IBM

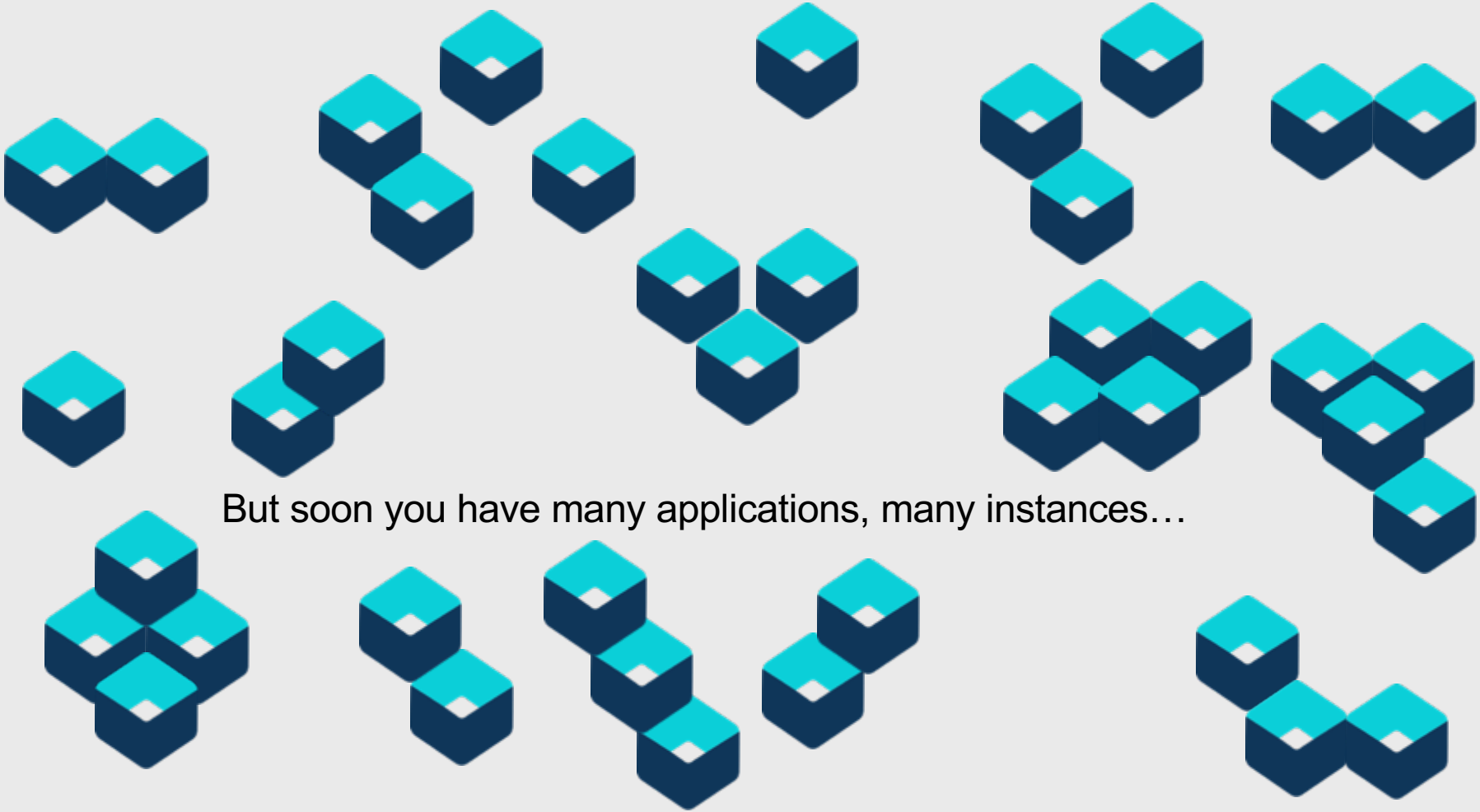
Kubernetes

Basics

IBM **Cloud**

Everyone’s container journey starts with one container….

At first the growth is easy to handle….



But soon you have many applications, many instances…

And that is why there is container orchestration

What is container orchestration?

