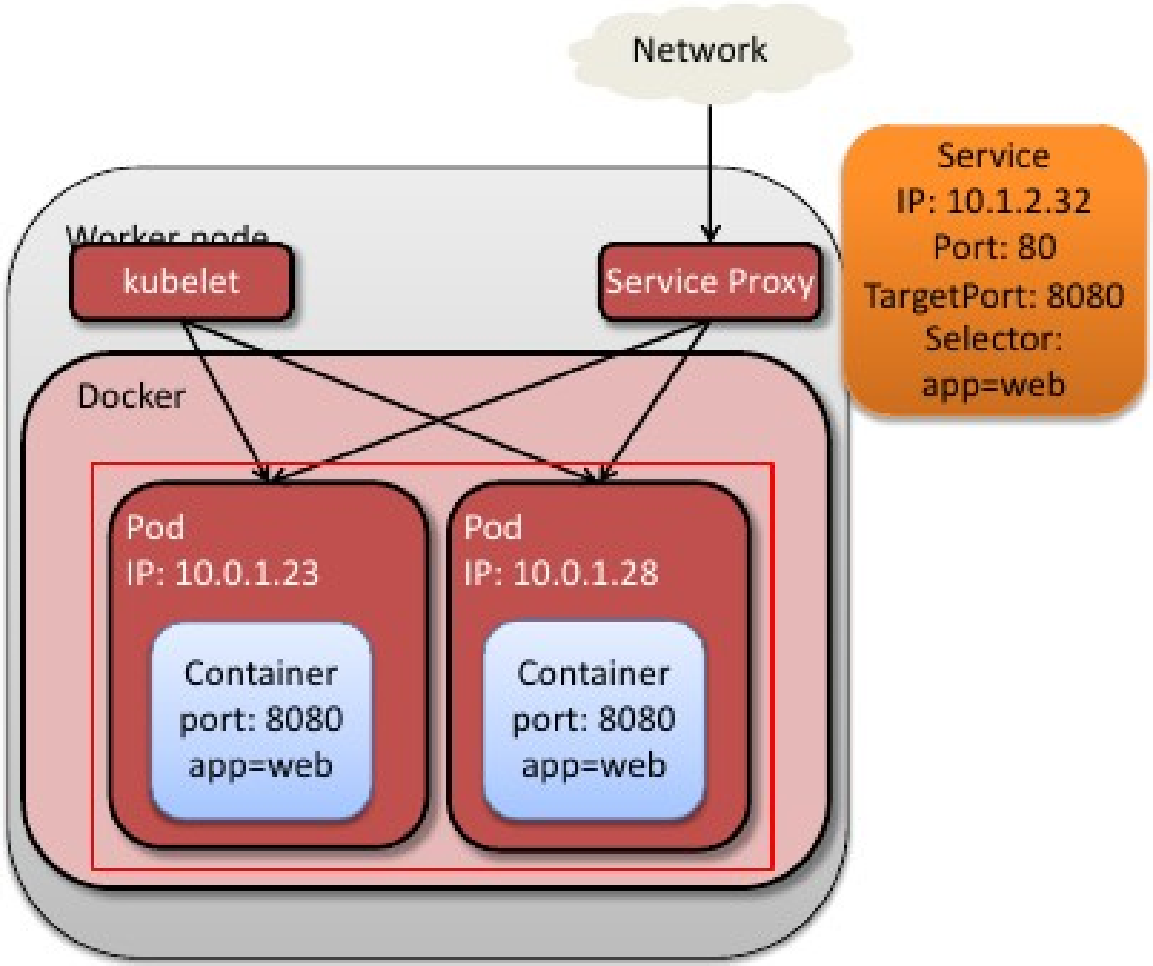
- Services provide abstraction between layers of an application
  - Provides a stable IP for a collection of pods
  - Uses a label selector to target a specific set of pods as an endpoint to receive proxied traffic
  - Clients can reliably connect via the service IP and ports
  - Even if individual pods are being created and destroyed dynamically
  - Can model other types of backends using services without selectors

