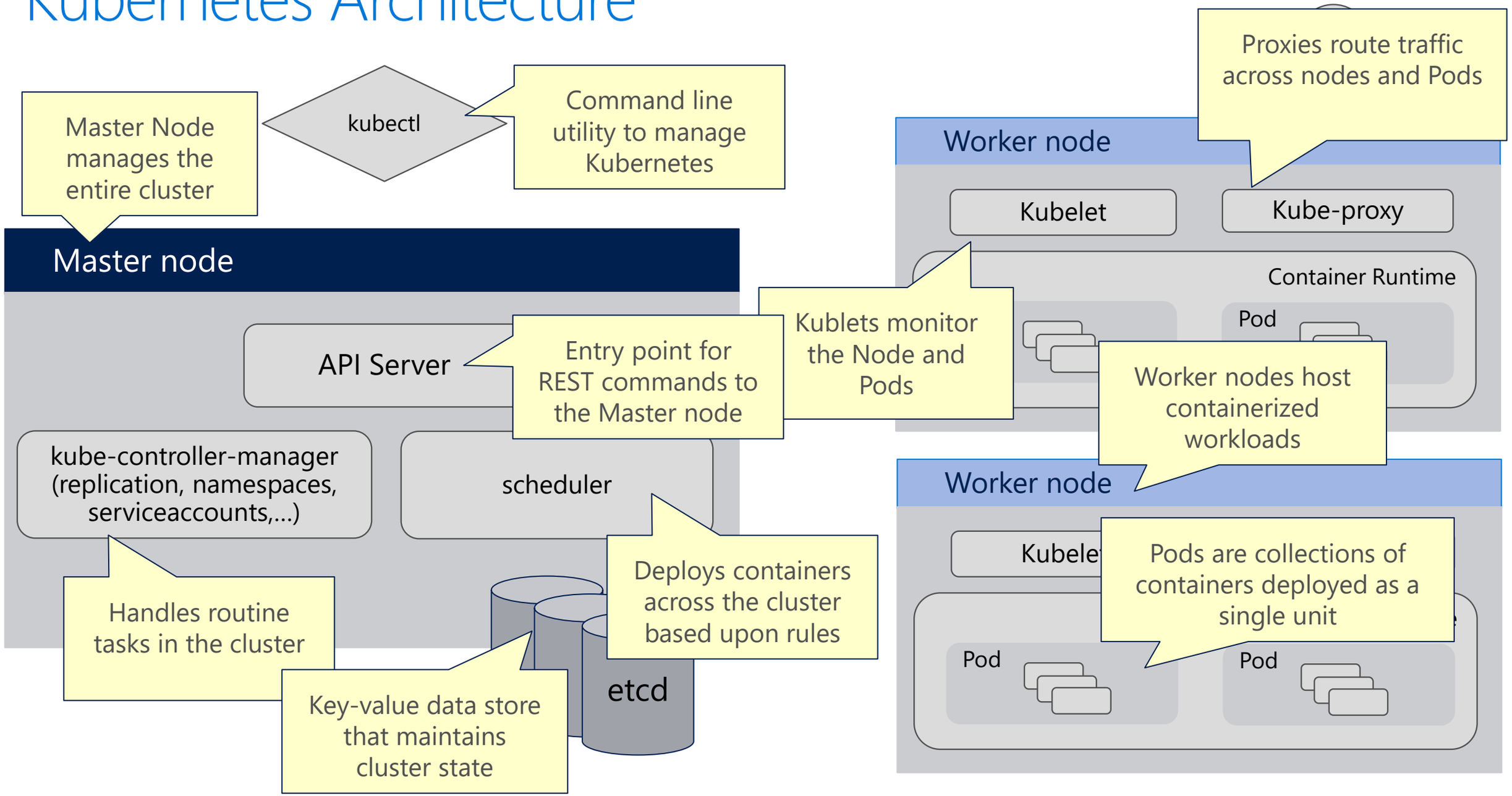
Microsoft

Kubernetes Fundamentals

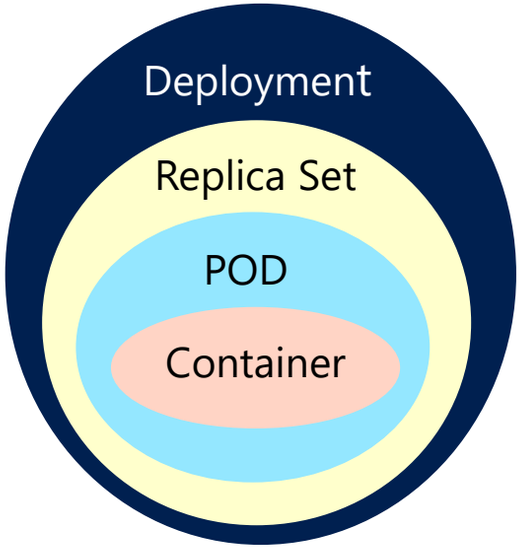
Kubernetes Resources

Microsoft Services

Kubernetes Architecture



What are Pods?



- Pods are the smallest building block in Kubernetes…
  - **A collection of co-located containers and volumes**
  - **Running in the same execution environment**
  - **Managed as a single atomic unit**
- You never directly run a container, instead you run a Pod
- Apps running in a Pod share the same IP, port and communicate using native interprocess communication channels
- Apps in different Pods are isolated from each other; they have different IP addresses, different hostnames, etc.
- Pods are immutable - if a change is made to a pod definition, a new pod is created, and the old pod is deleted

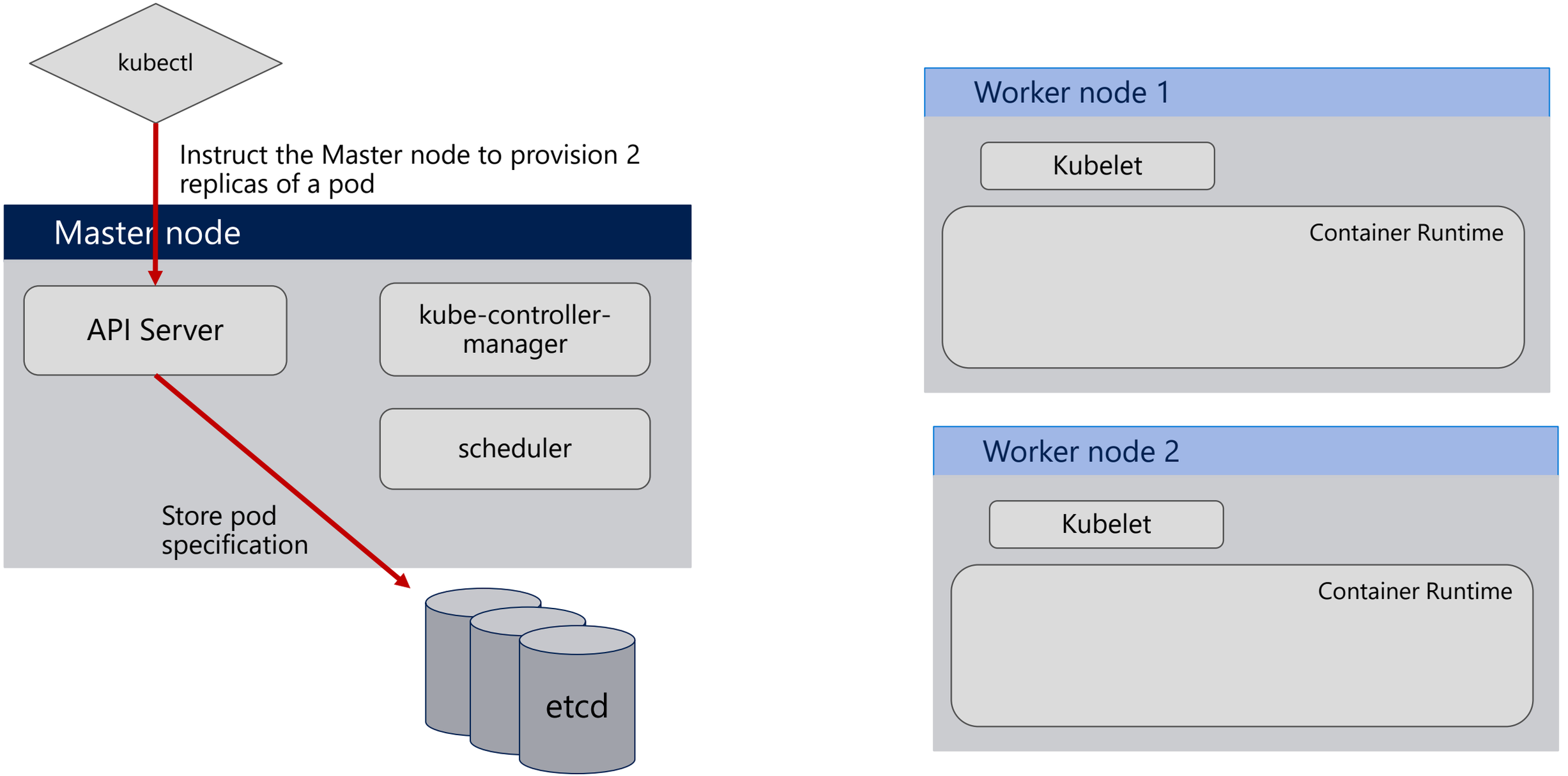
Declarative Configuration

- Pods are defined in a Pod manifest: A readable, declarative text-file
- Kubernetes itself thrives on declarative configuration…
  - Capture the desired state of a Kubernetes object in a configuration
  - Submit that configuration to a service that takes actions to ensure the desired state becomes the actual state
  - Provides for a more manageable, dynamic and reliable system
- Contrast with imperative configuration where you explicitly instruct the system what to do, typically by issuing a series of commands