Management of the deployment, placement, and lifecycle of workload containers

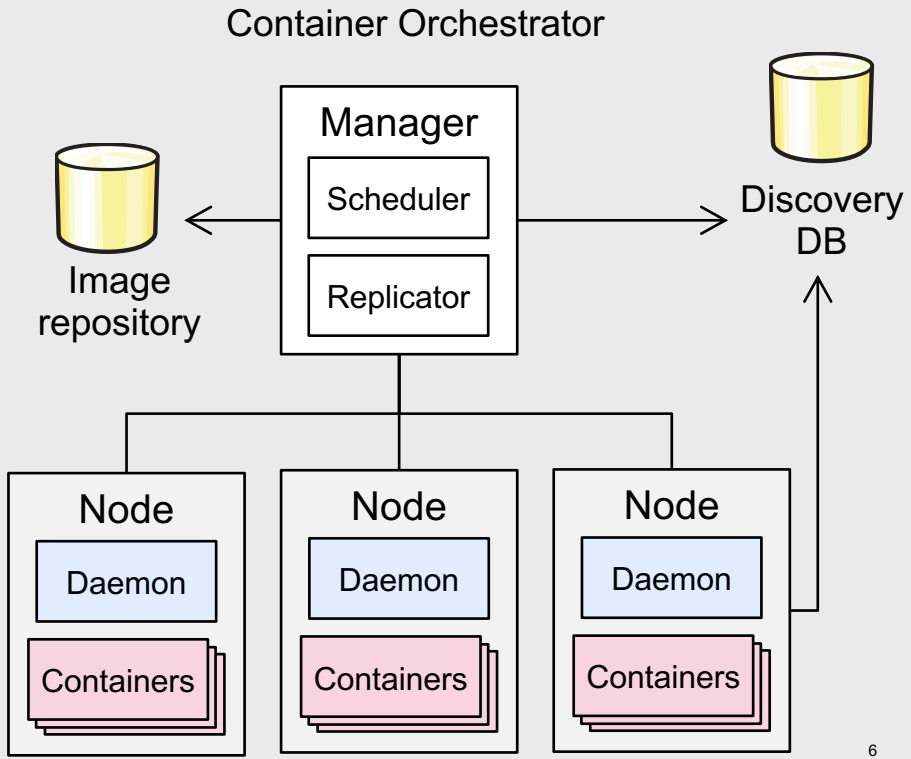
Cluster management creates unified targets for varied workload

Scheduling intelligently distributes containers across nodes

Service discovery knows where containers are located and provides a method to direct requests to them

Replication allows the workload to be scaled

Health management creates a method to ensure the application is assured to be viable by allowing unhealthy containers to be replaced



Kubernetes and Container Orchestration

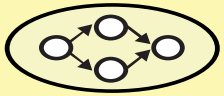
Kubernetes is a Container Orchestrator

- Provisions, manages, and scales containerized applications
- Supports:
  - Automated scheduling and scaling
  - Zero downtime deployments
  - High availability and fault tolerance
  - A/B deployments
- Manage infrastructure resources needed by applications
  - Volumes
  - Networks
  - Secrets
  - And many many many more..
- Declarative model
  - Provide the "desired state" and Kubernetes will make it happen

Kubernetes fun facts:

- Builds upon 15 years of experience of running production workloads at Google
- Open Governance via CNCF
- Adopted by IBM, Amazon, Microsoft, Red Hat, Google,
- Means “helmsman” in Greek

Container ecosystem

7		Application workflow	
6		Container orchestration	Kubernetes, Docker Swarm, Apache Mesos
5		Container scheduling	
4		Container engine	Docker
3		Operating system	Ubuntu, RedHat, CoreOS
2		Virtual infrastructure	VMWare, AWS
1		Physical infrastructure	