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
```

sampleservice.yaml

Kubernetes Services

- Can be used for communications between application tiers
- Can be used to expose applications outside the cluster
- Distribute requests over the set of pods matching the service selector
  - Functions as a TCP and UDP proxy for traffic to its pods
  - Maps the defined service ports to listening ports on pods



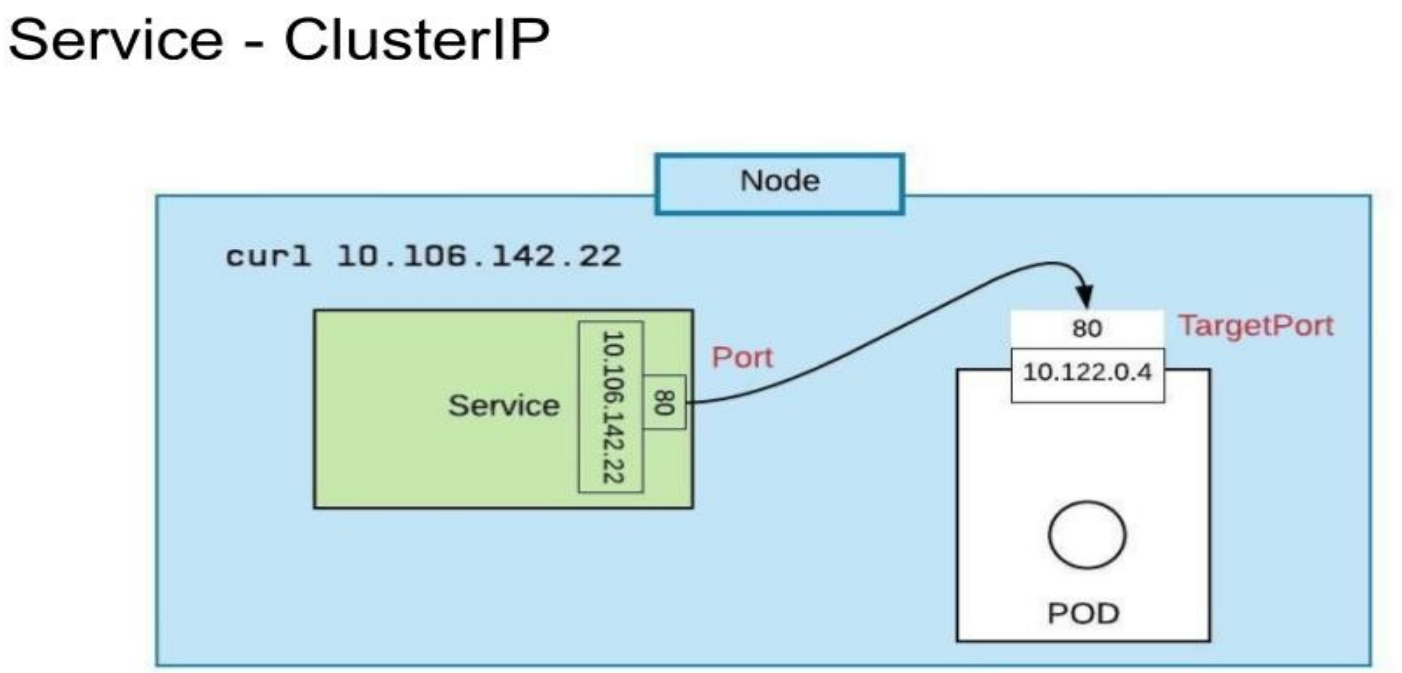
Defining a Service

- Kind field specifies “Service”
- Metadata includes
  - Name to assign to the service
  - Labels attached to the service
- Spec includes the the ports associated with the service
  - “port” is the service port
  - “targetport” is the listening port on the pods
- Selector identifies the labels on the pods used to identify endpoints for the service