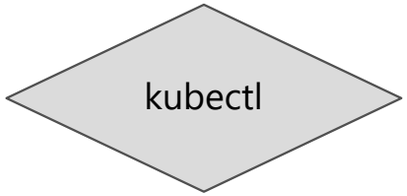
The Pod Lifecycle

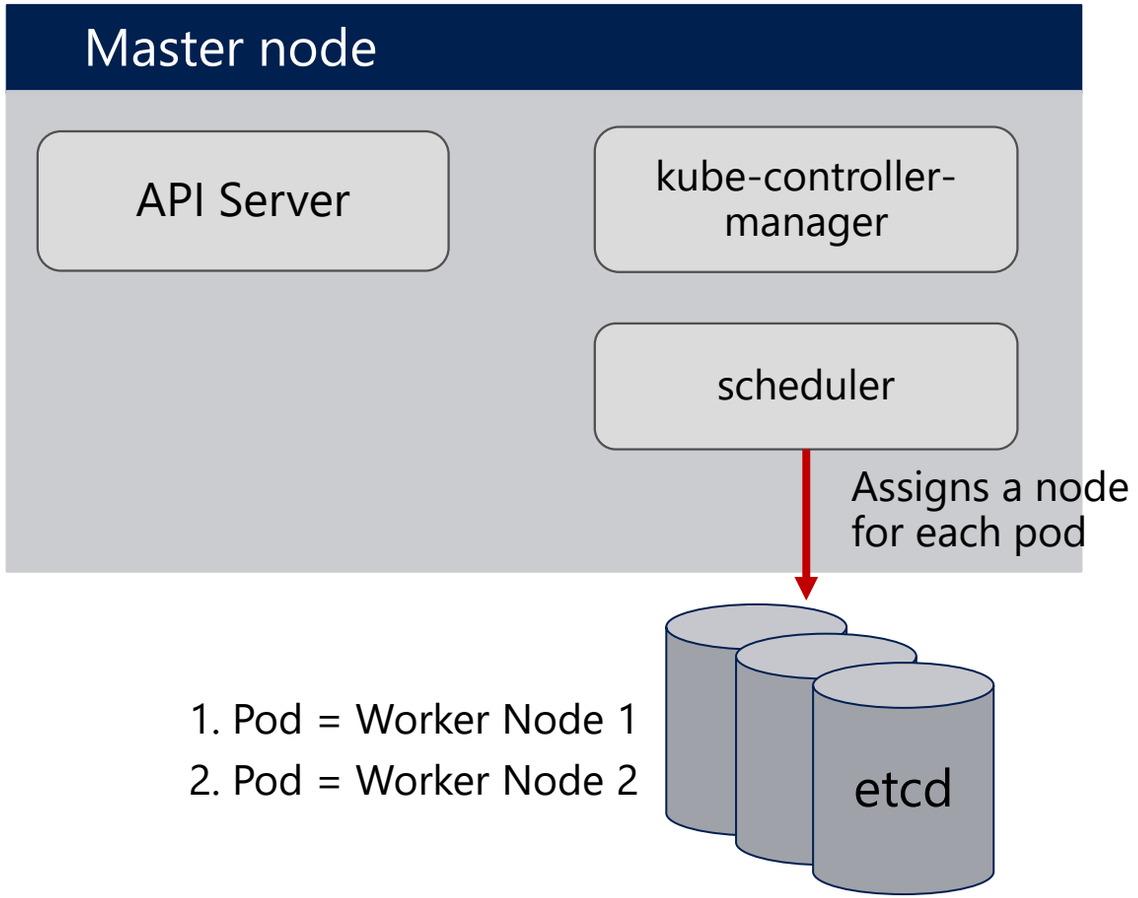
- Let's say you want to provision a container in Kubernetes
- From the Kubectl console, you…
  - Make a Pod request to an API server using a Pod definition (YAML) file
  - The API server saves the configuration data to the persistent storage (ETCD store)
  - The scheduler finds the unscheduled Pod and schedules it to an available node
  - The Kubelet sees the Pod scheduled and fires up Docker
  - Docker runs the container
- The Kubelet manages objects on the worker nodes
- The entire lifecycle state of the Pod is stored in the Etcd store

How Pods Work

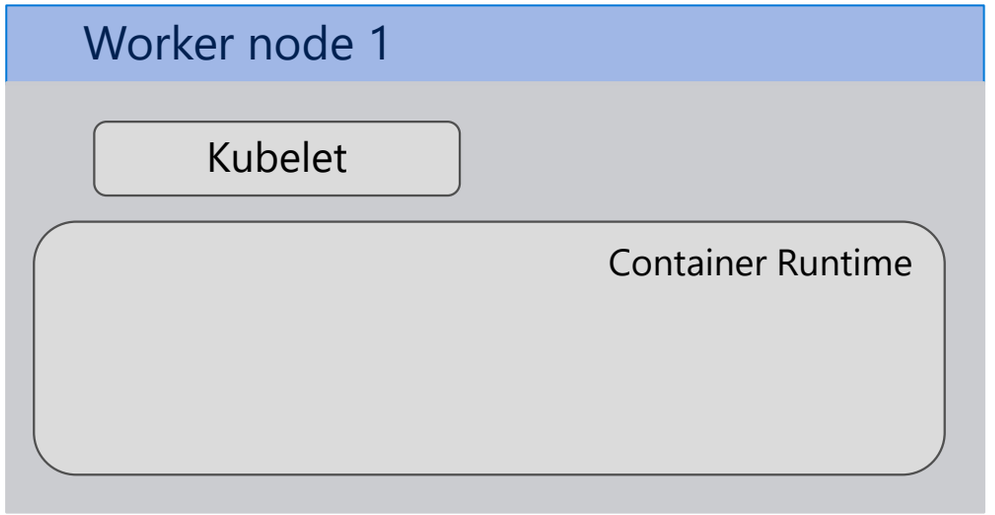


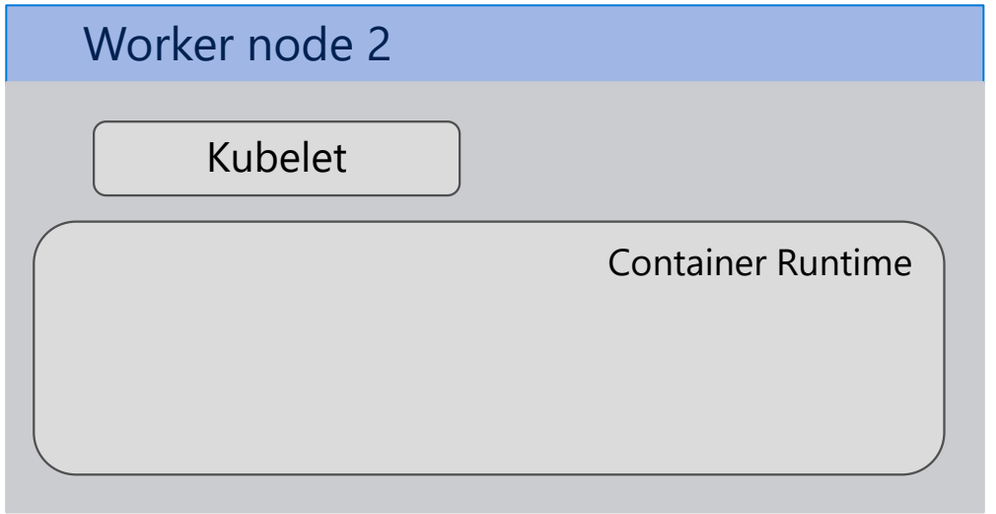
How Pods Work



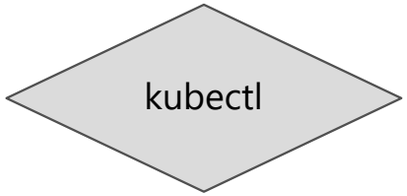


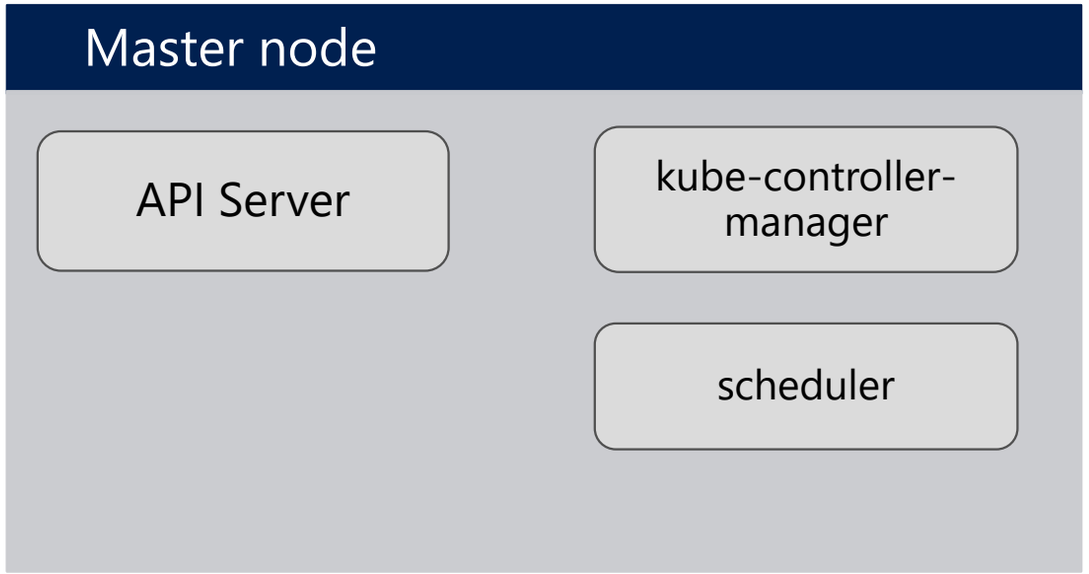
1. Pod = Worker Node 1
2. Pod = Worker Node 2