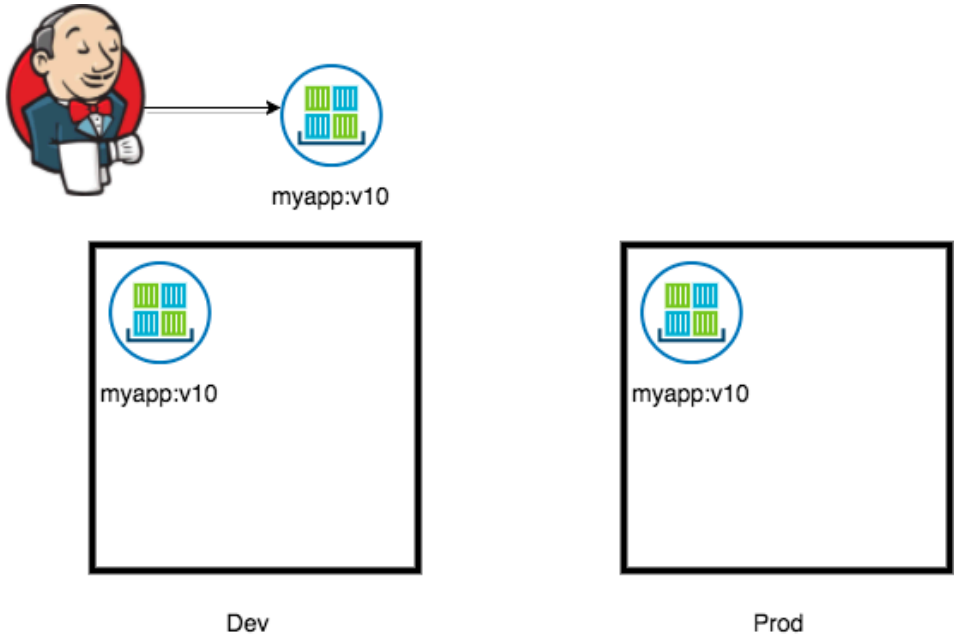
Kubernetes

The Building Blocks

Immutability

Build Once - Deploy Everywhere

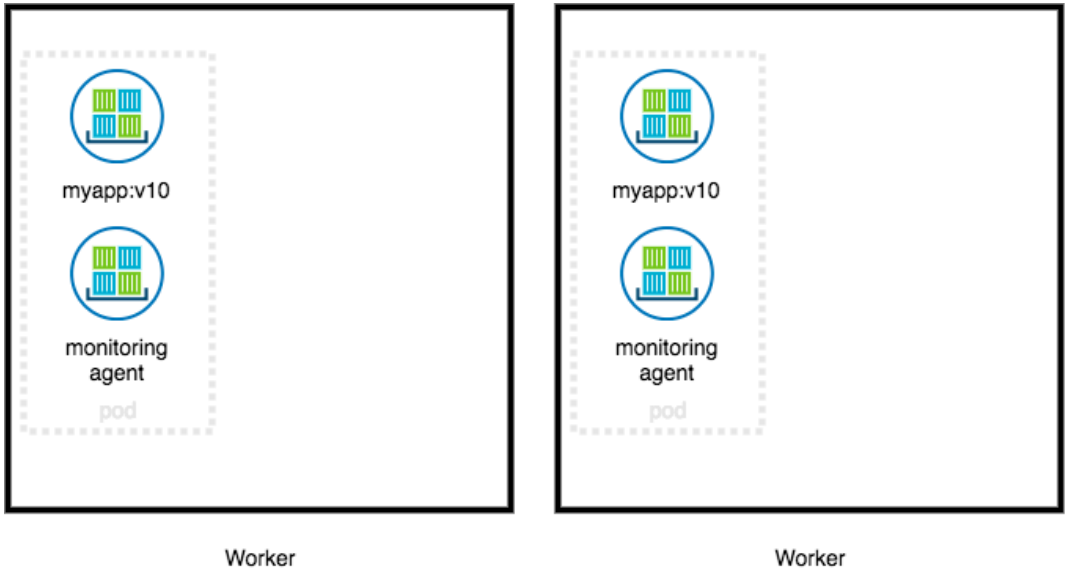
The same container image is built once and is moved between environments



Pod

A single unit of work in Kubernetes, which may consist of one or more containers

All containers in a pod are co-located and co-scheduled, and share the kernel namespace (process, storage, network, etc.)



Pod Health Checking

Pods are automatically kept alive by “process check” checking the basic status of the main process for the application

To go beyond this Kubernetes allows you to create a liveness probe to provide additional means for identifying health.