```
kind: Service
apiVersion: v1
metadata:
  name: test-service
  labels:
    stage: test
spec:
   ports:
     - port: 80
       targetPort: 8080
   selector:
     tier: frontend
     stage: test
```

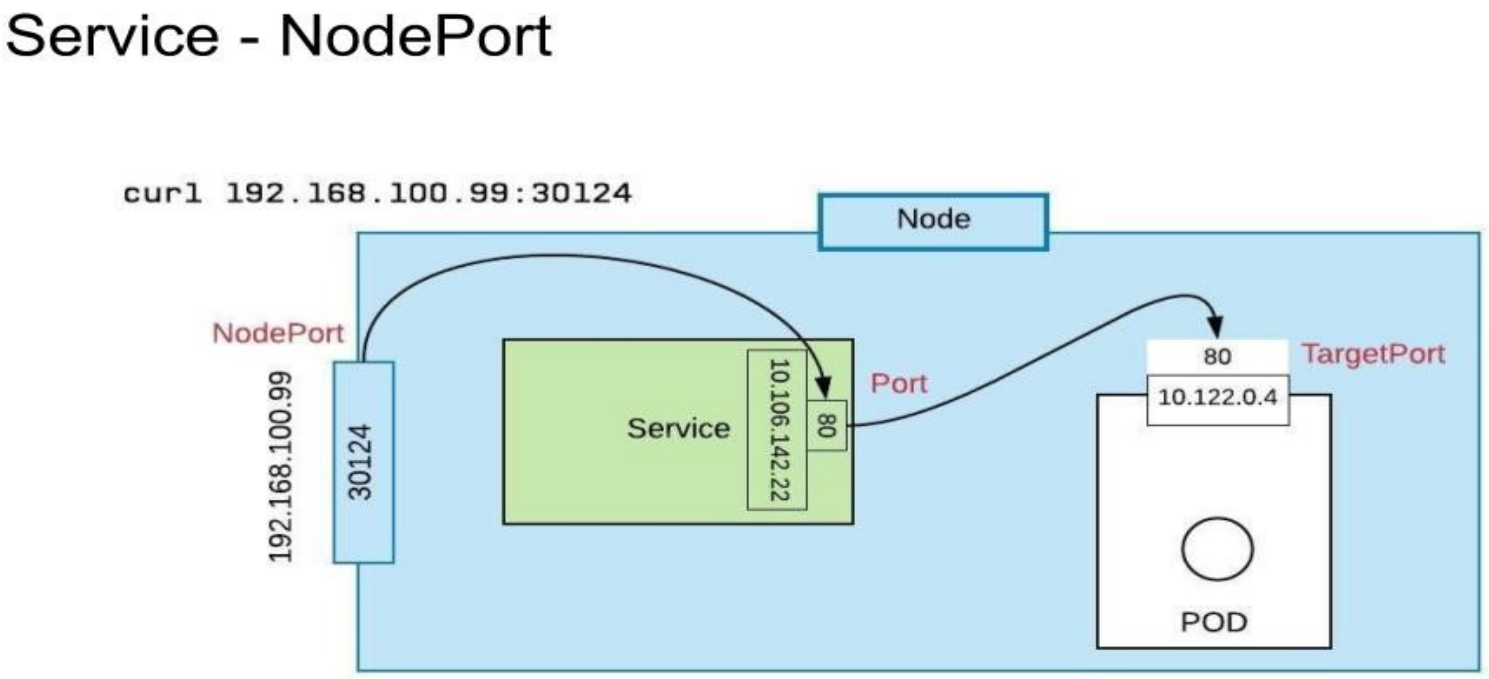
Service Types – ClusterIP

- Provides a single IP address within a cluster for a service
  - Good for layers within an application that are only accessed within a cluster
  - Default type of service



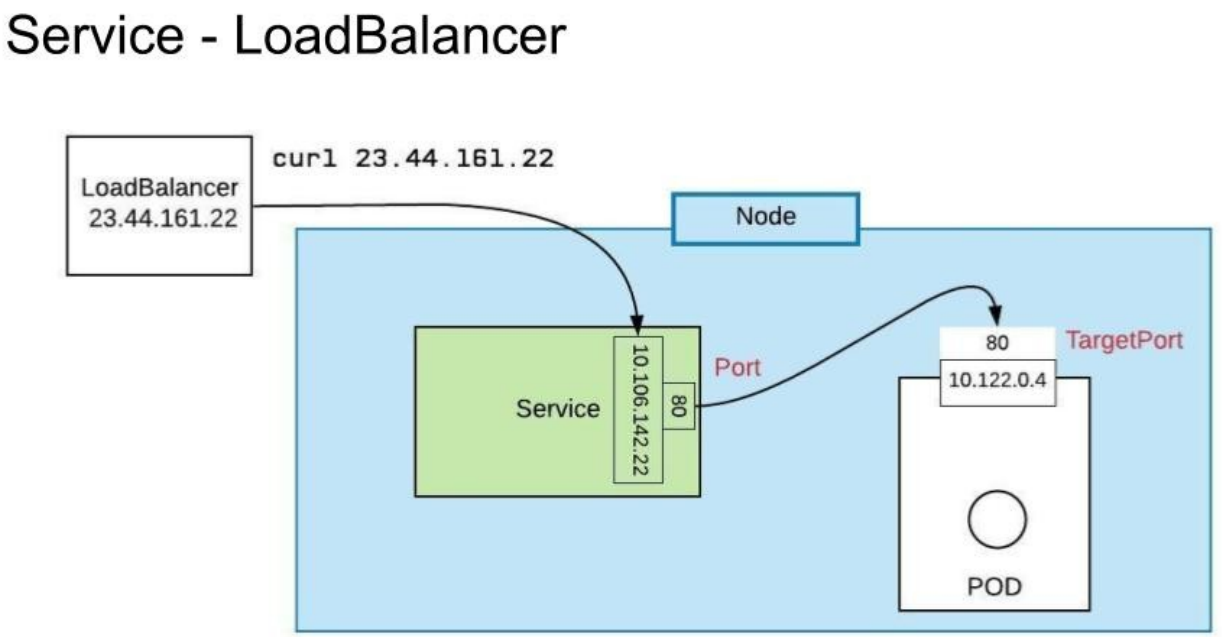
Service Types – NodePort

- Exposes a service on each node's static IP



Service Types – LoadBalancer

- Exposes the service externally using the IP exposed by the external load balancer