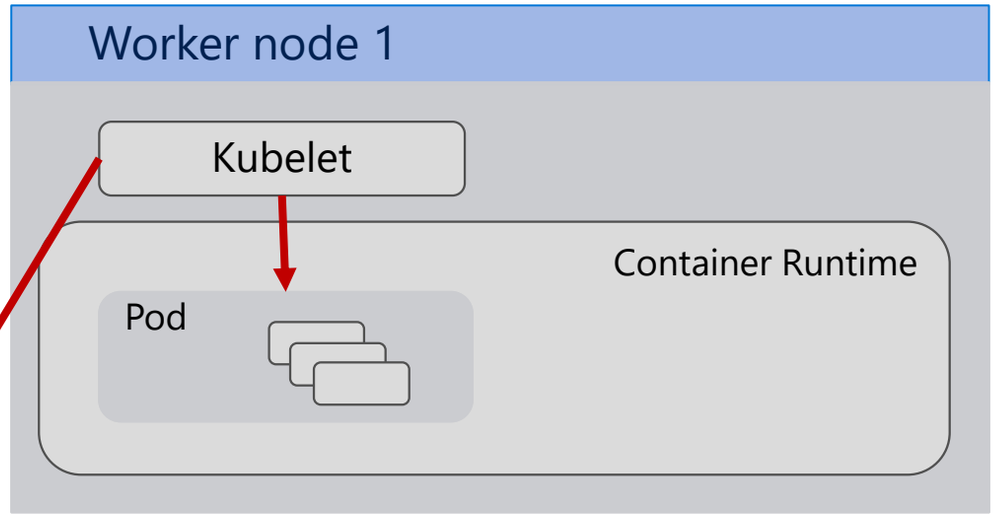


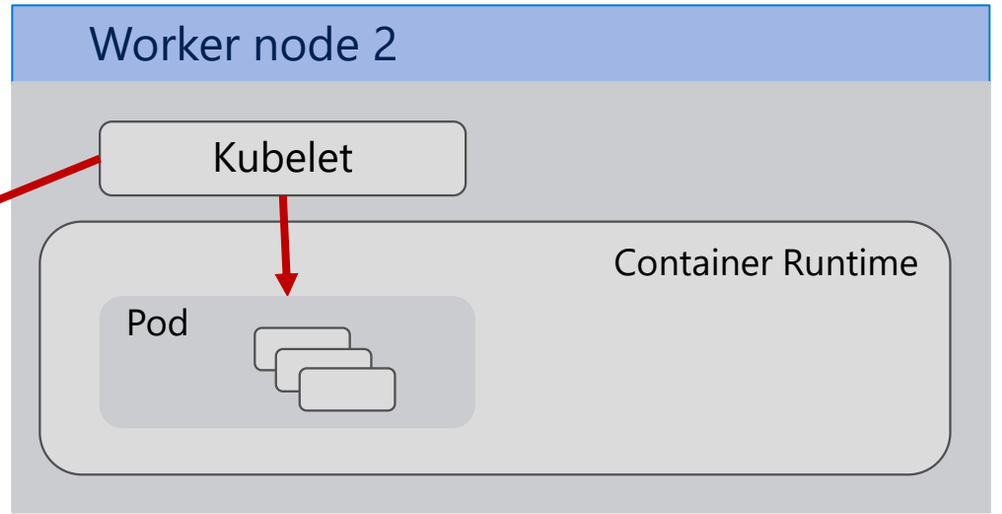
How Pods Work



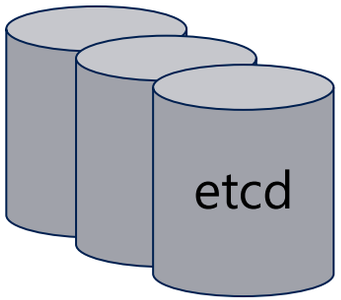


The Kubelets check if there are pods assigned and starts them

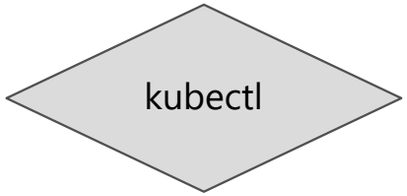




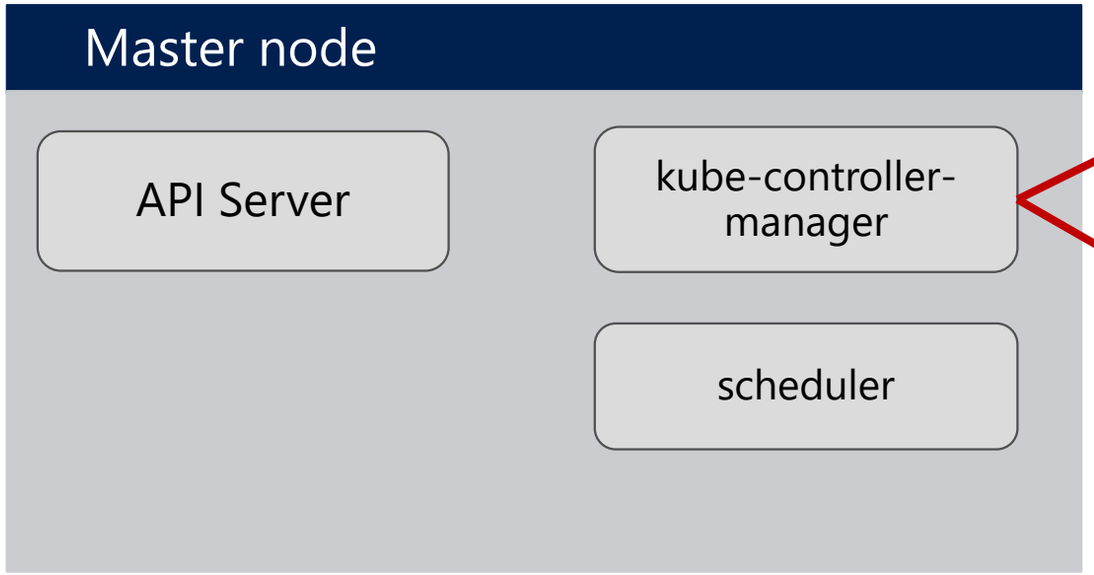
1. Pod = Worker Node 1
2. Pod = Worker Node 2

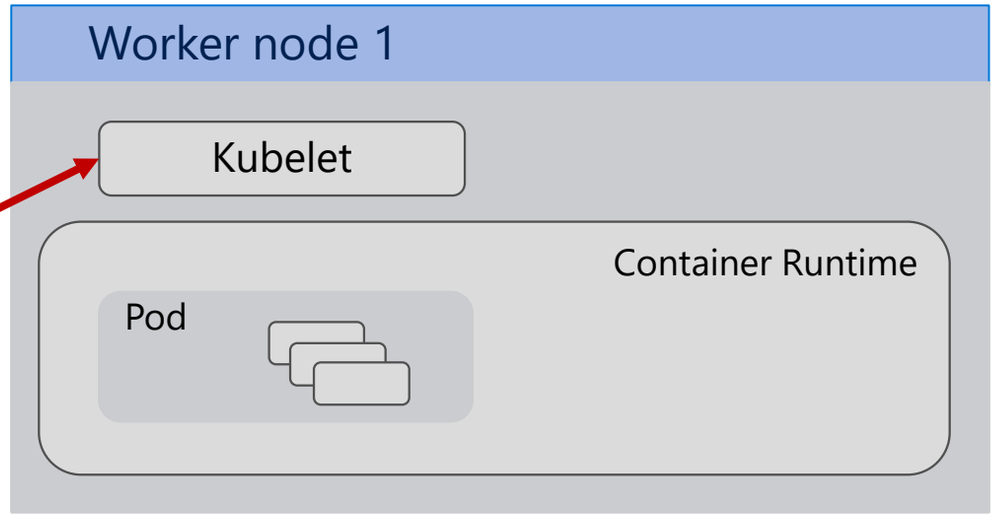


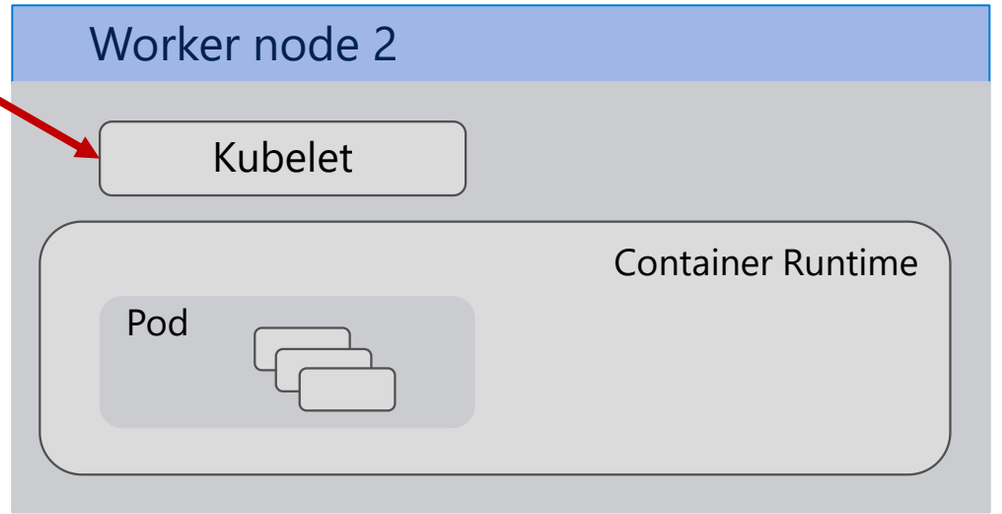
How Pods Work



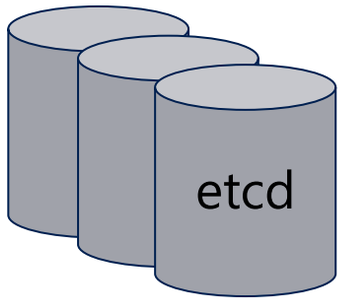
Kube-controller-manager ensures the correct number of pods is running in the cluster







1. Pod = Worker Node 1
2. Pod = Worker Node 2



Microsoft

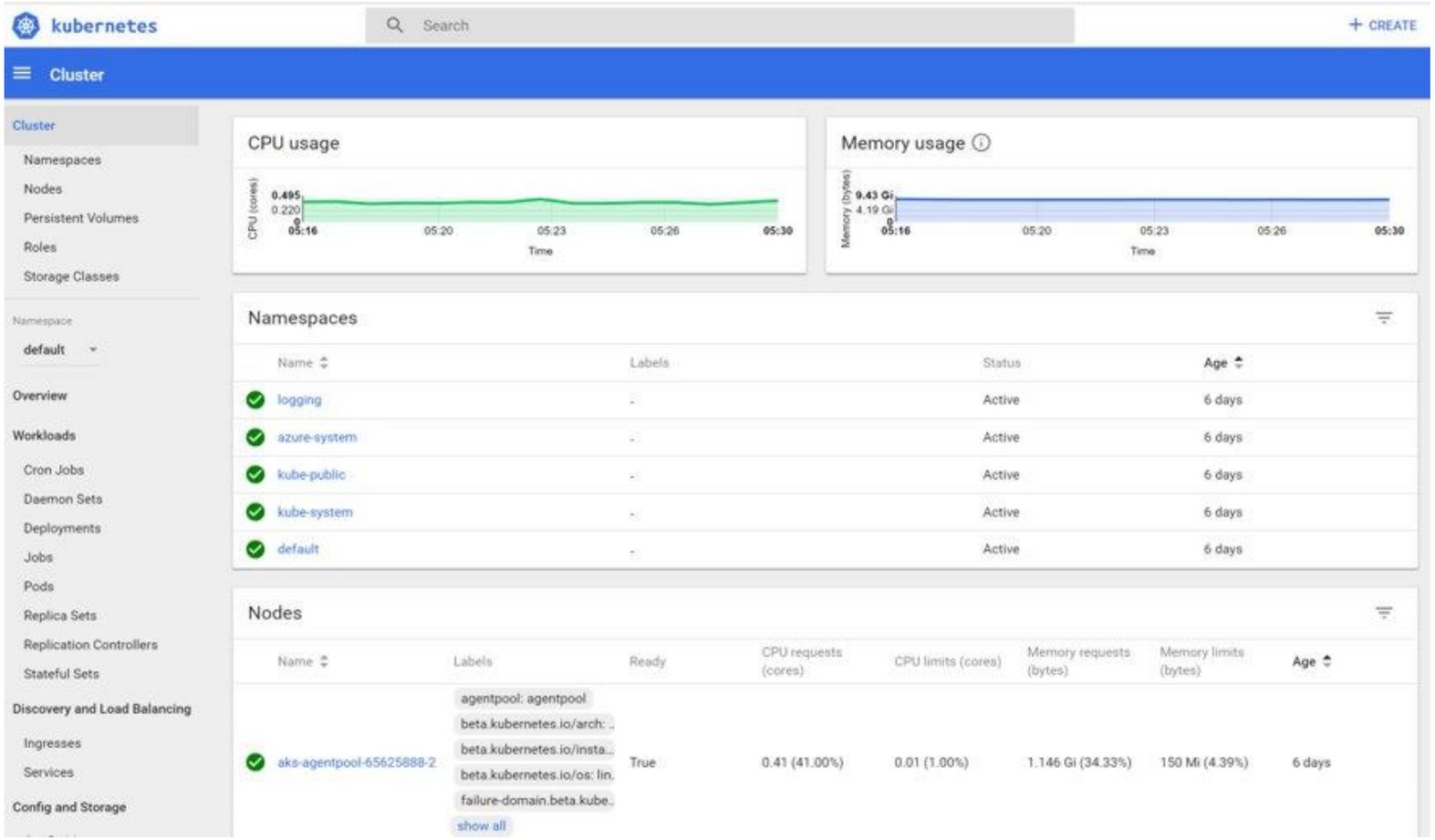
Kubernetes Fundamentals

Kubernetes Dashboard

Microsoft Services

Kubernetes Dashboard

- General purpose, web-based UI for Kubernetes clusters
- Allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself
- Very basic means of deploying and interacting with those resources



Deploy the Kubernetes Dashboard

To deploy Dashboard, execute the following command:

kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta1/aio/deploy/recommended.yaml

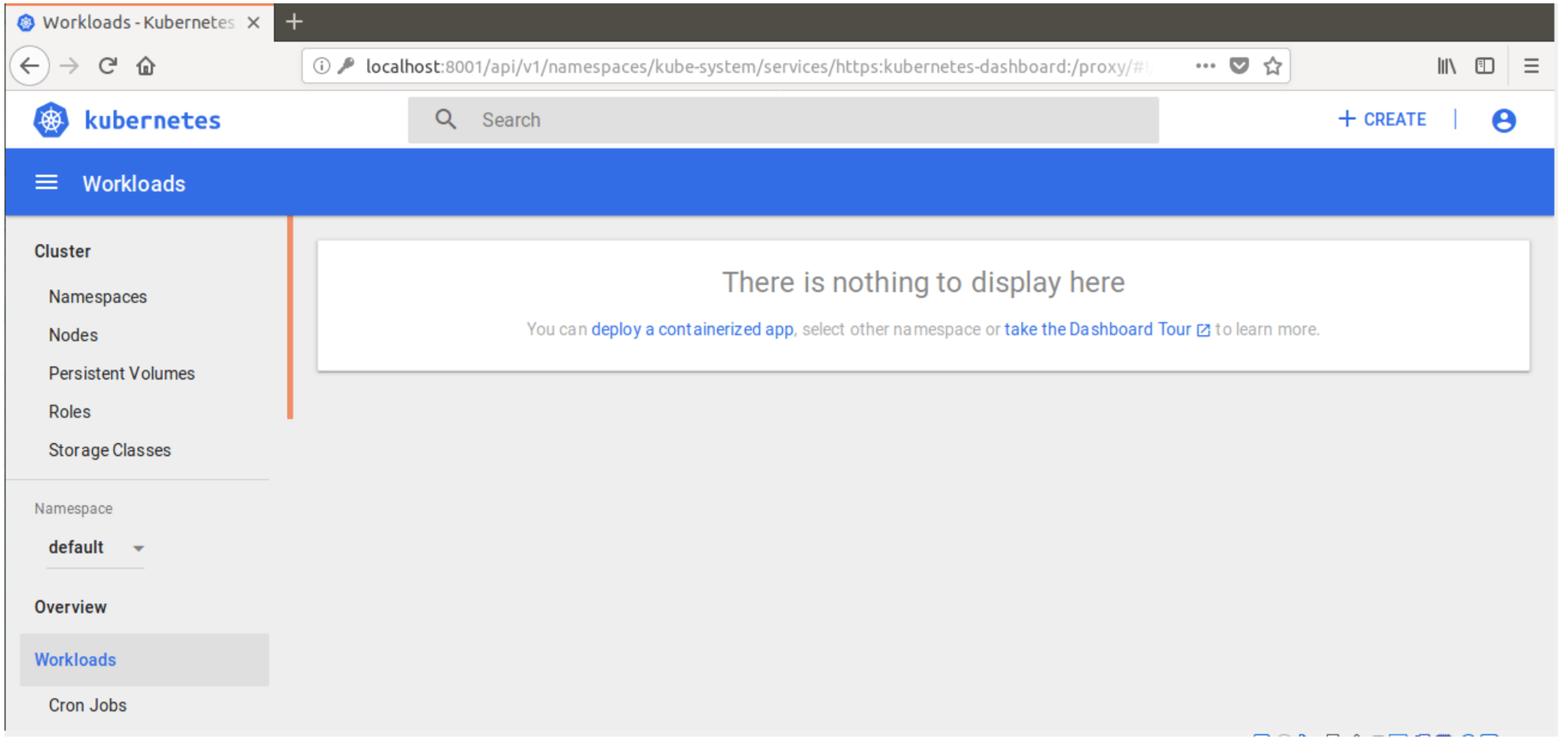
To access Dashboard from your local workstation, you must create a secure channel to your Kubernetes cluster by running the following command:

Kubectl proxy

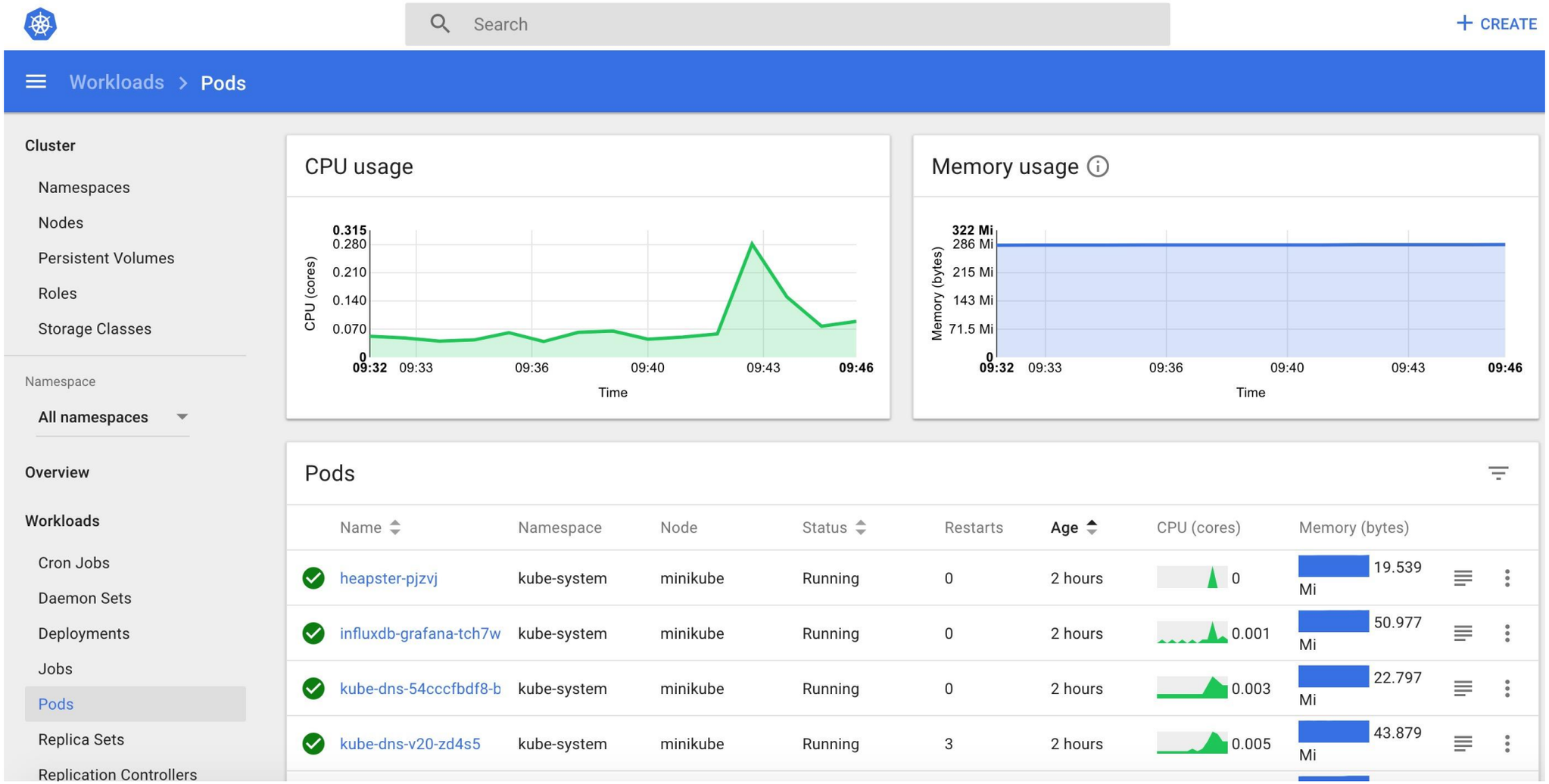
Kubectl will make Dashboard available at

http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

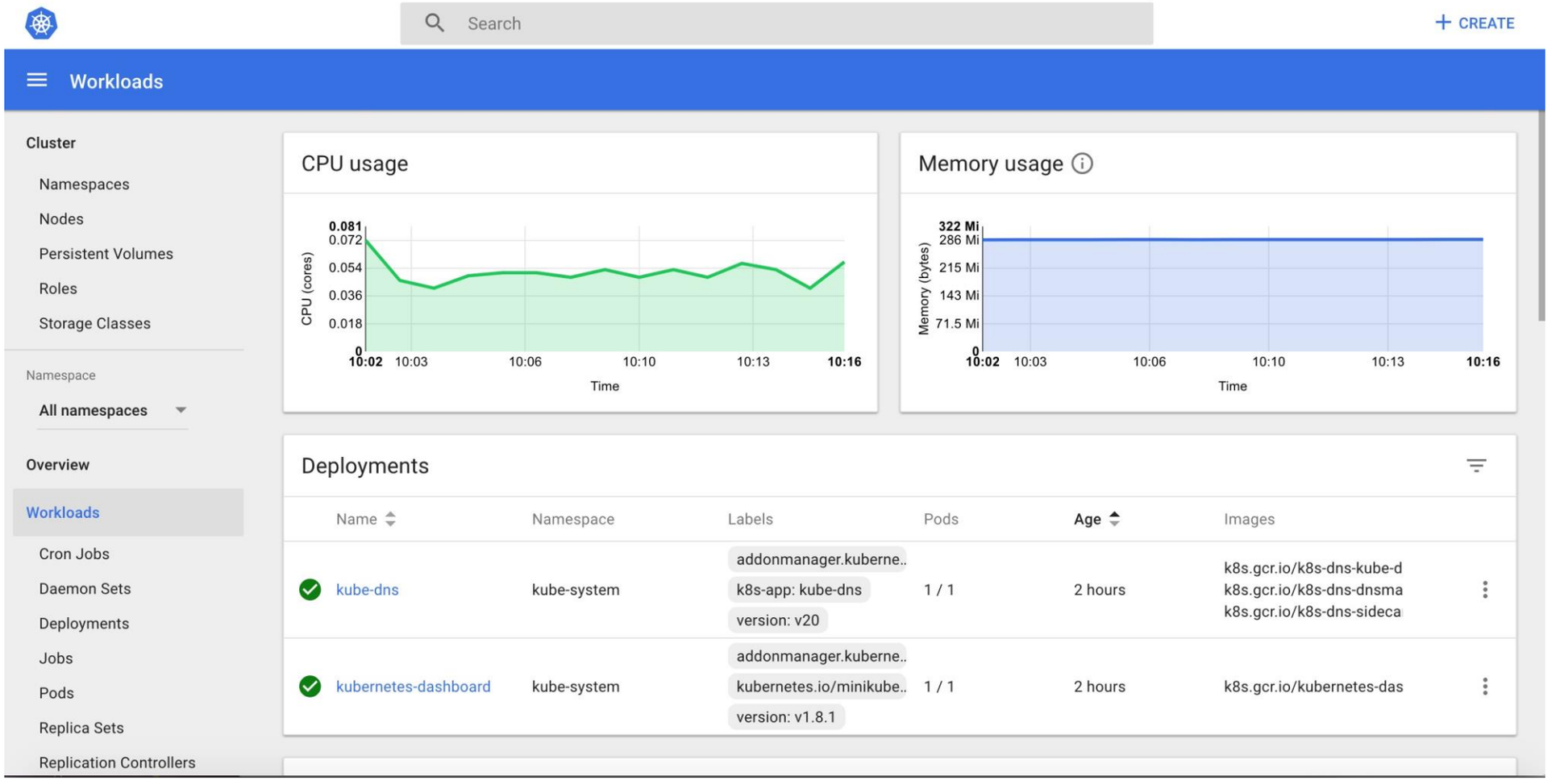
Kubernetes Dashboard



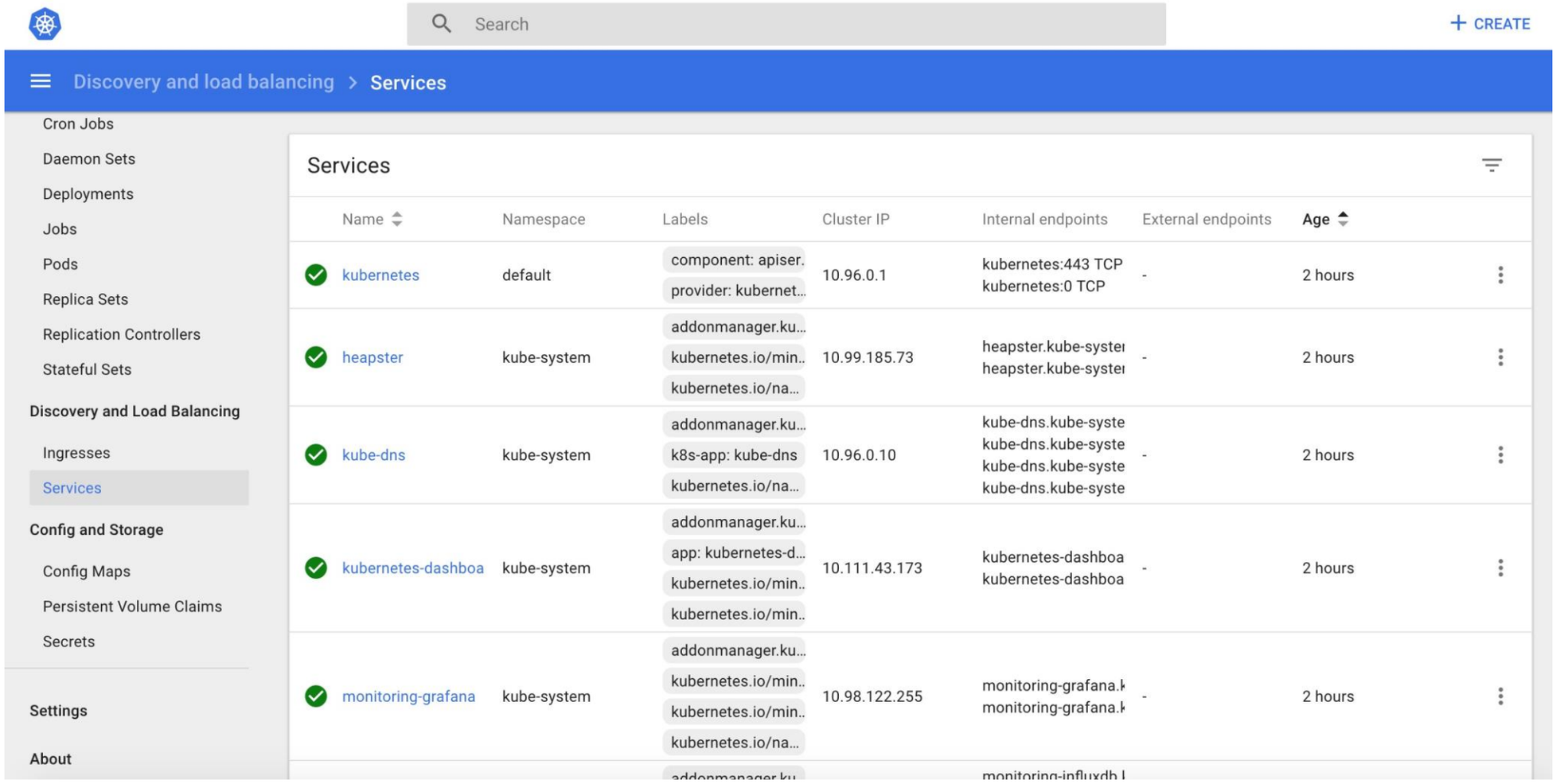
Kubernetes Dashboard



Kubernetes Dashboard



Kubernetes Dashboard



Microsoft

Kubernetes Fundamentals

The Pod Manifest

Microsoft Services

The Pod Manifest

- Pods are described in a Pod manifest. The Pod manifest is just a text-file representation of the Kubernetes API object.
- A Kubernetes manifest file defines a desired state for the cluster, such as what container images to run
- Pod manifests can be written using YAML or JSON, but YAML is generally preferred because YAML tends to be more user-friendly. *Fun fact, kubectl converts the information to JSON when making the API request*
- The Kubernetes API server accepts and processes Pod manifests before storing them in persistent storage (etcd).

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

Required Fields in Kubernetes YAML Files

There are a few required fields in every Kubernetes YAML file:

- apiVersion — Which version of the Kubernetes API used to create the file
- Kind — What kind of object to be created
- Metadata — Data that helps uniquely identify the object, including a name string, UID and optional namespace
- Spec: — You'll also need to provide the object spec field. The precise format of the object spec is different for every Kubernetes object, and contains nested fields specific to that object. The Kubernetes API reference can help you find the spec format for all objects you can create using Kubernetes

The Kubernetes Client - Kubectl

The official Kubernetes client is kubectl: a command-line tool for interacting with the Kubernetes API. kubectl can be used to manage most Kubernetes objects, such as Pods, ReplicaSets, and Services. kubectl can also be used to explore and verify the overall health of the cluster.

- $ kubectl version

```
coweiner@Azure:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.2",
amd64"}
Server Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.5",
amd64"}
```

Kubectl

- $ kubectl get nodes

```
coweiner@Azure:~$ kubectl get nodes
NAME                       STATUS   ROLES   AGE     VERSION
aks-nodepool1-14452431-1   Ready    agent   25h     v1.14.5
aks-nodepool1-14452431-2   Ready    agent   2m20s   v1.14.5
coweiner@Azure:~$
```

You can see this is a four-node cluster that's been up for 25h and one up for 2m20s. These are the worker nodes where your containers will run.

Demonstration: *Pods*

Create a Pod

Create a Pod

- Copy the pod manifest from:
  https://raw.githubusercontent.com/kubernetes/examples/master/staging/podsecuritypolicy/rbac/pod.yaml
- kubectl apply -f pod.yaml
- kubectl get pods
- kubectl describe pod nginx

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

Create a Pod

- Create the pod manifest

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

Create a Pod

- Kubectl delete pod nginx
- Kubectl get pods

```
colin@coweiner-0402:~/mod4$
colin@coweiner-0402:~/mod4$ kubectl apply -f pod.yaml
pod/nginx created
colin@coweiner-0402:~/mod4$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          17s
colin@coweiner-0402:~/mod4$
```

Create a Pod

- Kubectl describe pod nginx

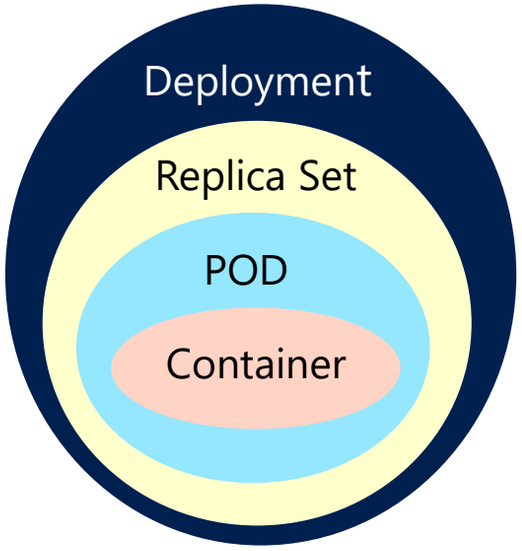
```
colin@coweiner-0402:~/mod4$ kubectl describe pod nginx
Name:         nginx
Namespace:    default
Priority:     0
Node:         docker-desktop/192.168.65.3
Start Time:   Sun, 08 Sep 2019 20:29:08 -0700
Labels:       name=nginx
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":{},"labels":{"name":"nginx"},"name
Status:       Running
IP:           10.1.0.209
Containers:
  nginx:
    Container ID:   docker://4a5ff7b9d843373598454d381f1cb452ebb0e8748770b1adedc40870b4b90e1c
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:53ddb41e46de3d63376579acf46f9a41a8d7de33645db47a486de9769
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 08 Sep 2019 20:29:11 -0700
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-zsfn8 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-zsfn8:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-zsfn8
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age    From                     Message
  ----    ------     ----   ----                     -------
  Normal  Scheduled  2d13h  default-scheduler        Successfully assigned default/nginx to docker-desktop
  Normal  Pulling    2d13h  kubelet, docker-desktop  Pulling image "nginx"
  Normal  Pulled     2d13h  kubelet, docker-desktop  Successfully pulled image "nginx"
  Normal  Created    2d13h  kubelet, docker-desktop  Created container nginx
  Normal  Started    2d13h  kubelet, docker-desktop  Started container nginx
colin@coweiner-0402:~/mod4$
```

Delete the Pod

- Kubectl get pods
- Kubectl delete pod nginx
- Kubectl get pods

```
colin@coweiner-0402:~/mod4$ kubectl get pods
NAME    READY   STATUS    RESTARTS   AGE
nginx   1/1     Running   0          8s
colin@coweiner-0402:~/mod4$ kubectl delete pod nginx
pod "nginx" deleted
colin@coweiner-0402:~/mod4$ kubectl get pods
No resources found.
colin@coweiner-0402:~/mod4$
```

Labels & Selectors

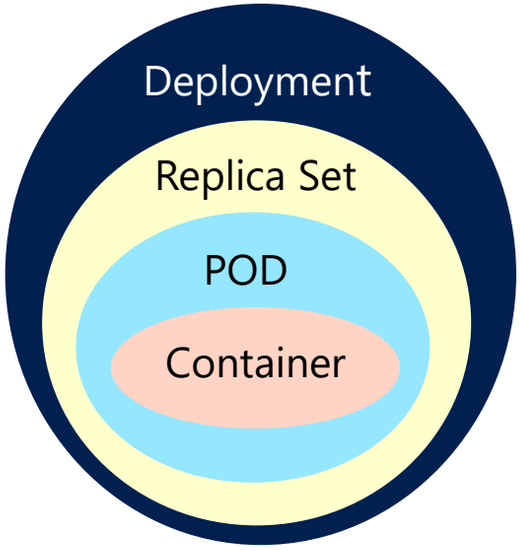


- A declarative way for which to identify Kubernetes objects
- Labels…
  - Simple key-value pair
  - Mechanism to attach arbitrary but meaningful metadata to an object
  - Ways for things in Kubernetes to find other things in Kubernetes
  - A Kubernetes object can have zero to many labels
- Selectors…
  - Mechanism to filter for labels that match certain criteria or logic

```
“labels” : {
   “tier” : “staging”,
   “type” : “redis”
}
```

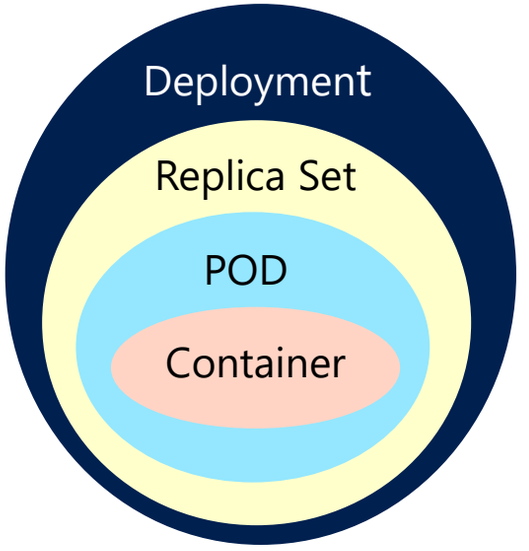
```
tier = staging
type != nginx
```

What are Replica Sets?



- A Pod is essentially a one-off singleton instance
- Replica Sets are a Kubernetes object that manage Pods
  - Redundancy – allow for failure
  - Scale – allow for more requests to be processed
- They monitor the cluster and ensure the desired number of Pods are correctly running
  - If no Pods are provisioned, the Replica Set Controller will schedule them
  - If actual count drops below the desired, the controller will schedule replacements
  - If you exceed the desired count, the controller will destroy them
- Replica sets are created by and managed through Kubernetes Deployment objects

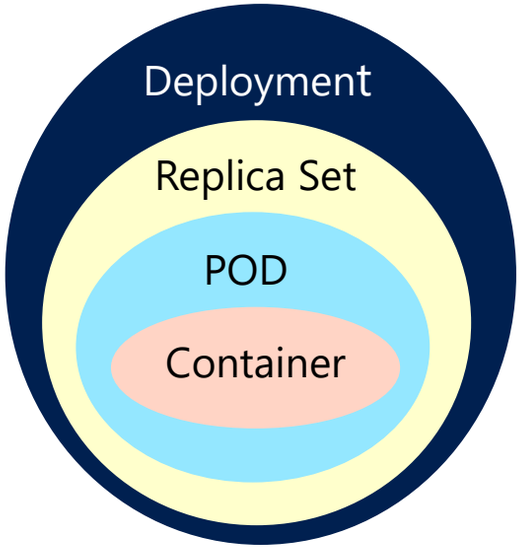
What are Deployments?



- A deployment defines the lifecycle of an application
  - Is made up of pods
  - Controls Replica Sets
  - Includes the functionality to update the desired state
  - Rolling updates are included
  - Provides fine-grained control over how and when a new pod version is rolled out as well as rolled back to a previous state

- With a deployment, you can declaratively state how many instances of your pod you would like, you can define rollout strategies, gain self-healing behavior, and much more. This provides a scalable platform to deploy your application.

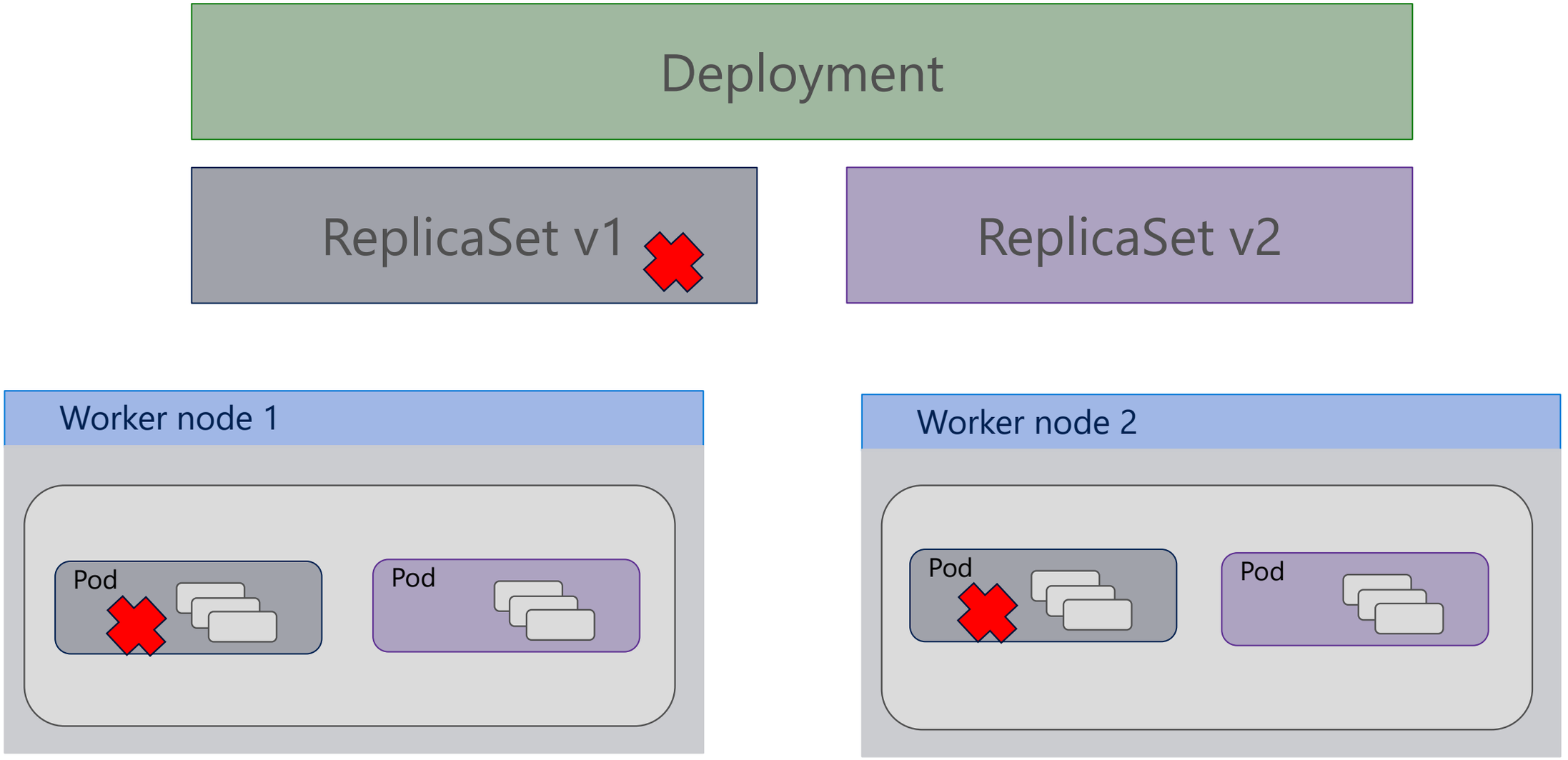
Replica Sets and Deployments



- Deployments extend the functionality of Replica Sets.

- While Replica Sets still have the capability to manage pods, and scale instances of certain pod, they don't have the ability to perform a rolling update and some other features. Instead, this functionality is managed by a Deployment, which is the resource that a user using Kubernetes today would mostly likely interact with.

How Deployment Works

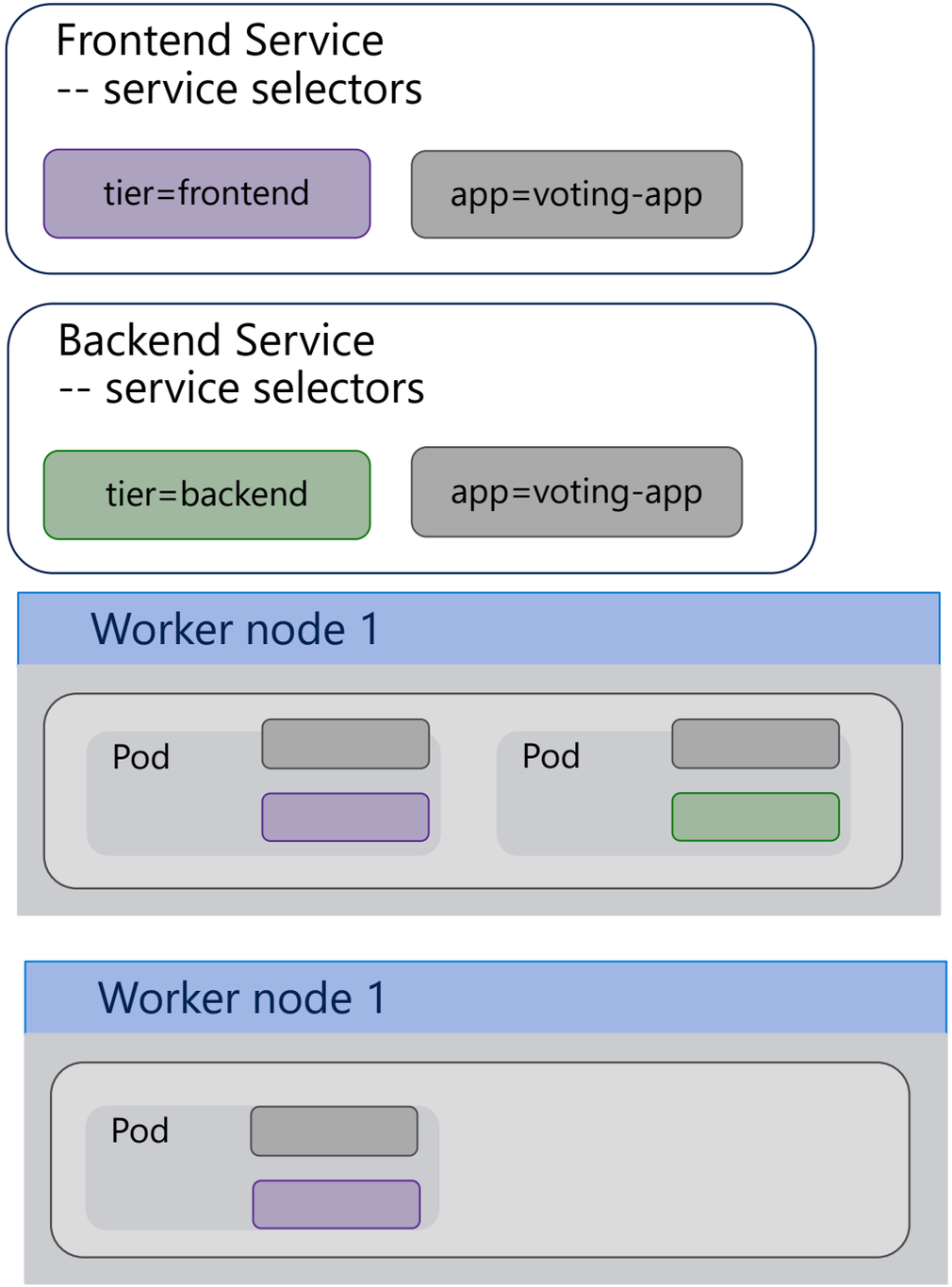


Demonstration: *Deployments and Replica Sets*

Demonstrate the use of ReplicaSets and Deployments

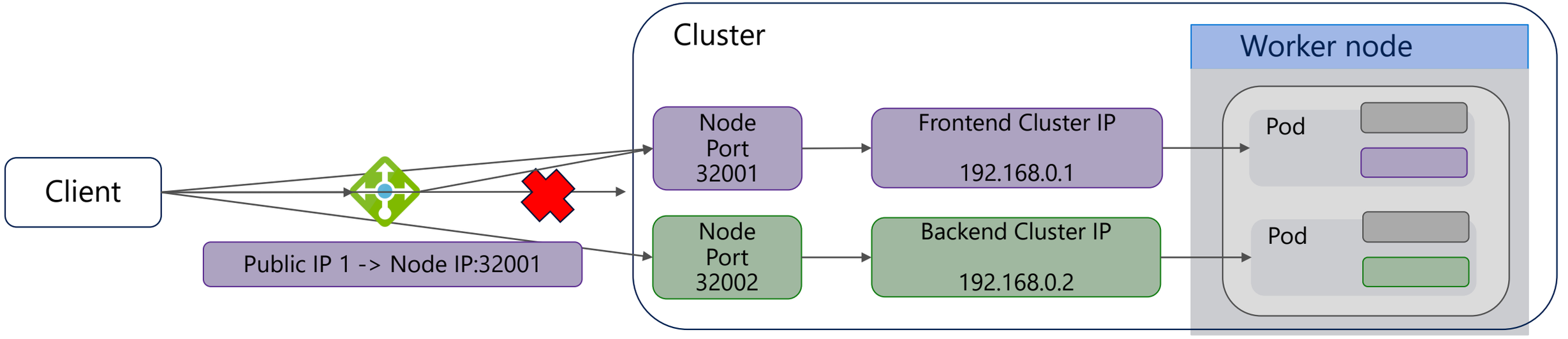
What is a Service?

- An abstraction that defines a logical set of loosely-coupled Pods and a policy by which to access them
  - Defined with a YAML markup file
  - Use “selectors” to define which pods to include
- Load balance traffic to Pods
  - Expose a frontend service to a public load balancer using selectors to bind to a specific Pod
    - labels app=voting-app
    - tier=frontend

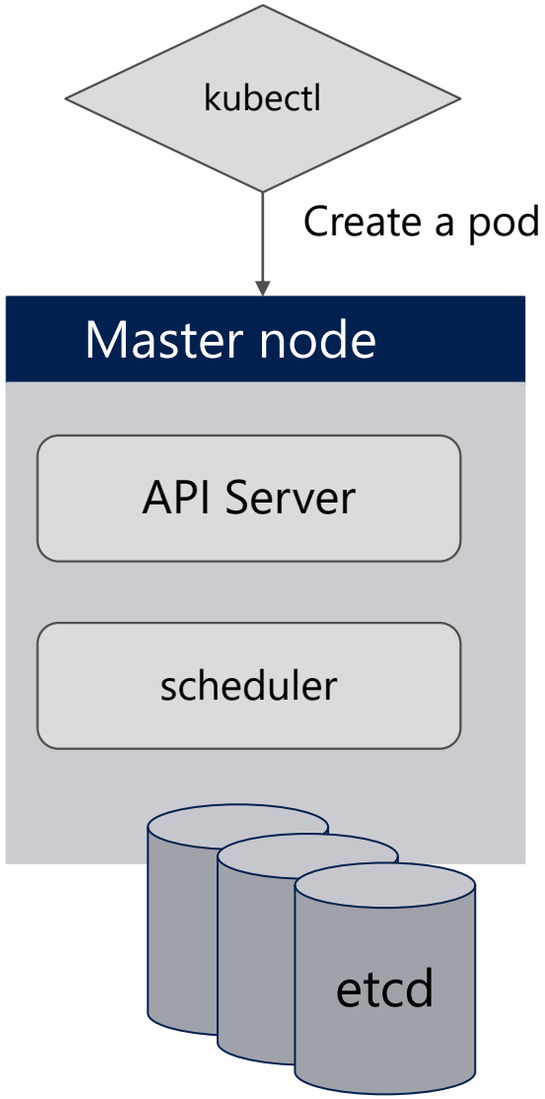


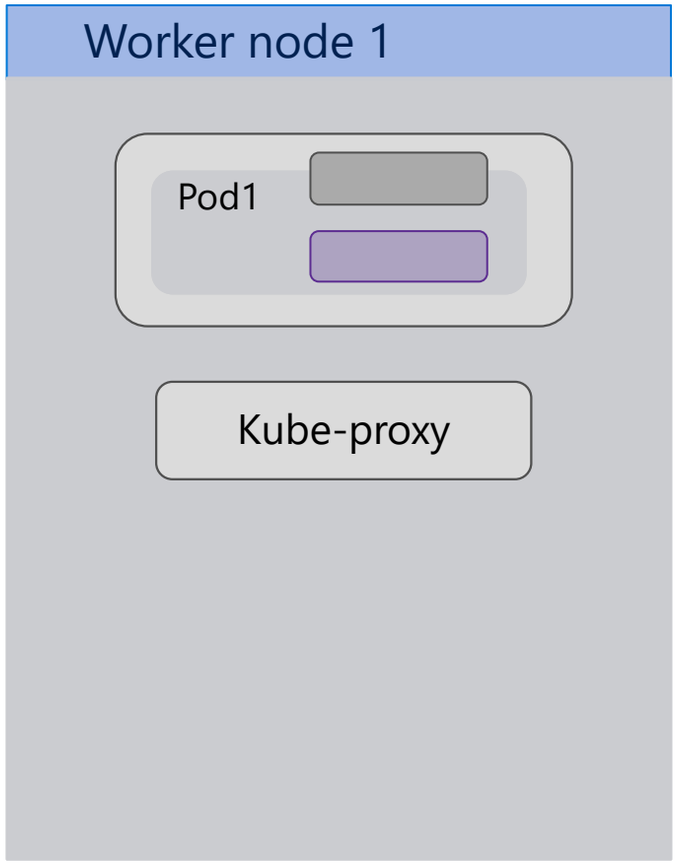
Service Object Types

- Pods are not exposed outside of the immediate cluster without a Service object – they allow Pods to receive traffic
- There are different types of services that expose your pod in different ways
  - ClusterIP: Provides a single IP internal to the cluster to represent a set of pods
  - NodePort: Reserves a specified port on the node to represent a set of pods
  - LoadBalancer: Creates a public-facing, load-balanced IP address

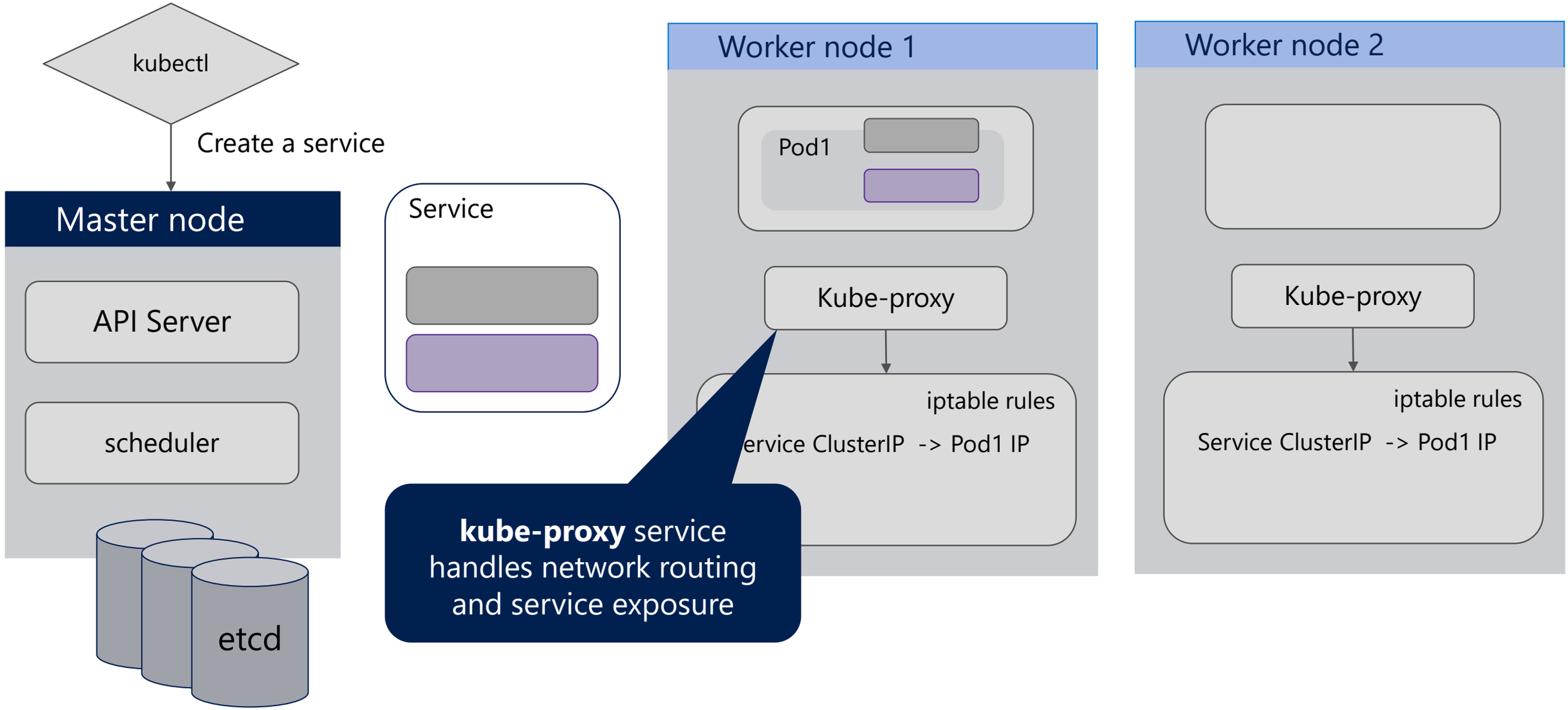


How Services Work

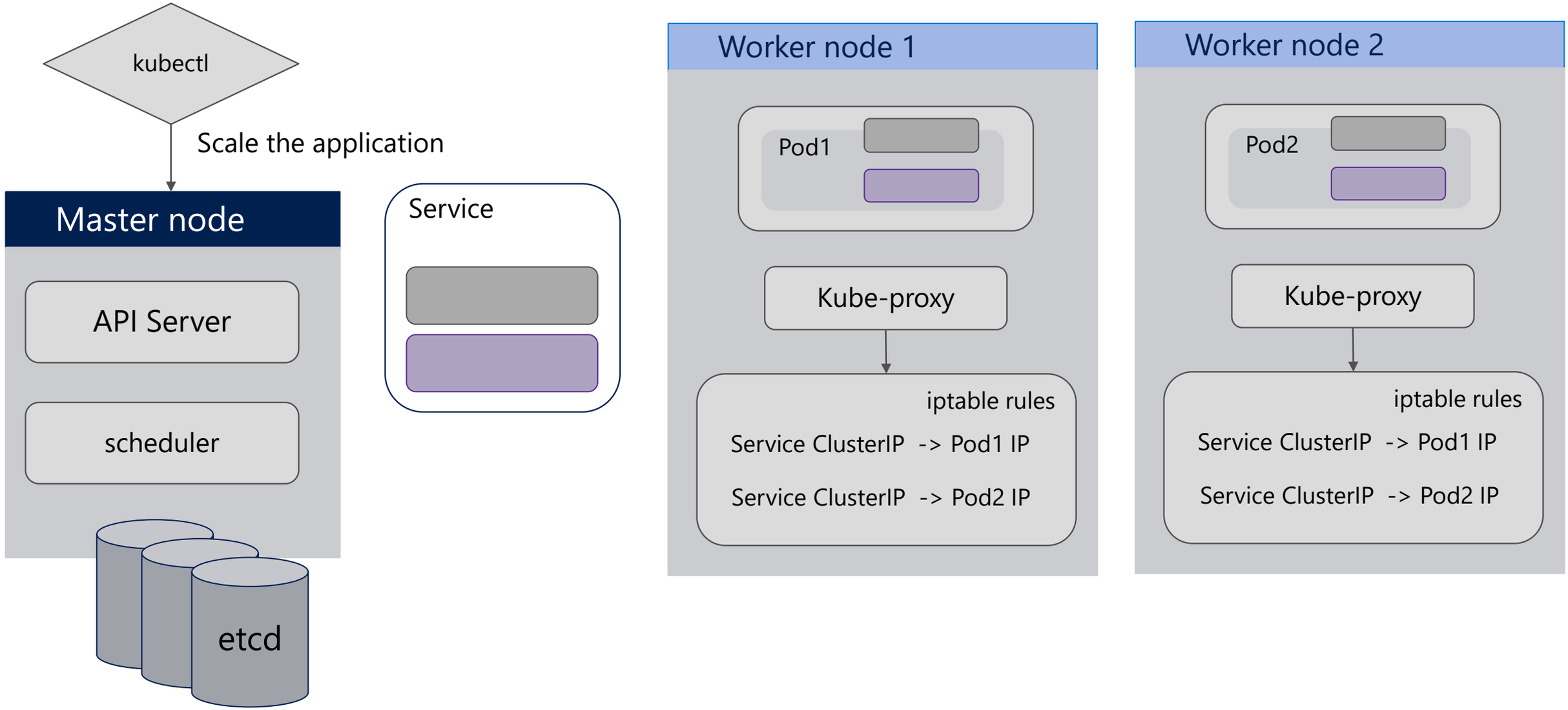




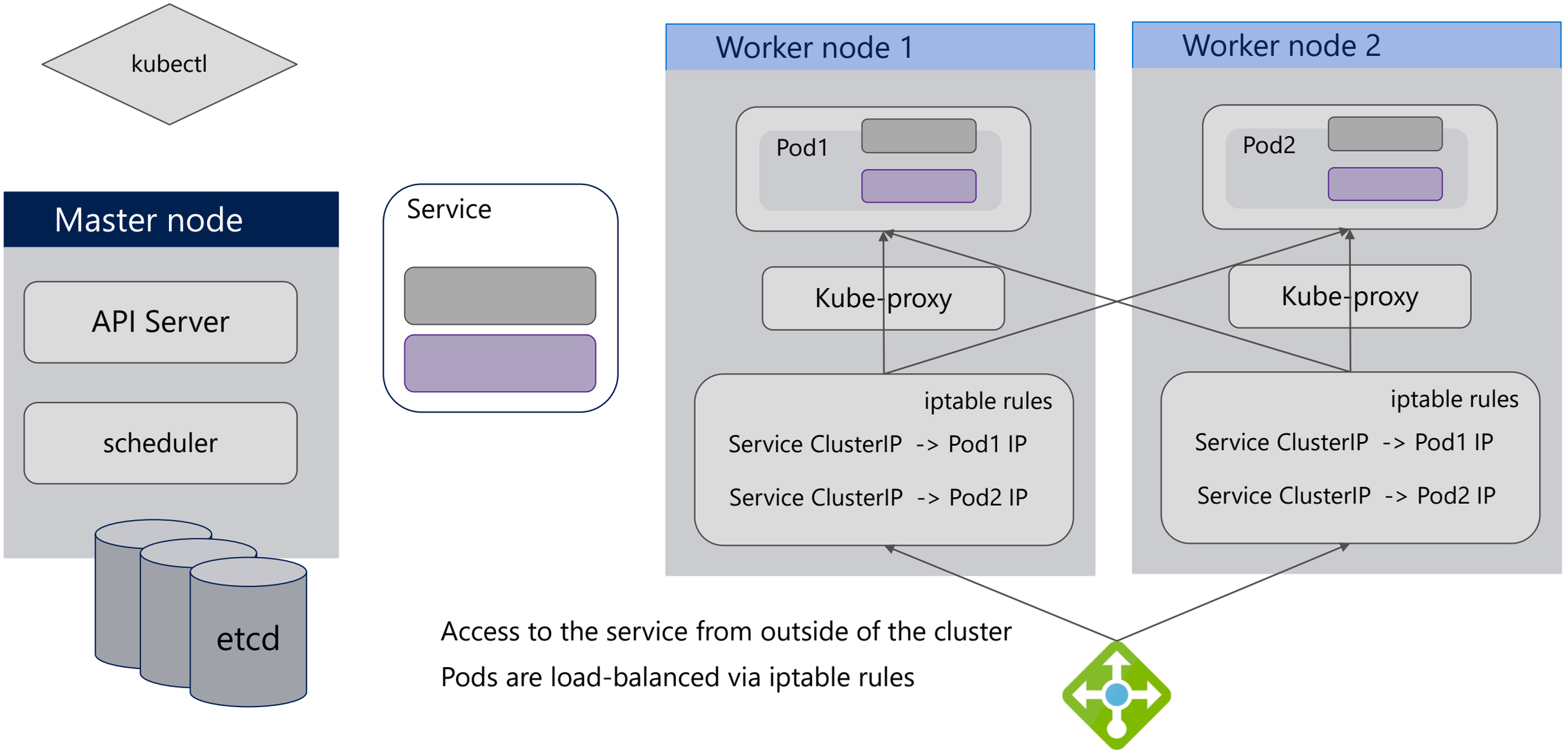
How Services Work



How Services Work



How Services Work



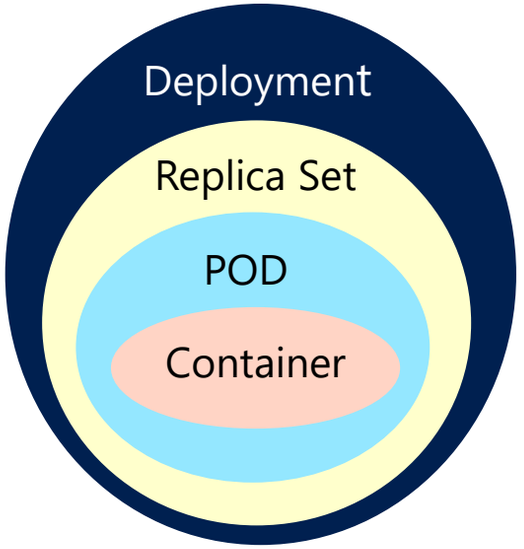
Access to the service from outside of the cluster

Pods are load-balanced via iptable rules

Demonstration: *Services*

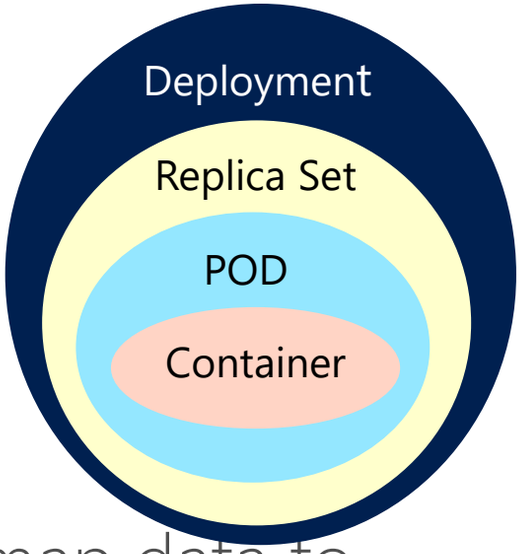
Kubernetes Services

What are ConfigMaps & Secrets?



- Expose configuration information to pods as environment variables
- Assigned at runtime as key-value pairs or contained in file
- Enables the same image to be re-used across environments
- Secrets contain sensitive information that is encoded
  - **Example: Connection Strings, Passwords**
- ConfigMaps contain non-sensitive information
  - **Example: The name of a backend component**

What are Volumes?



- Enable persistent data outside of a Pod
- Deleting or restarting a Pod destroys any persistent data stored in the containers file system
- Follows same concepts as in traditional containers- volumes map data to the underlying host
- There are different types of volumes:
  - Volume- mounts a volume on the underlying node
  - Persistent Volume- defines a static, external volume (e.g. Azure file, Azure disk)
  - Persistent Volume Claim (PVC)- reserves a set of memory from a Persistent Volume

Demonstration: *Volumes*

Volumes and Secrets

Knowledge Check

1. What is a “kubelet” and why is it needed?
2. What is the smallest deployable artifact in Kubernetes?
3. What are deployments?