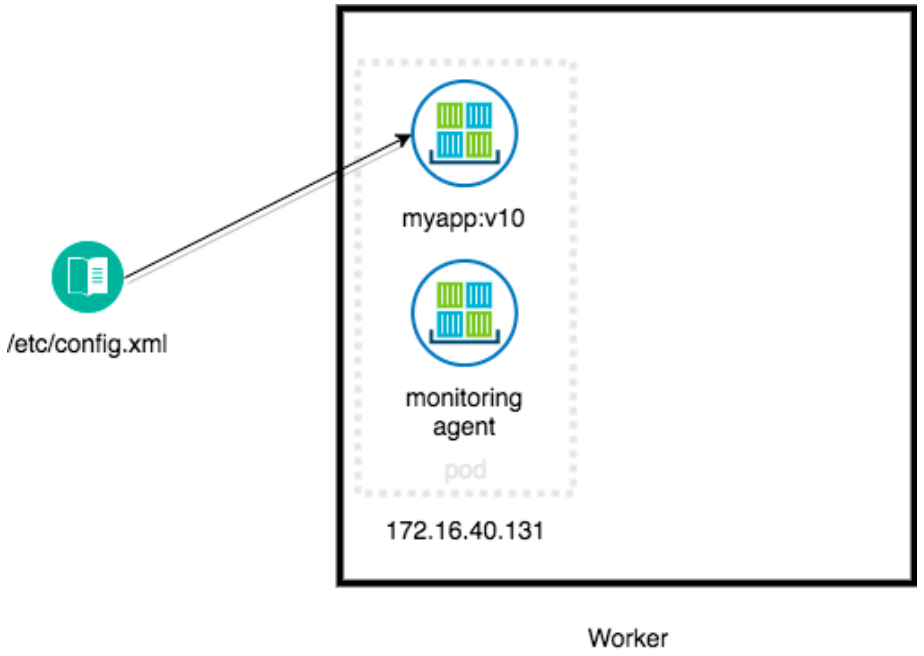
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-http-healthcheck
spec:
  containers:
  - name: nginx
    image: nginx
    # defines the health checking
    livenessProbe:
      # an http probe
      httpGet:
        path: /_status/healthz
        port: 80
      # length of time to wait for a pod to initialize
      # after pod startup, before applying health checking
      initialDelaySeconds: 30
      timeoutSeconds: 1
    ports:
    - containerPort: 80
```

Config Maps & Secrets

Share and store configurations, credentials and more

Store the configurations and secrets (credentials, certificates) in the K8s environment and mount them to the local filesystem within container(s)

The container image can move un-changed between environments (i.e. container immutability)

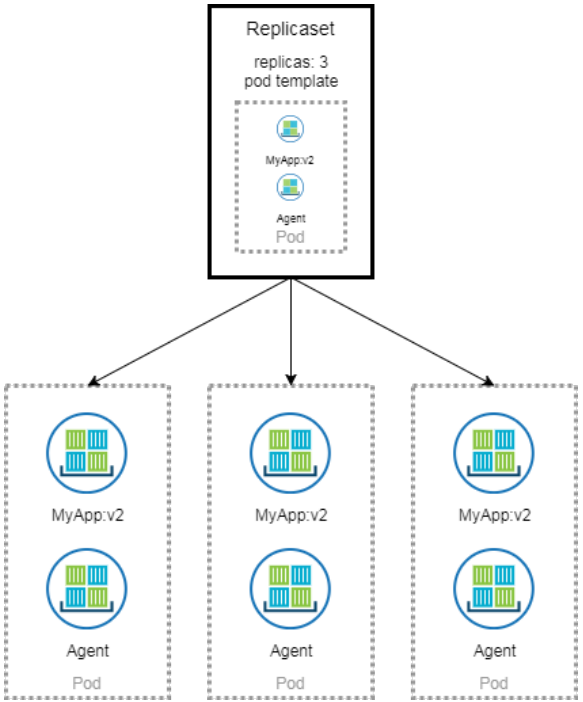


Replicaset

Scale pods horizontally and provide resiliency

Replicasets run one-to-many instances of the desired pod

When possible the replica pod should be stateless or near-stateless



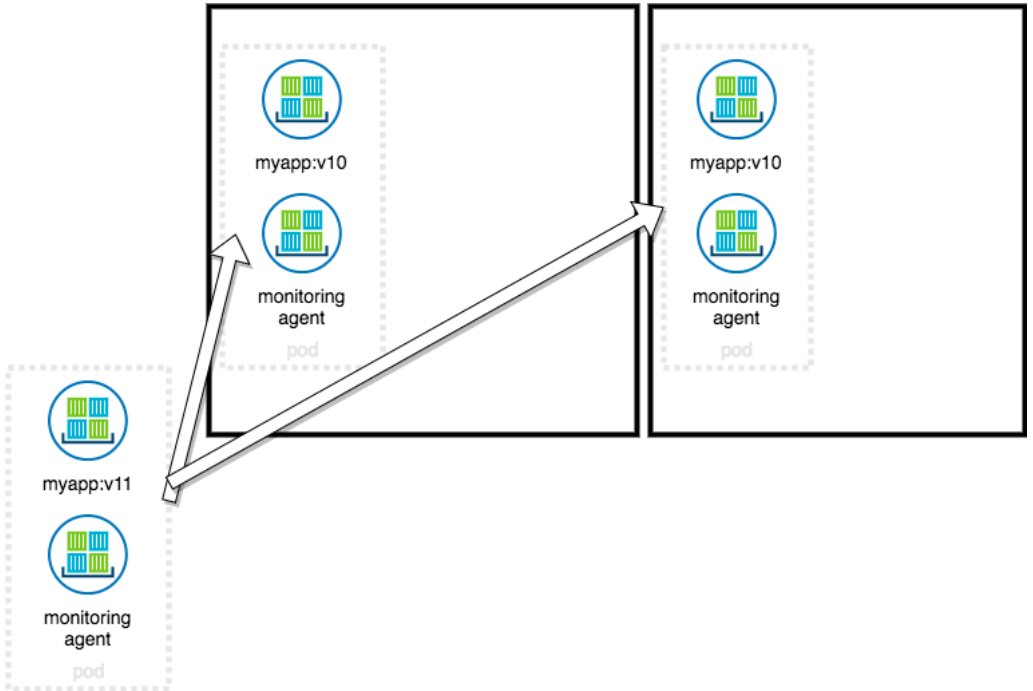