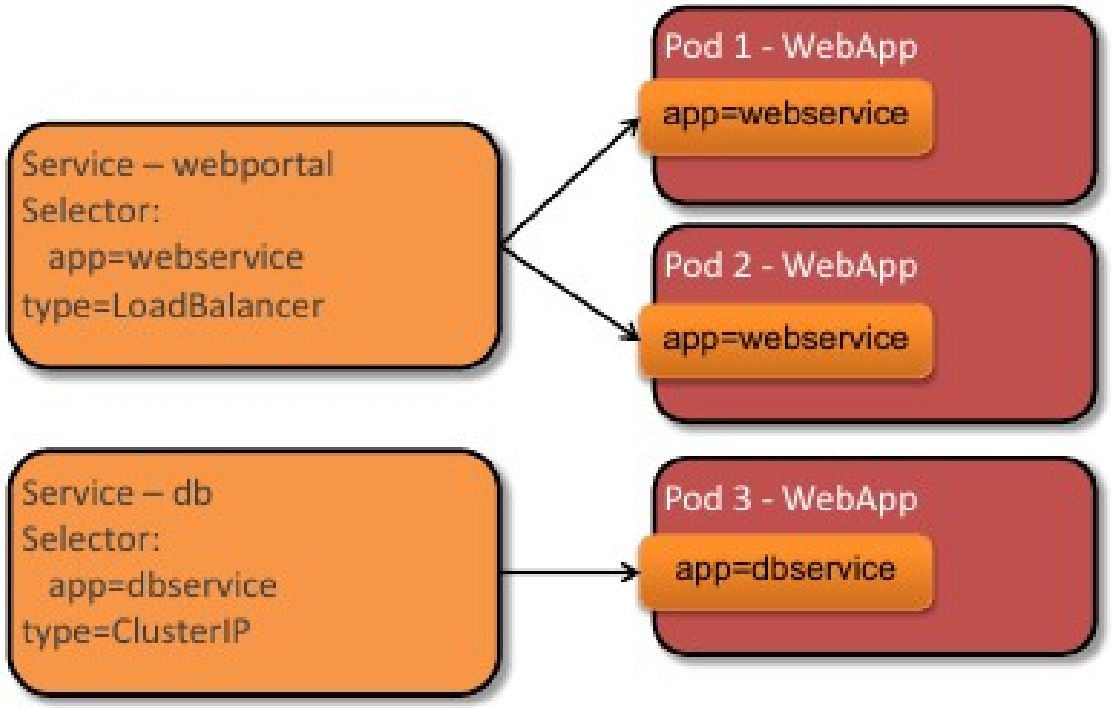
Ingress Rules wit L7 Loadbalancer

- Ingress resource is used to configure an external L7 loadbalancer
- Set of rules matching host/url paths to specific service backends
- Requires the cluster to be running a appropriately configures Ingress controller

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
spec:
  rules:
  -host: bar.foo.com
   http:
      paths:
      - path: /first
        backend:
          serviceName:
firstservice
          servicePort: 80
      - path: /second
        backend:
          serviceName:
secondservice
          servicePort: 80
```

Selecting Pods as Service Endpoints

- Multiple pods can have the same lable
- Kubernetes re-evaluates a service's selector continuously
- Maintains endpoints object of the same name as the service with a list of pod IP:port's matching the selector

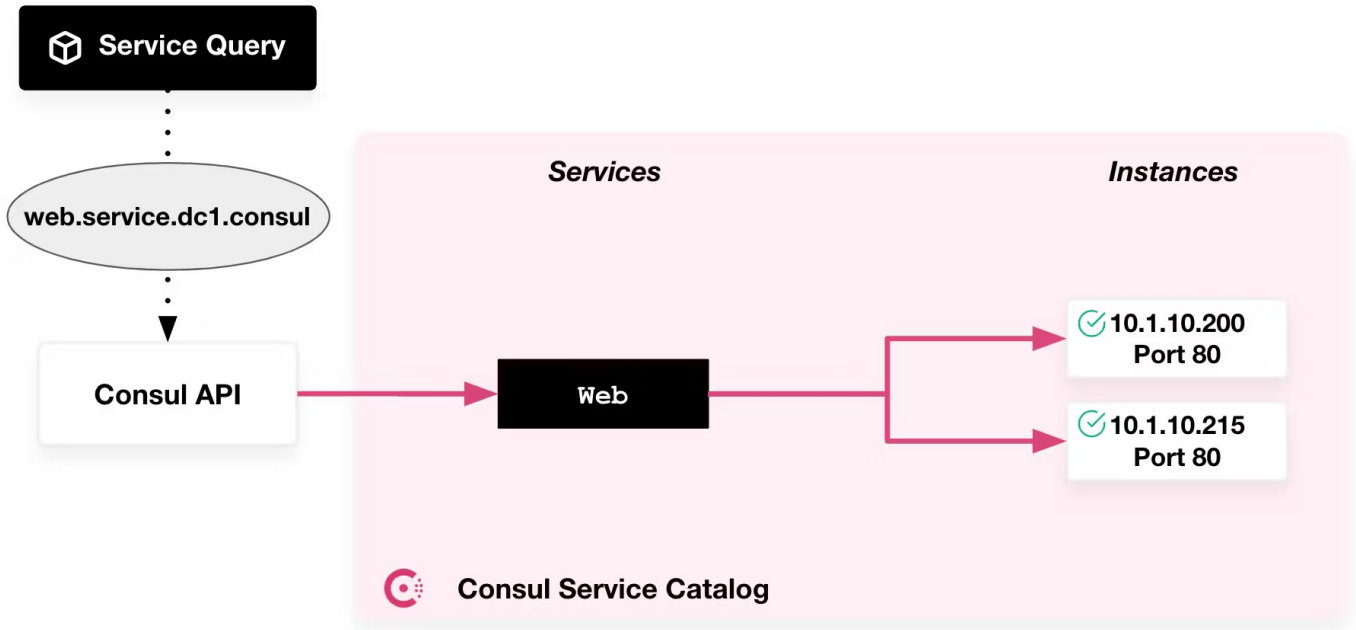


Service Mesh

- What does a Service Mesh do?
  - Enforces routing rules and restrictions
  - Binds service to implementation instance
  - Determines optimal instance
  - Keep track of performance aspects, e.g., latency
  - Coordinates retries and failures
  - Ejects failing instances from the load balancing pool
  - Provides monitoring, e.g. tracing and performance metrics
- Kubernetes does not have a build in service mesh
  - These are provided as plugins with different capabilities and features
  - There is no one service mesh that meets everyone’s needs

Explicit Services with Consul

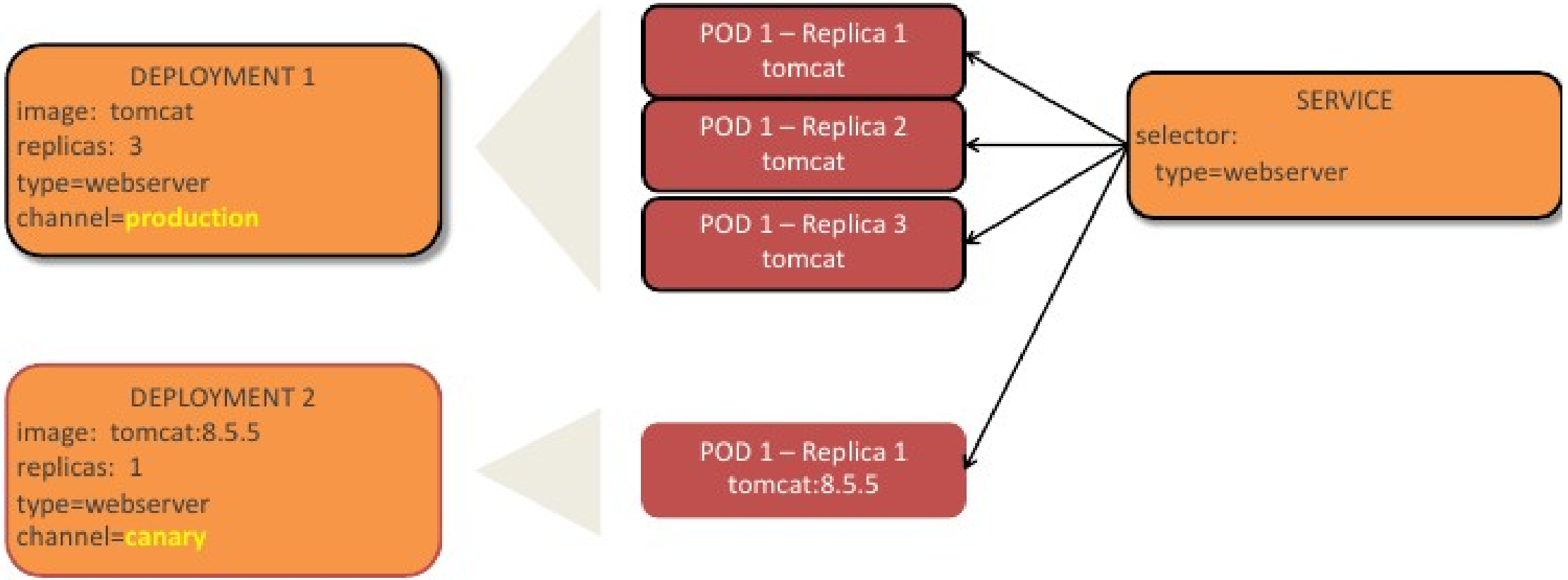
- Hashicorp Consul is a service mesh that provides a service catalogue
- Services are registered with the catalogue
  - Avoids having to work with IP and ports directly
  - The catalogue knows which pods are endpoints



Canary Deployment

- Controlled testing of an update on production
- Consider a service selecting pods from a deployment
- A second “canary” deployment is created with a new version
  - Has the same selector labels and the existing service
- Service directs some requests to the canary endpoints
  - Allows for testing of the new version in production
- If a malfunction is detected, it will only impact a small portion of the pod and can be undone
- If the error rate is not increasing and the canary deployment is stable
  - The main deployment can be updated
  - Or the canary deployment can be scaled up to replace the old deployment

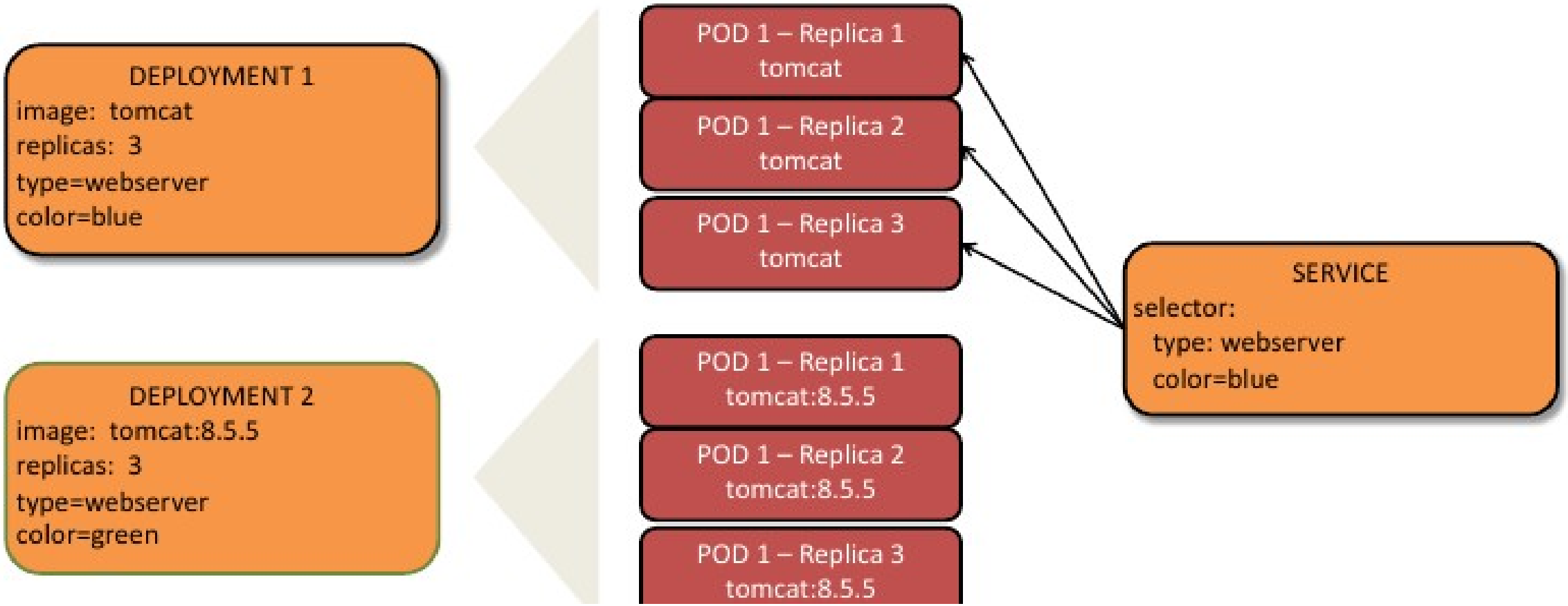
Canary Deployment



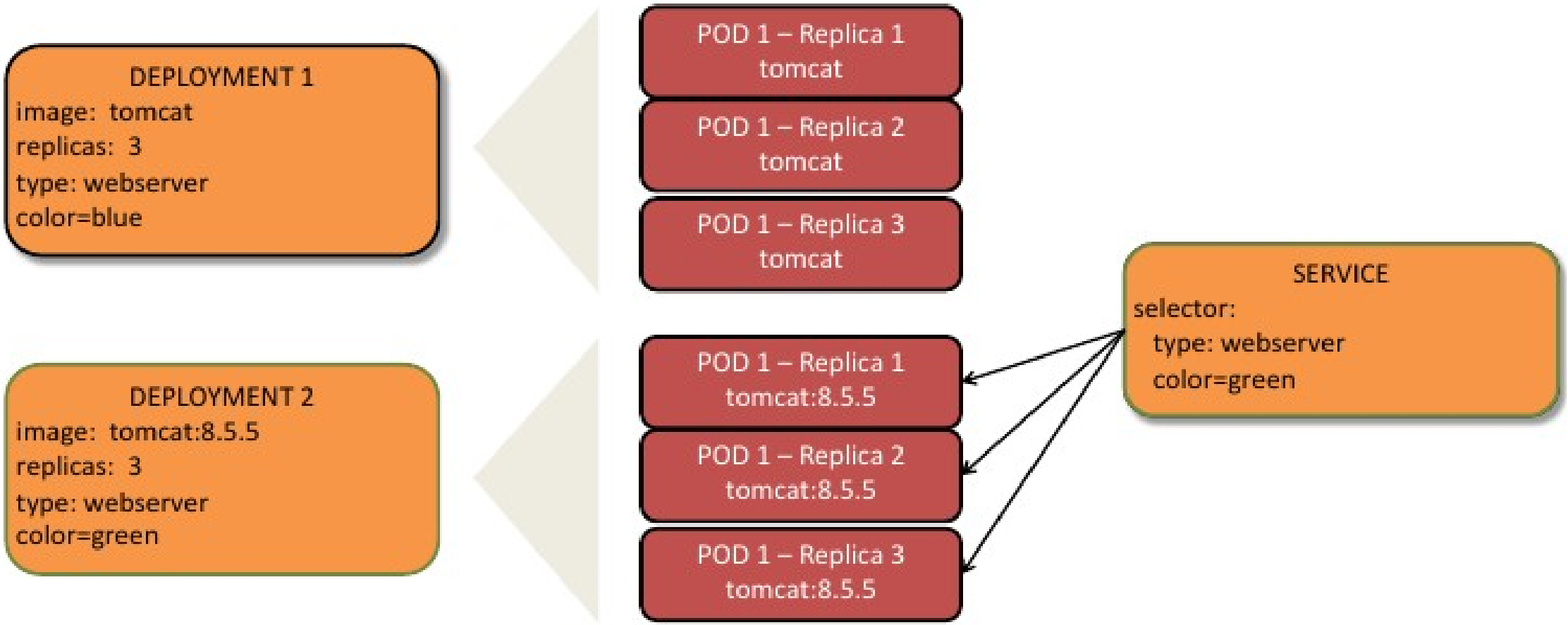
Blue/Green Deployment

- Complete environment shift from one version to another
- A new full scale deployment is created in addition to the current production deployment
- Reconfiguring the pod selector label on the service allows the choice of directing service requests to either the old or new deployment
- Allows changing a deployment endpoint seamlessly with no downtime

Blue/Green Deployment



Blue/Green Deployment



End of Module