```yaml
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # this replicas value is default
  # modify it according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
    matchExpressions:
      - {key: tier, operator: In, values: [frontend]}
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google_samples/gb-frontend:v3
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and uncomment the
          # line below.
          # value: env
        ports:
        - containerPort: 80
```

Deployments

Deployments manage rolling updates to ReplicaSets and StatefulSets

When a new version of the application is available, the Deployment provides the ability to scale down the previous version of the application and scale up the new version in a controlled fashion with zero downtime

Enables rollback in the case of failure

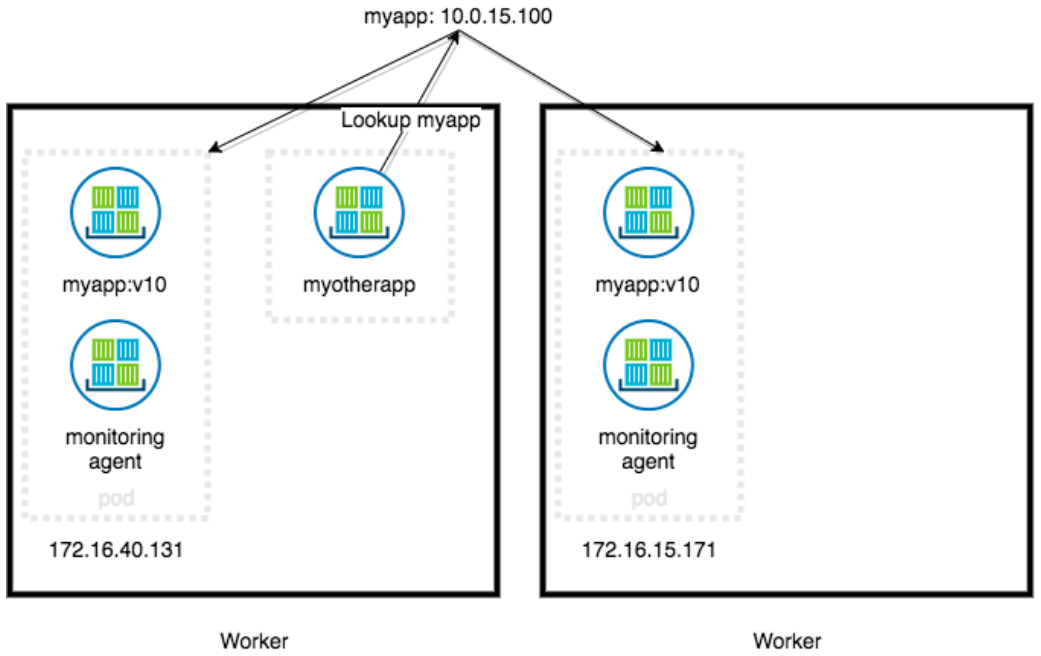


Service Discovery

Kubernetes has an internal DNS that is used as a Service Registry.

A Service resource in Kubernetes results in an entry in the internal DNS

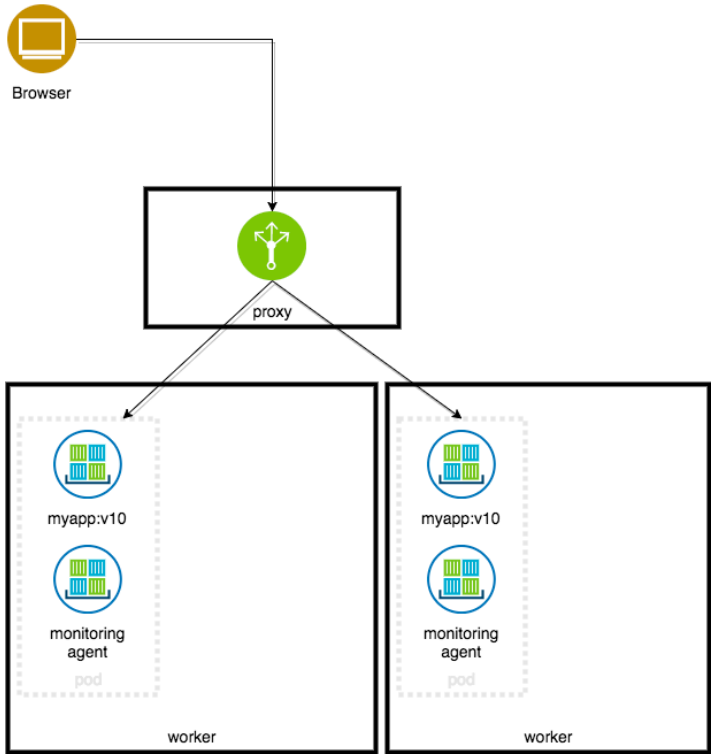
By default, a Service points to an internal Cluster IP that load balances between a set of healthy running pods



Ingress Resources

External access to applications running in Kubernetes may be enabled through Ingress resources and proxy nodes

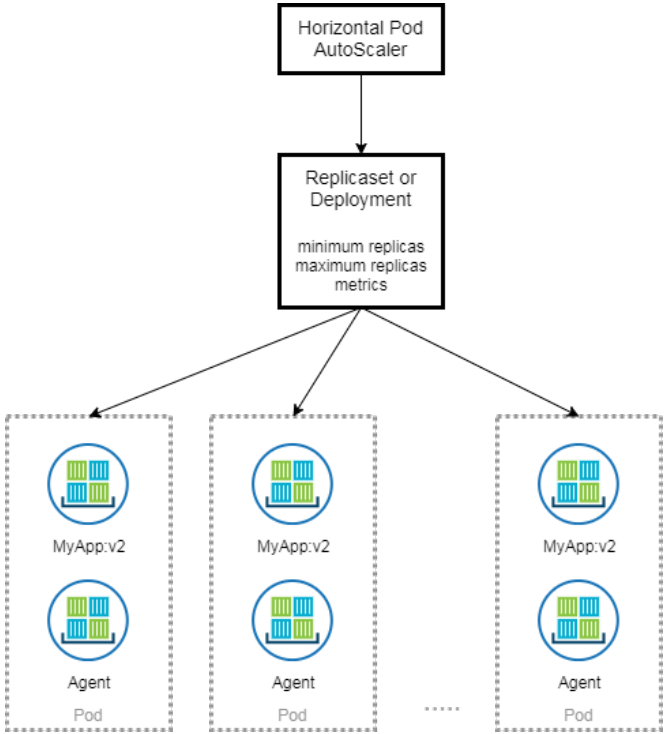
The proxy node(s) in ICP expose services defined in Kubernetes



More on Scaling

Horizontal Pod Auto-scaling (HPA)

Allows you to scale the number of running pods in a replicaset based upon resource (or application custom) metrics




```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
status:
  observedGeneration: 1
  lastScaleTime: <some-time>
  currentReplicas: 1
  desiredReplicas: 1
  currentMetrics:
  - type: Resource
    resource:
      name: cpu
      currentAverageUtilization: 0
      currentAverageValue: 0
```

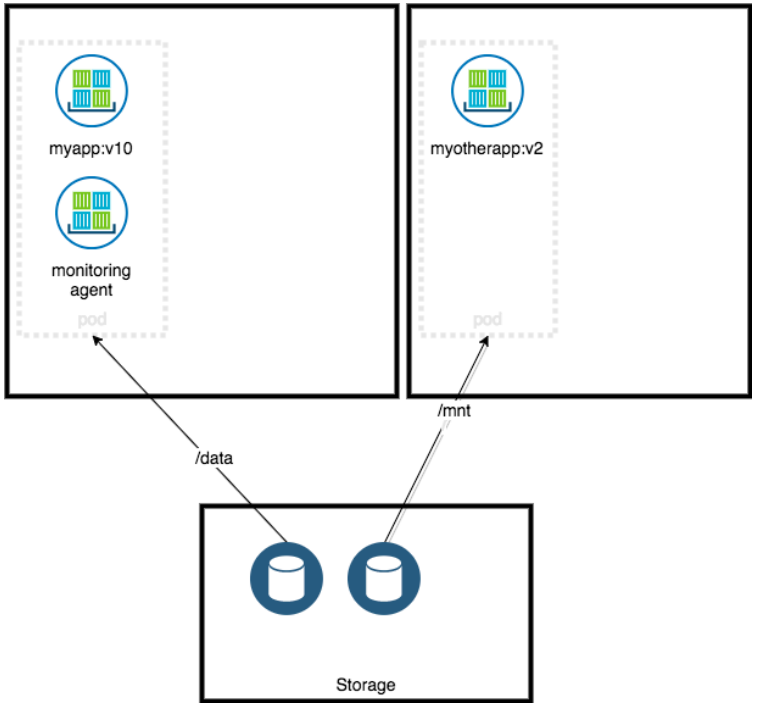
Persistence & Storage

There are many types of persistent storage and many provider options

Some pods must be able to persist data so that if Kubernetes restarts them on the same or another node data loss is avoided

Kubernetes will re-attach the shared storage when the pod (re)starts

Storage providers support different retention and recycling policies and the definitions of these are not universal



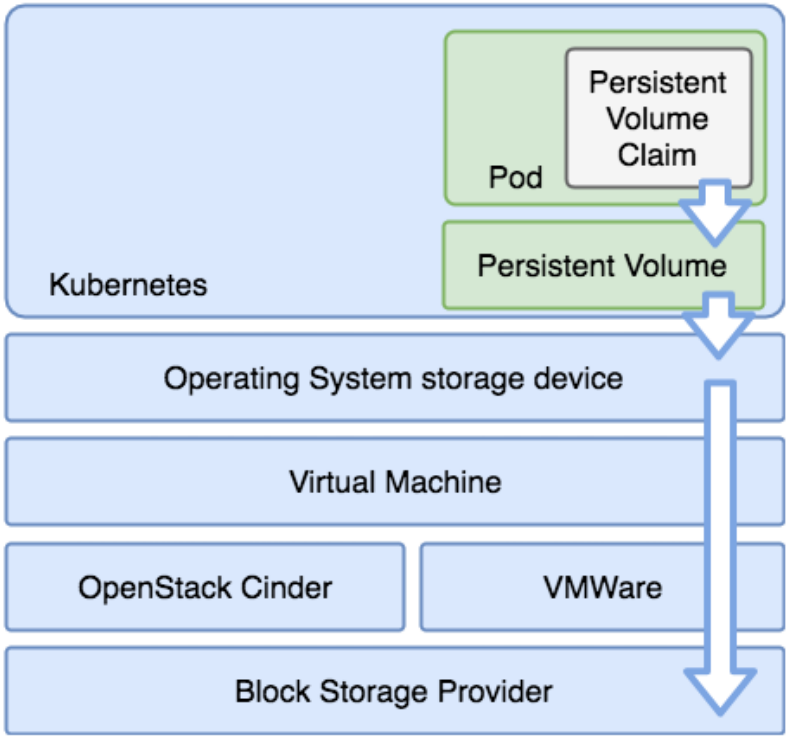
Persistent Storage

Solution components

Persistent Volume is a storage resource within the cluster. PVs have a lifecycle independent of any individual pod that uses it. This API object encapsulates the details of the storage implementation or cloud-provider-specific storage system.

A **Persistent Volume Claim** is a storage request, or claim, made by the developer. Claims request specific sizes of storage, as well as other aspects such as access modes.

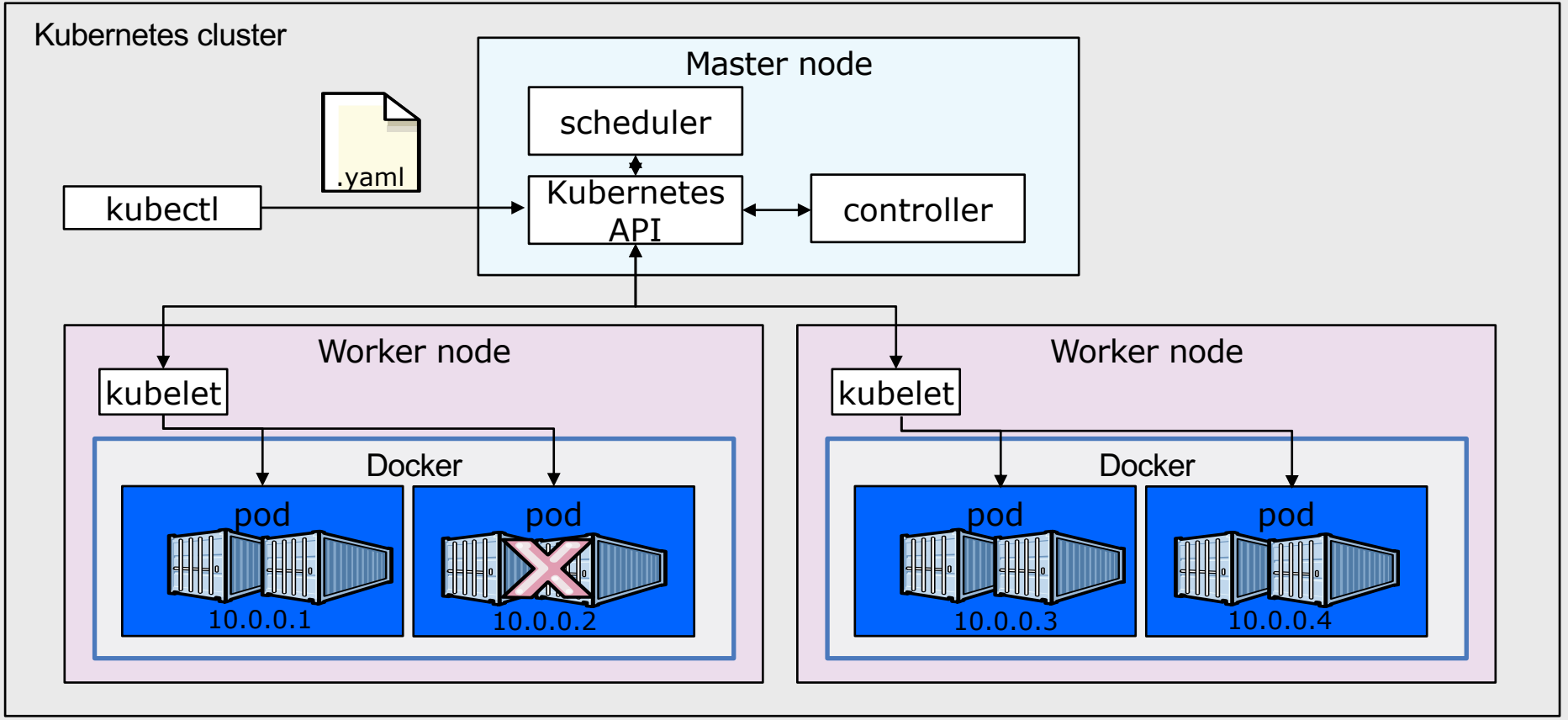
A **StorageClass** describes an offering of storage and allow for the dynamically provisioning of PVs and PVCs based upon these controlled definitions.



Kubernetes

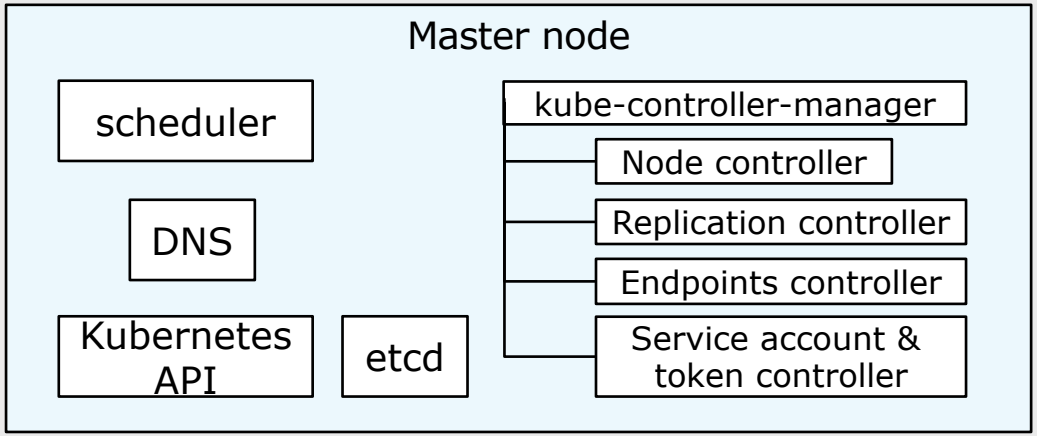
The Cluster

Kubernetes cluster architecture



Master Node components

- Make scheduling decisions for the cluster, and respond to cluster events, like a node failure
- Can run on any node in the cluster, but typically all master components run on the same virtual machine (vm), and do not run any container apps on that vm



Master Node Components

Etcd

- A highly-available key value store
- Stores all cluster data

API Server

- Exposes API for managing Kubernetes
- Used by kubectl CLI

Scheduler

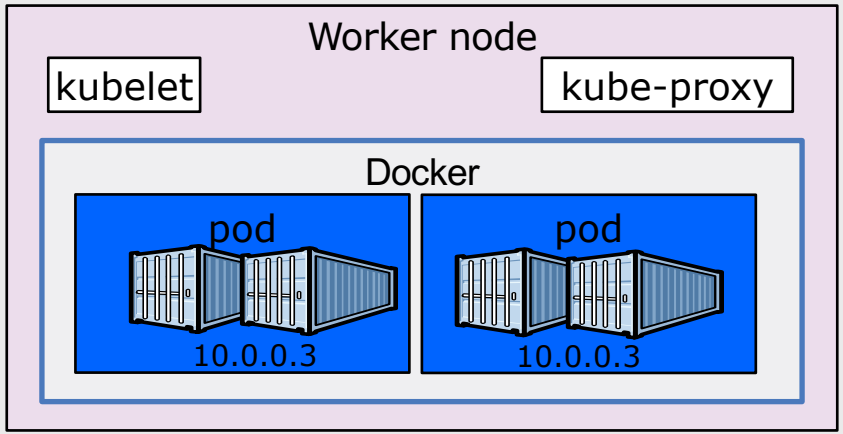
- Selects the worker node for each pods runs

Controller manager

- Daemon that runs controllers (background threads that handle routine tasks in the cluster)
- Node Controller – Responsible for noticing and responding when nodes go down
- Endpoints Controller – Populates the Endpoints object (joins services and pods)
- Service Account and Token Controllers – Create default accounts and API access tokens for new namespaces

Worker Node Components

- Provide the Kubernetes runtime environment; run on every node
- Maintain running pods



Kubernetes Clients (CLI and Dashboard)

Kubernetes CLI

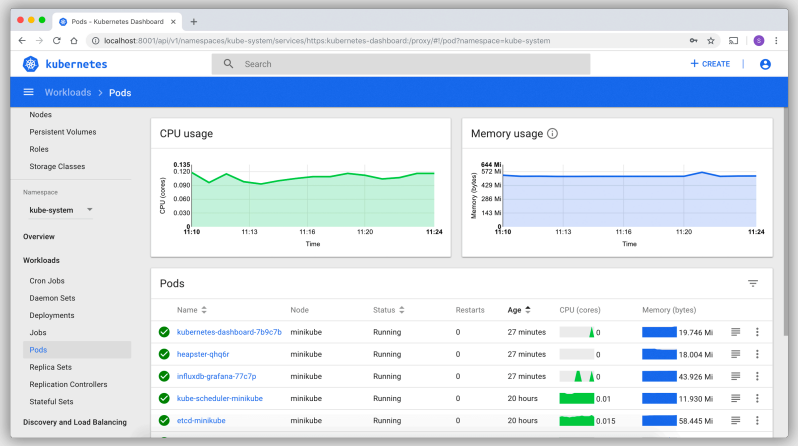
- Directly manipulate YAML
  - `kubectl (create|get|apply|delete) -f myResource.yaml`

- https://kubernetes.io/docs/tasks/tools/install-kubectl

```
> kubectl
```

Kubernetes Dashboard

- Another way to view and modify resources



Kubernetes in Action!

1. User via "**kubectl**" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a kubelet
6. Kubelet detects pods and deploys them via the container runtime (e.g. Docker)
7. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing