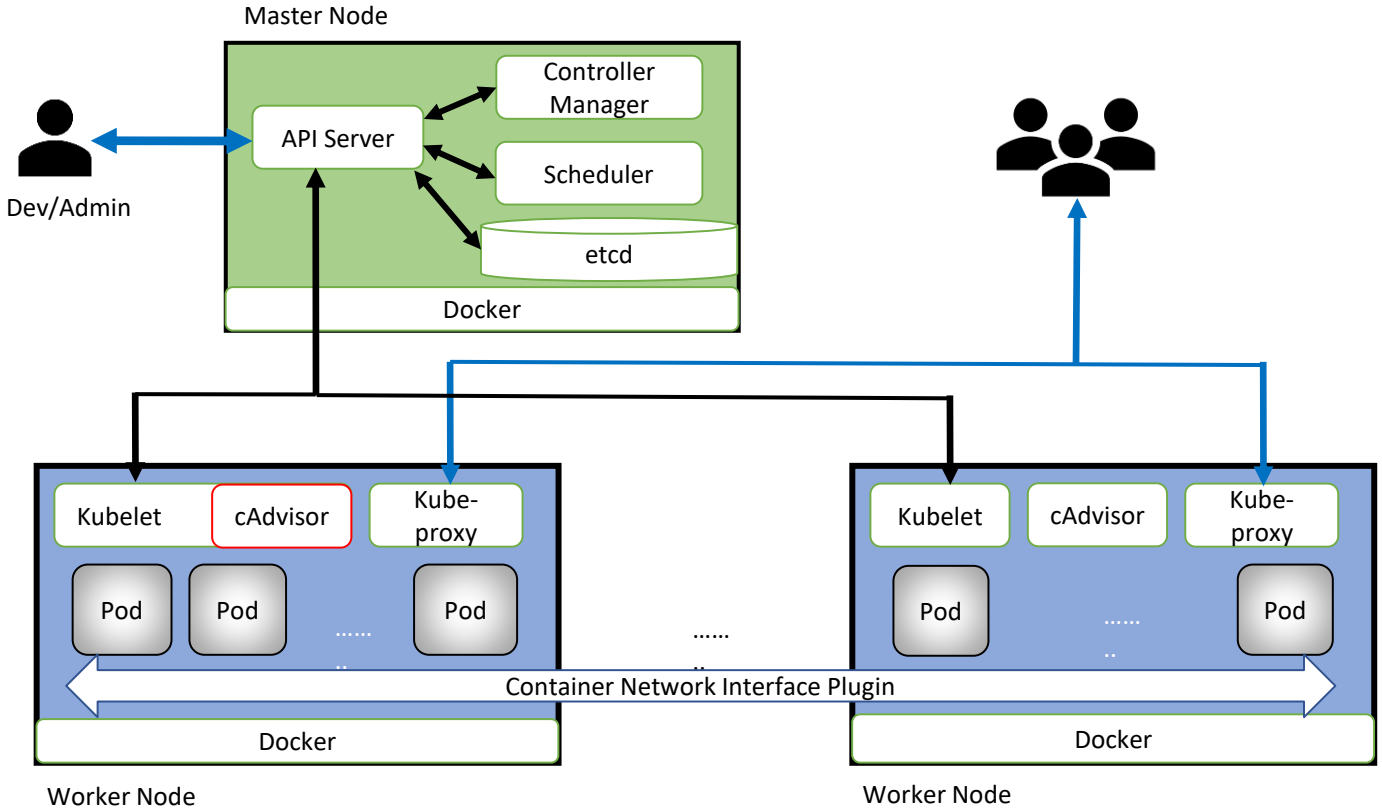
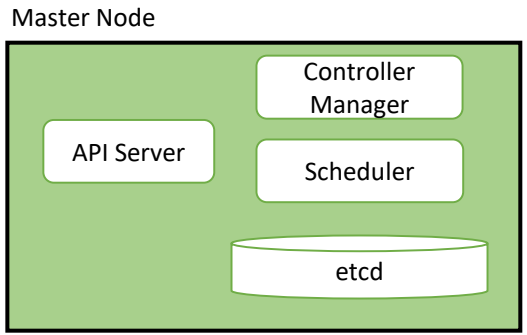
Kubernetes Architecture

Kubernetes Architecture



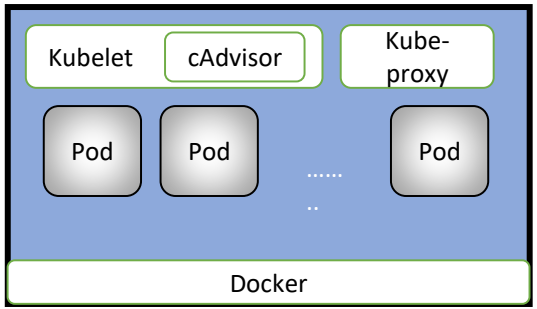
Master Node Architecture

- **API Server**: Configures and validates data for api objects like pods, services. Its a front-end of control plane
- **Scheduler**: It decides where in the cluster the workloads are to be run
- **etcd**: Stores all cluster-related data
- **Controller**: Daemon that embeds core control loops that regulates system state via routine tasks



Worker Node Architecture

- **kubelet:** Primary node agent which performs various tasks like mounting volumes, running containers, etc. for pods assigned to the node
- **kube-proxy:** Provides service abstraction and connection forwarding
- **Docker:** Container engines for running containers
- **cAdvisor:** Provides container users an understanding of the resource usage and performance characteristics of their running containers



Worker Node

Kubernetes Basics

K21Academy
Learn Cloud from Experts

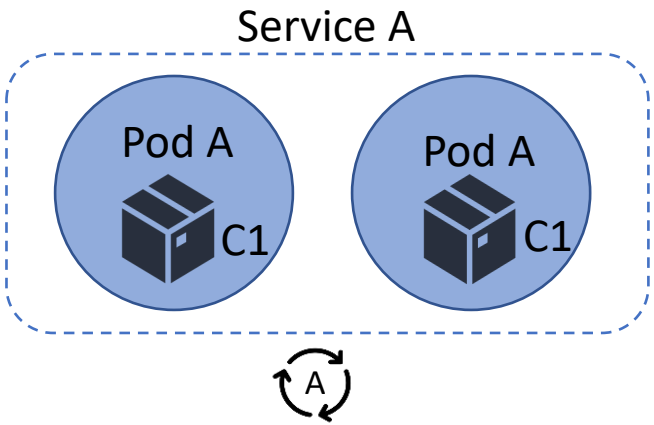
- Everything is an API call served by the *Kubernetes API server* (kube-apiserver)
- The API server is a gateway to an etcd datastore that maintains the desired state of your application cluster
- To update the state of a Kubernetes cluster, we make API calls to the API server describing our desired state
- Once we've declared the desired state of your cluster using the API server, controllers ensure that the cluster's current state matches the desired state by continuously watching the state of the API server and reacting to any changes
- Controllers operate using a simple loop that continuously checks the current state of the cluster against the desired state of the cluster
- If there are any differences, controllers perform tasks to make the current state match the desired state.

K21Academy
Learn Cloud from Experts

Kubernetes Building Blocks

Kubernetes Object

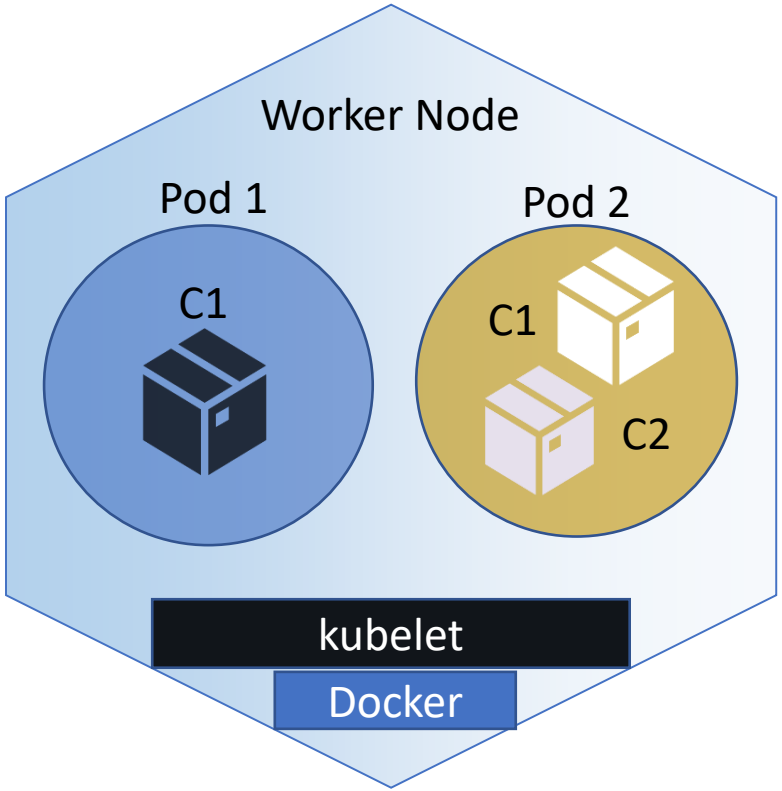
- Anything that persists in the system
- It defines desired state of the system
- Pod
- Deployment
- Service



Namespace

- A way to divide cluster resources
- Names of resources need to be unique within a namespace
- Kubernetes starts with three initial namespaces:
  - default - The default namespace for objects with no other namespace
  - kube-system - The namespace for objects created by the Kubernetes system
- Each Kubernetes resource can only be in one namespace

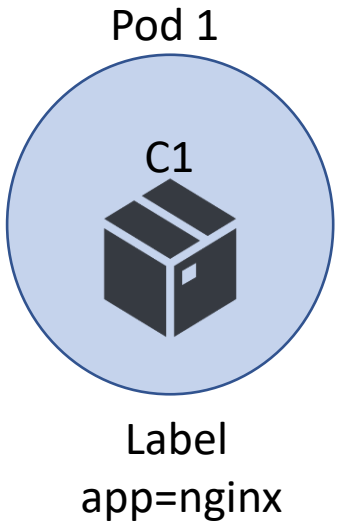
Kubernetes Pod



- Pod is the atomic unit on the K8s platform
- Pod can be a group of one or more application containers
- Pod has a unique Ip Address
- Containers in a pod share the IP address and port namespace

Label and Selector

- Labels are key/value pairs that are attached to objects
- Labels are intended to be used to specify identifying attributes of objects
- Labels can be used to organize and to select subsets of objects
- Selectors match the labels to connect to K8s objects



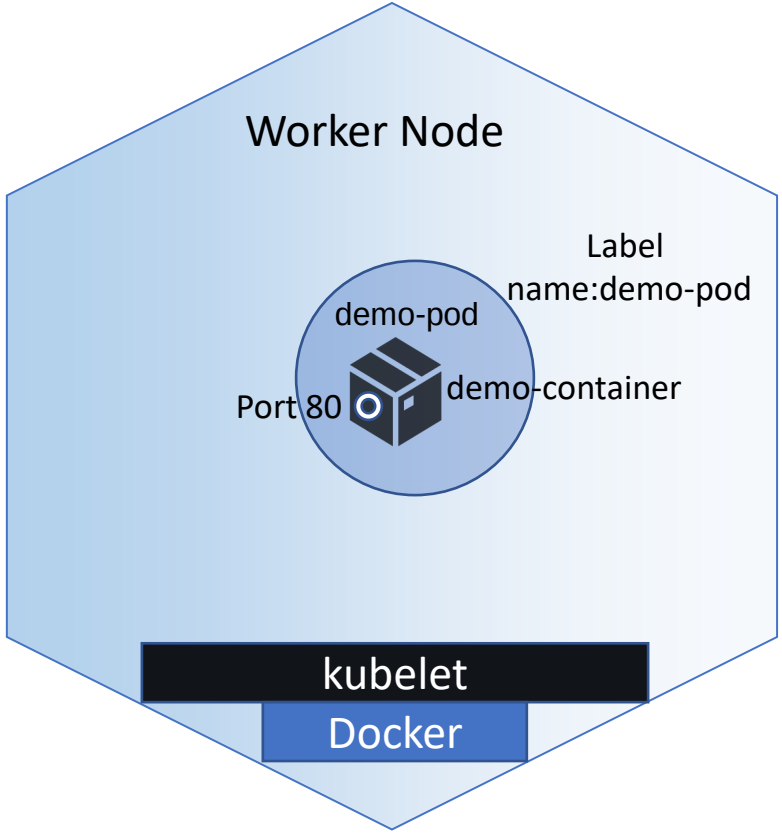
Annotations

- Kubernetes labels and annotations are both ways of adding metadata to Kubernetes objects
- To attach arbitrary non-identifying metadata to objects
- Labels allow you to identify, select and operate on Kubernetes objects
- Annotations are not used to identify and select objects
- The metadata in an annotation can be small or large, structured or unstructured, and can include characters not permitted by labels
- Eg: Build, release, or image information like timestamps, release IDs, git branch, PR numbers, image hashes, and registry address

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: annotations-demo
  annotations:
    imageregistry: "https://hub.docker.com/"
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
```

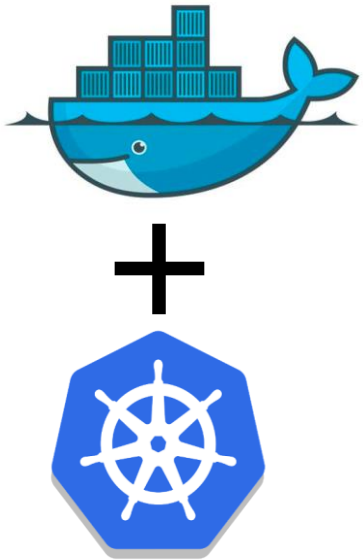
Pod Definition

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



Pod Limitations

- IP Address is not persistent - Service
- Memory is not persistent - Persistent Volume
- Can restart or get deleted - Controller
- Need to have a management layer to manage

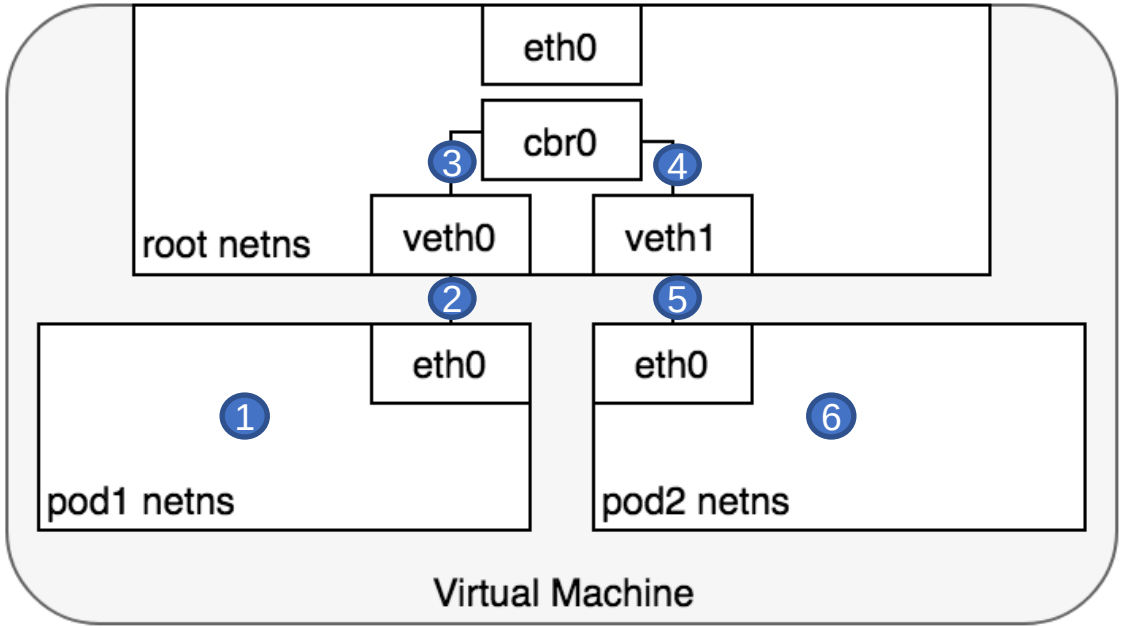


K21Academy
Learn Cloud from Experts

Kubernetes Networking

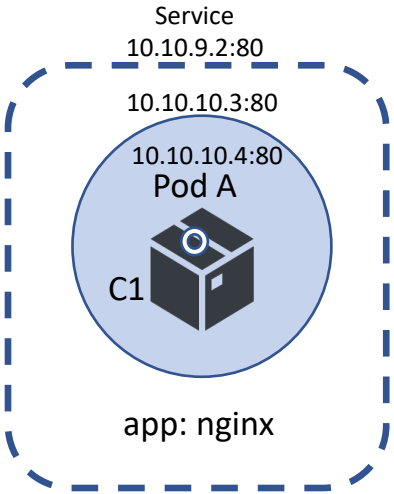
Pod-to-Pod Networking – Same Node

- Every Pod has a real IP address and each Pod communicates with other Pods using that IP address



Kubernetes Service

- Service in Kubernetes aims to solve the connectivity issue
- Service is an abstraction which defines a logical set of Pods along with policies with which to access them
- Service has a static IP address
- With Label/Selector, Pod and Service match
- Service exposes the application Pod to other applications and if needed to external world
- Does load balance and health check

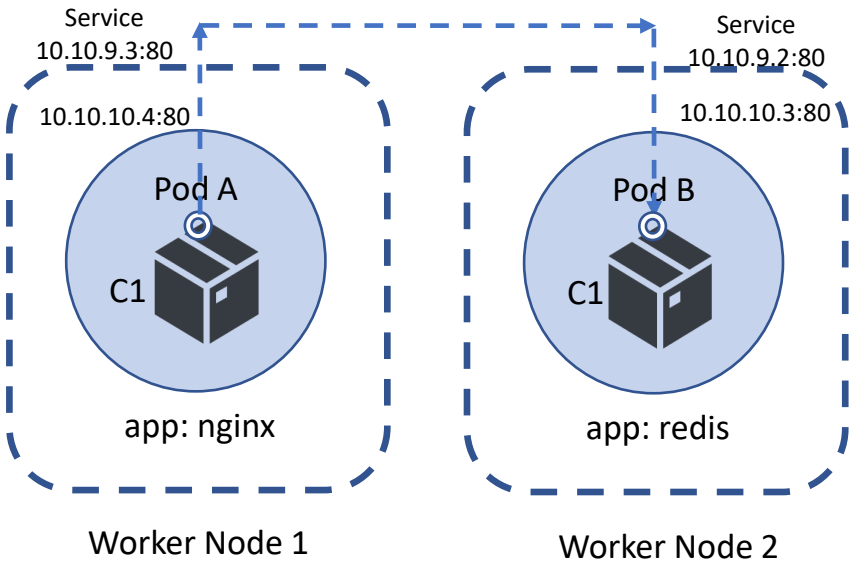


Kubernetes Service

- A Kubernetes Service manages the state of a set of Pods, allowing us to track a set of Pod IP addresses that are dynamically changing over time
- Services act as an abstraction over Pods and assign a single virtual IP address to a group of Pod IP addresses
- Any traffic addressed to the virtual IP of the Service will be routed to the set of Pods that are associated with the virtual IP
- This allows the set of Pods associated with a Service to change at any time — clients only need to know the Service's virtual IP, which does not change

Kubernetes Service - ClusterIP

- Exposes the Pod to only within the cluster
- Assigns a Private IP Address in Cluster IP range
- Can talk to applications only within the cluster
- Doesn't have external connectivity



Kubernetes Service - ClusterIP

```
$ kubectl expose pod demo-pod --port 80 --target-port 80 --type ClusterIP
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
demo-pod     ClusterIP   10.108.93.193   <none>        80/TCP    6s
```

Kubernetes Service - ClusterIP

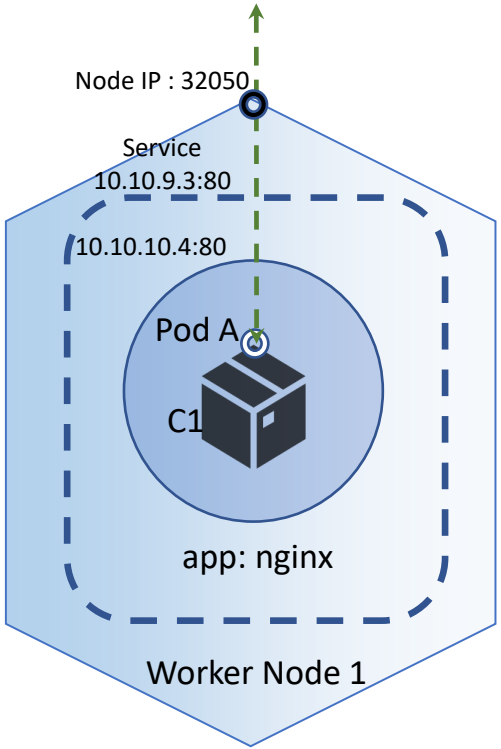
```
root@kubeadm-master:/home/ubuntu/Kubernetes#
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl describe svc demo-pod
Name:              demo-pod
Namespace:         default
Labels:            name=demo-pod
Annotations:       <none>
Selector:          name=demo-pod
Type:              ClusterIP
IP:                10.108.93.193
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.46.0.1:80
Session Affinity:  None
Events:            <none>
root@kubeadm-master:/home/ubuntu/Kubernetes#
```

```
apiVersion: v1
kind: Service
metadata:
  name: demo-pod
  labels:
    name: demo-pod
spec:
  type: ClusterIP
  ports:
  - port: 80
    protocol: TCP
  selector:
    name: demo-pod

~
~
~
```

Kubernetes Service - NodePort

- Exposes the Pod to outside the cluster
- Assigns a Private IP Address in Cluster IP range
- Assigns a port on the worker nodes to expose it outside cluster
- Provides external connectivity



Kubernetes Service - NodePort

```
$ kubectl edit svc demo-pod
```

```
spec:
  clusterIP: 10.108.93.193
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    name: demo-pod
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

```
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
demo-pod     NodePort    10.108.93.193   <none>        80:32561/TCP   18m
```

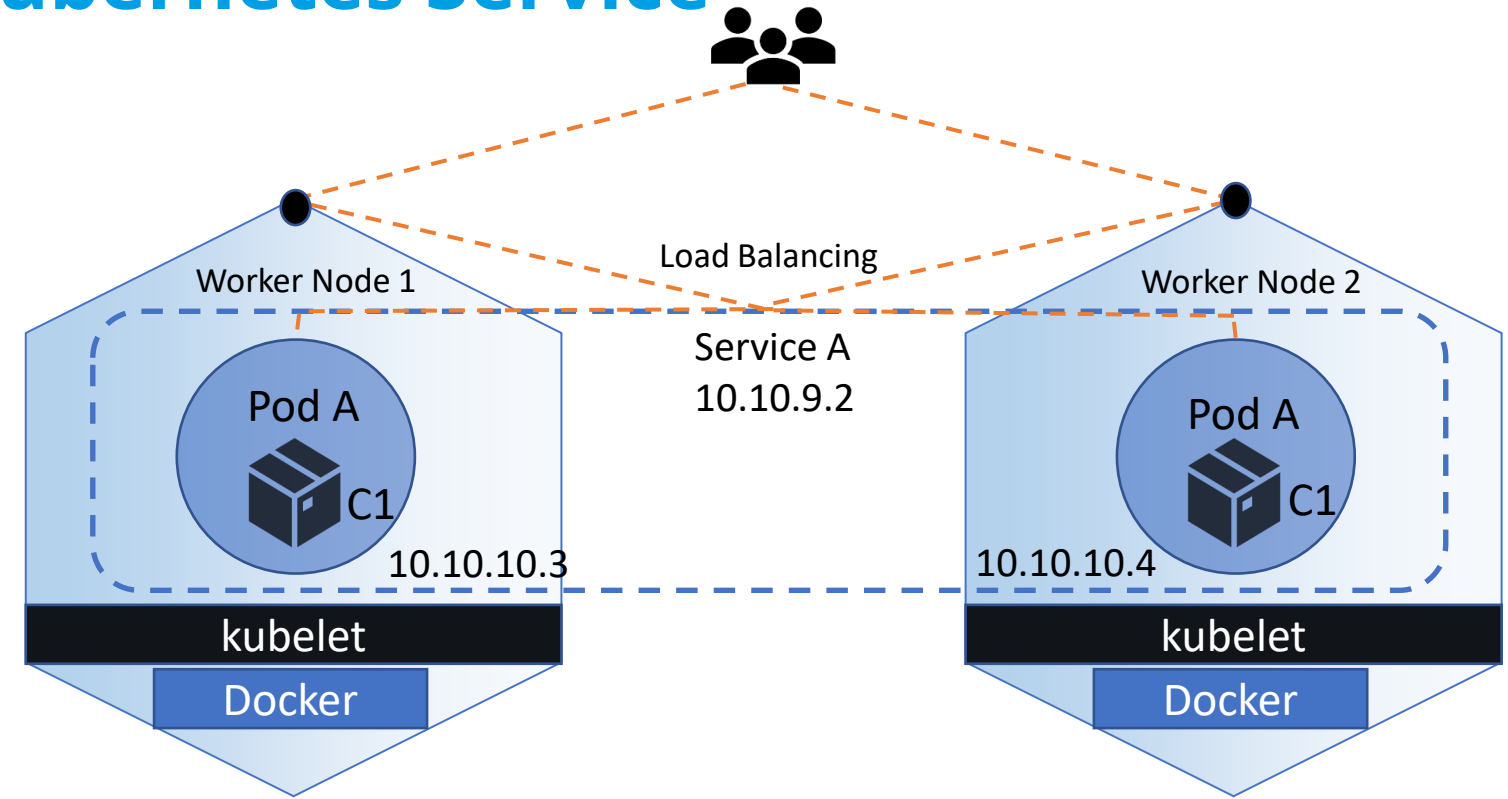
Kubernetes Service - NodePort

```
root@kubeadm-master:/home/ubuntu/Kubernetes#
root@kubeadm-master:/home/ubuntu/Kubernetes# kubectl describe svc demo-pod
Name:                     demo-pod
Namespace:                default
Labels:                   name=demo-pod
Annotations:              <none>
Selector:                 name=demo-pod
Type:                     NodePort
IP:                       10.108.93.193
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32561/TCP
Endpoints:                10.46.0.1:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
root@kubeadm-master:/home/ubuntu/Kubernetes#
```

```
apiVersion: v1
kind: Service
metadata:
  name: demo-pod
  labels:
    name: demo-pod
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
  selector:
    name: demo-pod

~
~
~
```

Kubernetes Service



Kubernetes DNS Service

- Kubernetes clusters automatically configure an internal DNS service to provide a lightweight mechanism for service discovery
- The full DNS A record of a Kubernetes service will look like the following example: **service.namespace.svc.cluster.local**
- A pod would have a record in this format, reflecting the actual IP address of the pod: 10.32.0.125.namespace.pod.cluster.local
- If you're addressing a service in the same namespace, you can use just the service name to contact it: **other-service**
- If the service is in a different namespace, add it to the query: **other-service.other-namespace**

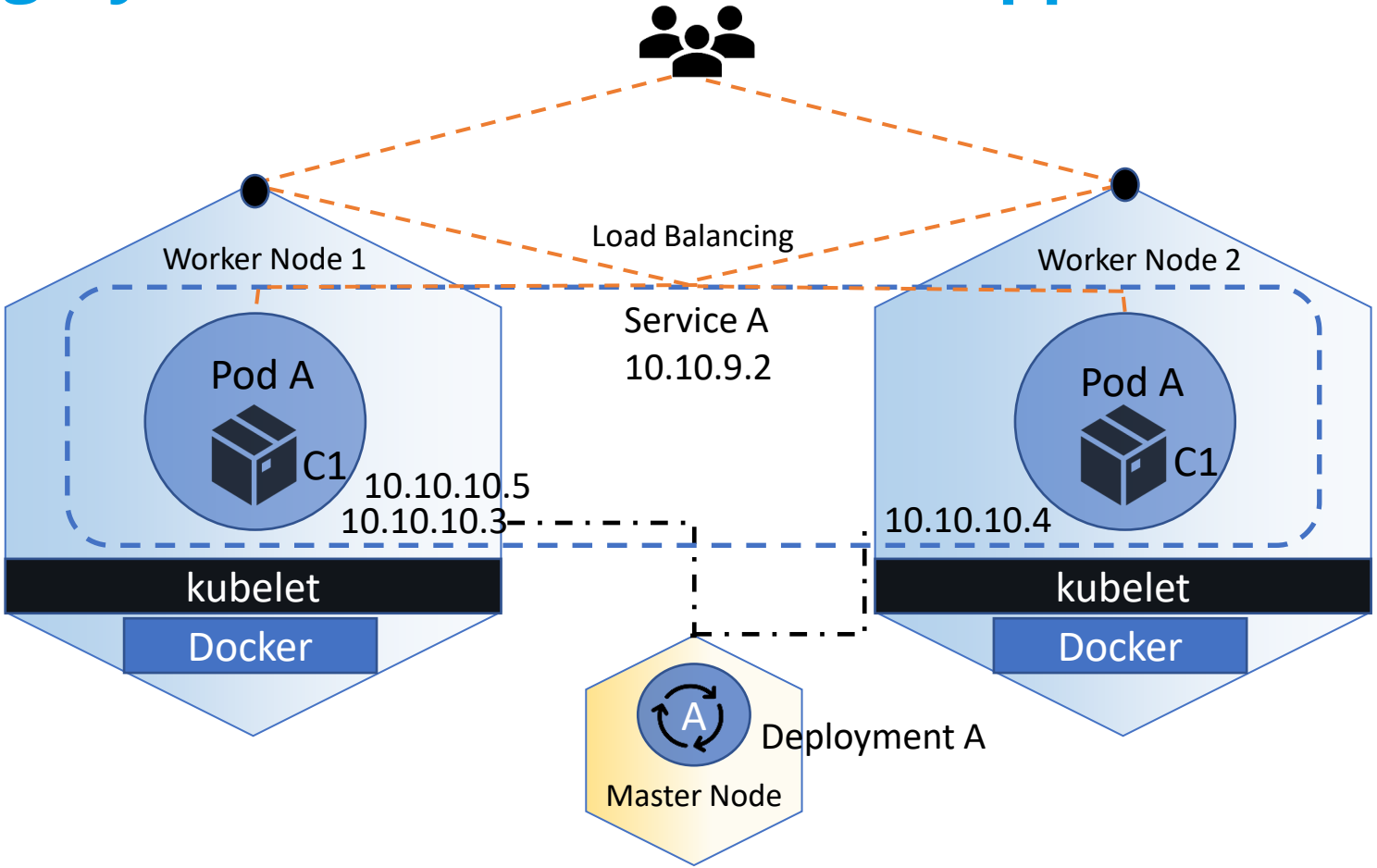
Deploy Highly Available Application

Kind: Deployment

- Deployment Controller defines the state of Deployment Objects, like Pods and ReplicaSets
- Deployments are used to create and deploy, scale, monitor and roll back
- Manage the state of Pods and ReplicaSets on the system
- Make application highly available, scalable and self-healing

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
        ports:
        - containerPort: 80
~
~
~
~
~
~
~
~
```

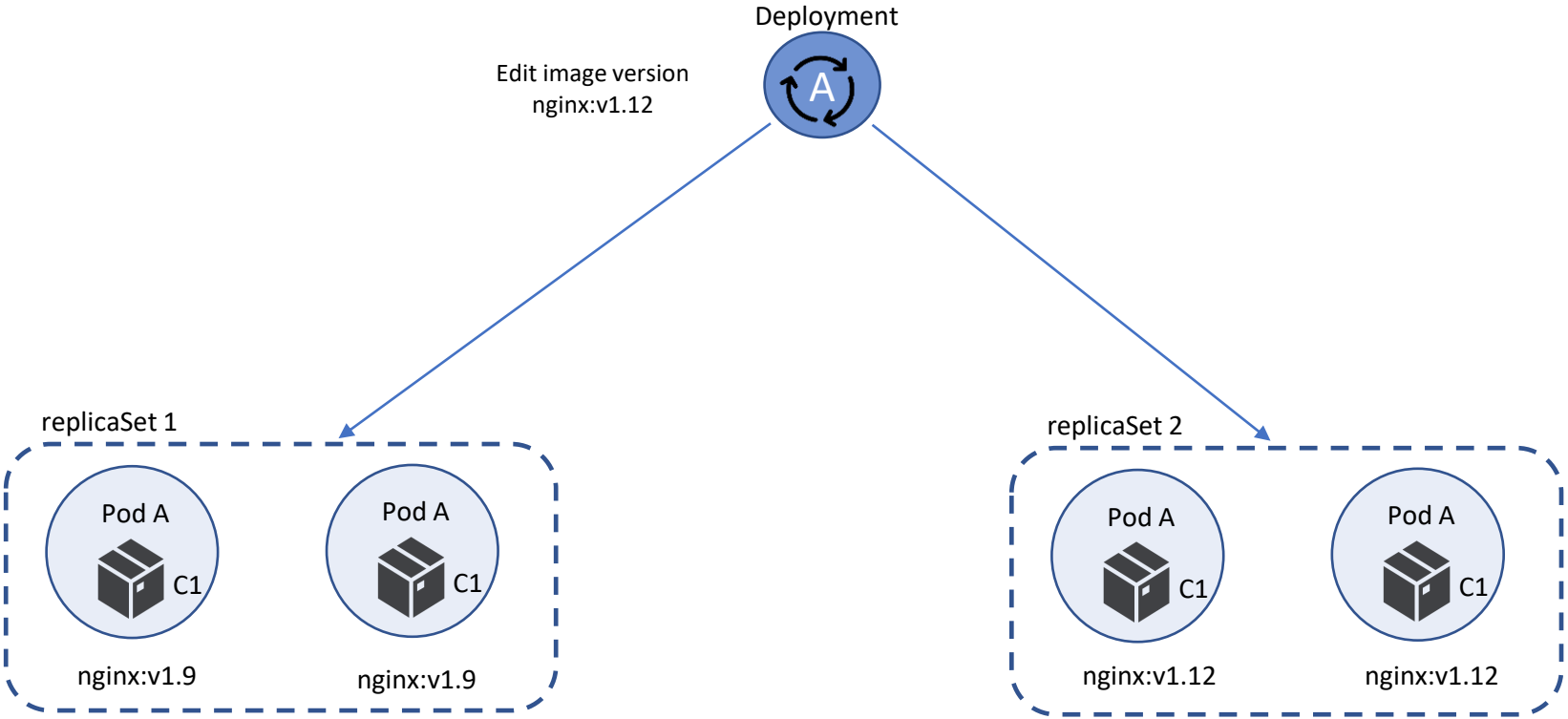
Highly Available and Scalable Application



Responsibilities - Deployment

- Changing or updating Pod state
- Scaling up of the deployment
- To rollout, ReplicaSet to create Pods and monitor its status
- Roll back to a previous Deployment
- Pause Deployment to fix template

Rolling Changes - Deployment



Deployment Strategy

- StrategyType:
  - "Recreate" or "RollingUpdate"
  - "RollingUpdate" is the default value
- RollingUpdate
  - **maxUnavailable:** An optional field that specifies the maximum number of Pods that can be unavailable during the update process
  - **maxSurge:** An optional field that specifies the maximum number of Pods that can be created over the desired number of Pods
  - The value can be an absolute number or %

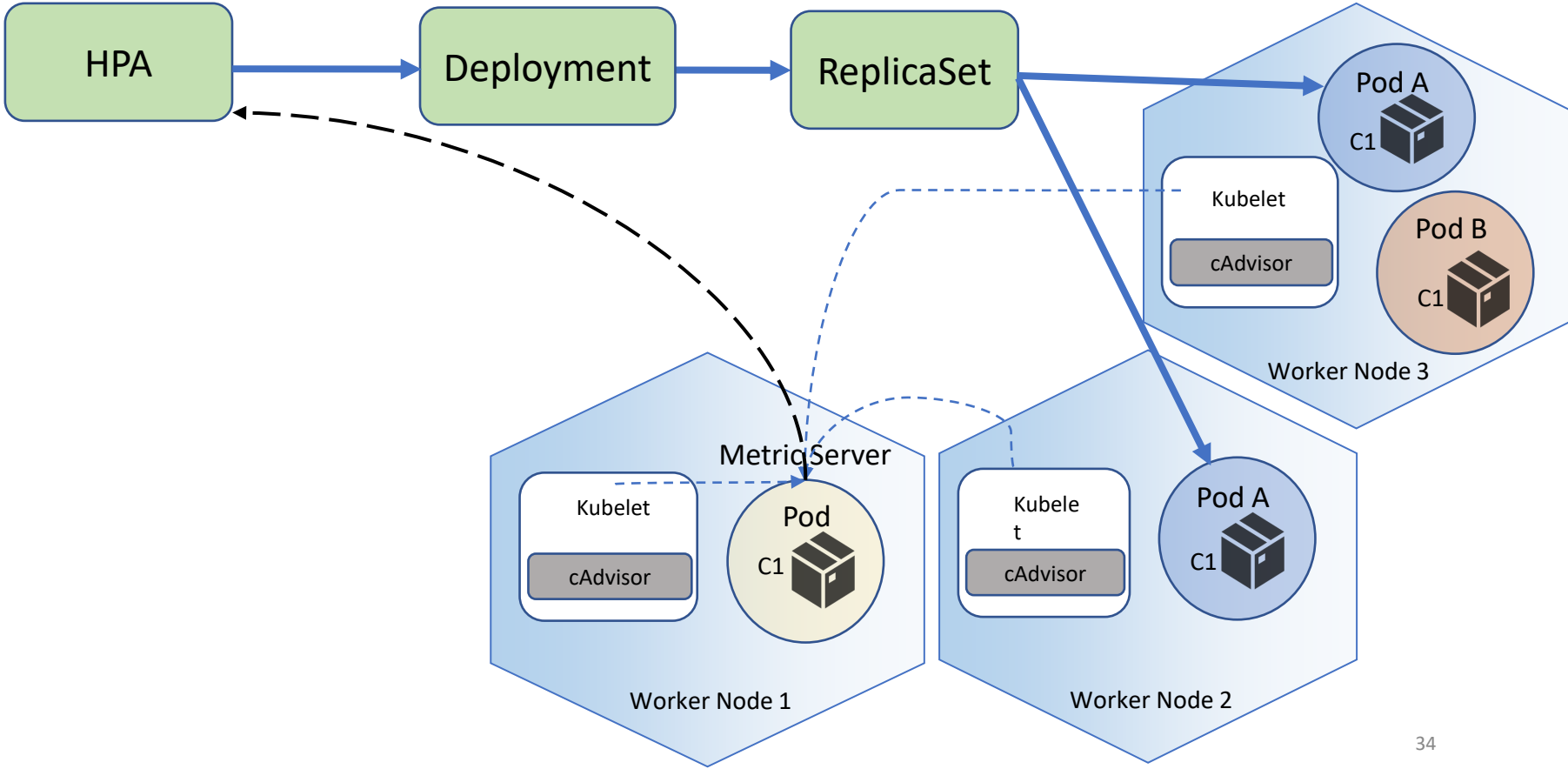
```
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Thu, 30 Nov 2017 10:56:25 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
   Containers:
    nginx:
      Image:        nginx:1.16.1
      Port:         80/TCP
      Environment:  <none>
      Mounts:       <none>
    Volumes:        <none>
```

Deployment Parameters

- **Progress Deadline Seconds:** Specifies the number of seconds we want to wait for our Deployment to progress before the system reports back that the Deployment has failed
- **Min Ready Seconds:** Specifies the minimum number of seconds for which a newly created Pod should be ready without any of its containers crashing, for it to be considered available
- **Revision History Limit:** Specifies the number of old ReplicaSets to retain to allow rollback. Default, 10 old ReplicaSets will be kept.
- **Paused:** Boolean field for pausing and resuming a Deployment. A Deployment is not paused by default when it is created.

Application Autoscaling

Autoscaling with HPA



Horizontal Pod Autoscaler

- HPA automatically scales the number of pods in a deployment based on observed CPU utilization, pod or object metrics
- HPA does not apply to objects that can’t be scaled
- Metric server monitoring needs to be deployed in the cluster
- Metric server collects metrics from cAdvisor

Questions

Persistent Storage

Volume Types

- **EmptyDir**
  - An emptyDir volume is an empty directory created in the pod
  - Containers can read and write files in this directory
  - The directory's contents are removed once the pod is deleted

- **hostPath**
  - hostPath volumes directly mount the node's filesystem into the pod
  - Often not used in production

Volume Types

- **NFS**
  - NFS volumes are network file system shares that are mounted into the pod
  - An nfs volume's contents might exist beyond the pod's lifetime
  - Kubernetes doesn't do anything in regards to the management of the nfs

- **configMap and secret**
  - useful for inserting configuration and secret data into your containers

Persistent Storage

- **PersistentVolumes (PV)** are volume plugins like Volumes
- PVs lifecycle is managed by Kubernetes
- PVs are cluster resources that exist independently of Pods
- PVs can be provisioned by :
  - Static provisioning by an administrator
  - Dynamically provisioned using Storage Classes
- **PersistentVolumeClaim (PVC)** is a request for storage by a user

Volume and Access Modes

- **VolumeModes of PersistentVolumes:**
  - Filesystem : It’s mounted into Pods into a directory
  - Block: Use a volume as a raw block device

- **Volume access modes are:**
  - ReadWriteOnce (RWO) – Mounted as read-write one
  - ReadOnlyMany (ROX) – Mounted read-only by many
  - ReadWriteMany (RWX) – Mounted as read-write by many

Yaml Definitions

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pvvol-1
spec:
  capacity:
    storage: 1Gi
  accessModes:
  - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /opt/sfw
    server: kubeadm-master
    readOnly: false
~
~
~
~
```

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs-claim
spec:
  accessModes:
  - ReadWriteMany
  resources:
    requests:
      storage: 500Mi
~
~
~
~
~
```

```
apiVersion: v1
kind: Pod
metadata:
  name: www
  labels:
    name: www
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: nfs-vol
        mountPath: /usr/share/nginx/html
  volumes:
    - name: nfs-vol
      persistentVolumeClaim:
        claimName: nfs-claim
~
~
~
~
```

Persistent Storage – Phase

- Available -- a free resource that is not yet bound to a claim
- Bound -- the volume is bound to a claim
- Released -- the claim has been deleted, but the resource is not yet reclaimed by the cluster
- Failed -- the volume has failed its automatic reclamation

Important Points – Persistent Storage

- Volume abstraction solves issues of sharing storage between containers
- Saving from losing all data in case container is deleted and recreated
- Pods access storage by using the claim as a volume
- Claims must exist in the same namespace as the Pod using the claim

Advanced Scheduling

kube-scheduler

- Kubernetes default Scheduler
- Scheduling is an optimization process
- The task of the Kubernetes Scheduler is to choose a placement
- **Step 1: Filtering**
  - Scheduler determines the set of feasible placements, the set of placements that meet a set of given constraints
- **Step 2: Scoring**
  - Scheduler determines the set of viable placements, the set of feasible placements with the highest score

Advanced Scheduling

- From Kubernetes 1.6 it offers four advanced scheduling features:
  - Node Selector
  - Node Affinity
  - Pod Affinity/Anti-Affinity
  - Taints and Tolerations

Node Selector

- The simplest recommended form of node selection constraint.
- nodeSelector is a field of PodSpec
- It specifies a map of key-value pairs
-

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
        ports:
        - containerPort: 80
      nodeSelector:
        disktype: ssd
~
~
~
~
```

Advanced Scheduling

- From Kubernetes 1.6 it offers four advanced scheduling features:
  - Node Selector
  - NodeAffinity
  - Pod Affinity/Anti-Affinity
  - Taints and Tolerations

Node Affinity

- Similar to nodeSelector
- Allows to constrain based on labels on the node
- Types of node affinity:
  - preferredDuringSchedulingIgnoredDuringExecution
  - requiredDuringSchedulingIgnoredDuringExecution

Node Affinity/Anti-Affinity

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
        ports:
        - containerPort: 80
      affinity:
        nodeAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: disktype
                operator: In
                values:
                - ssd
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.12
        ports:
        - containerPort: 80
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: disktype
                operator: NotIn
                values:
                - ssd
```

Taint & Tolerations

Adv Scheduling - Taint & Tolerations

- Node Affinity was a property of Pods that attracts them to a set of nodes (either as a preference or a hard requirement)
- Taints are the opposite -- they allow a node to repel a set of pods
- Tolerations are applied to pods, and allow (but do not require) the pods to schedule onto nodes with matching taints
- Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes
-

```
$
$ kubectl taint node aks-agentpool-40017546-vmss000001 disktype=magnetic:NoSchedule
node/aks-agentpool-40017546-vmss000001 tainted
```

```
apiVersion: v1
kind: Pod
metadata:
  name: tt-pod1
spec:
  containers:
   - name: nginx
     image: nginx
  tolerations:
  - key: "disktype"
    operator: "Equal"
    value: "magnetic"
    effect: "NoSchedule"
~
~
~
~
```

Taint Effect and Operator

- **Effect:**
  - **NoSchedule:** New pods that do not match the taint are not scheduled onto that node but the existing pods on the node remain
  - **PreferNoSchedule:** New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to and the existing pods on the node remain
  - **NoExecute:** New pods that do not match the taint cannot be scheduled onto that node and the existing pods on the node that donot have a matching toleration are removed
- **Operator:**
  - **Equal:** The key/value/effect all three parameters must match. This is the default
  - **Exists:**

```
apiVersion: v1
kind: Pod
metadata:
  name: tt-pod1
spec:
  containers:
   - name: nginx
     image: nginx
  tolerations:
  - key: "disktype"
    operator: "Equal"
    value: "magnetic"
    effect: "NoSchedule"
~
~
~
~
```

Taint Effect and Operator

- **Effect:**
  - **NoSchedule:** New pods that do not match the taint are not scheduled onto that node but the existing pods on the node remain
  - **PreferNoSchedule:** New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to and the existing pods on the node remain
  - **NoExecute:** New pods that do not match the taint cannot be scheduled onto that node and the existing pods on the node that donot have a matching toleration are removed
- **Operator:**
  - **Equal:** The key/value/effect all three parameters must match. This is the default
  - **Exists:**

```
apiVersion: v1
kind: Pod
metadata:
  name: tt-pod1
spec:
  containers:
   - name: nginx
     image: nginx
  tolerations:
  - key: "disktype"
    operator: "Equal"
    value: "magnetic"
    effect: "NoSchedule"
~
~
~
~
```

Taint Effect and Operator

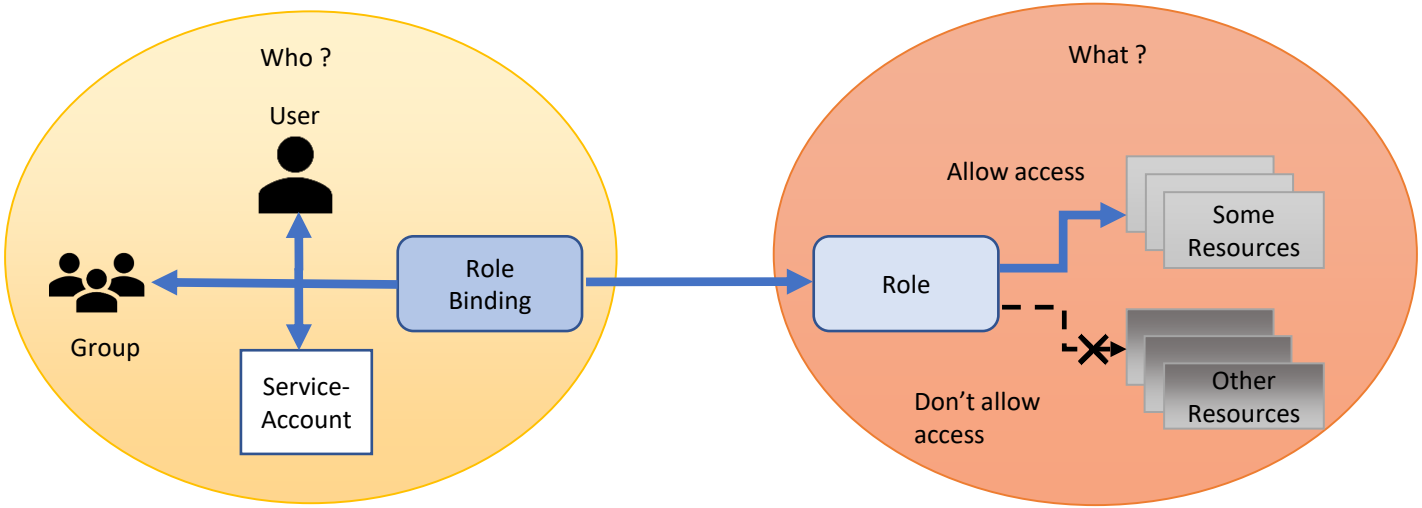
- **Effect:**
  - **NoSchedule:** New pods that do not match the taint are not scheduled onto that node but the existing pods on the node remain
  - **PreferNoSchedule:** New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to and the existing pods on the node remain
  - **NoExecute:** New pods that do not match the taint cannot be scheduled onto that node and the existing pods on the node that donot have a matching toleration are removed
- **Operator:**
  - **Equal:** The key/value/effect all three parameters must match. This is the default
  - **Exists:**

```
apiVersion: v1
kind: Pod
metadata:
  name: tt-pod1
spec:
  containers:
   - name: nginx
     image: nginx
  tolerations:
  - key: "disktype"
    operator: "Equal"
    value: "magnetic"
    effect: "NoSchedule"
~
~
~
~
```

Kubernetes Security

Role and RoleBinding

- Role is a set of rules that represent a set of purely additive permissions
- Role is used to grant access to resources within a single namespace
- RoleBinding binds the permissions defined in a role to a user or set of users



Cluster Role and RoleBinding

- **ClusterRole** : A role which has a cluster-wide scope. ClusterRole can also be used to grant access to:
  - define permissions on namespaced resources and be granted within individual namespace(s)
  - define permissions on namespaced resources and be granted across all namespaces
  - define permissions on cluster-scoped resources

- **ClusterRoleBinding** references a ClusterRole to grant the permissions to namespaced resources defined in the ClusterRole
- This allows administrators to define a set of common roles for the entire cluster, then reuse them within multiple namespaces

Network Security

- Network policies are namespaced resources, we create policy for each namespace
- By default, pods are non-isolated; they accept traffic from any source
- NetworkPolicy uses labels to select pods and define rules which specify what traffic is allowed to the selected pods
- **If no network policies apply to a pod, then all network connections to and from it are permitted**
- Once NetworkPolicy is attached to a pod, that pod will reject any connections that are not allowed by any NetworkPolicy
- **Pods are "isolated" if at least one network policy applies to them; if no policies apply, they are "non-isolated"**

Allow all Ingress traffic

- This manifest file is having empty the spec.ingress field
- Allow any traffic into all the Pod
- Target Pods with app=web label to police the network
- This NetworkPolicy will allow all traffic to pods of an application, selected using Pod Selectors
- Empty ingress rule ({}) allows traffic from all pods in the current namespace, as well as other namespaces. It corresponds to:

  - from:

    podSelector: {}

      namespaceSelector: {}

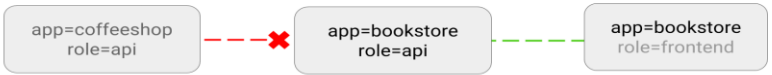
```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-ingress
spec:
  podSelector: {}
  ingress:
  - {}
  policyTypes:
  - Ingress
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - {}
  policyTypes:
  - Ingress
~
~
~
```

Allow to and from (same namespace)

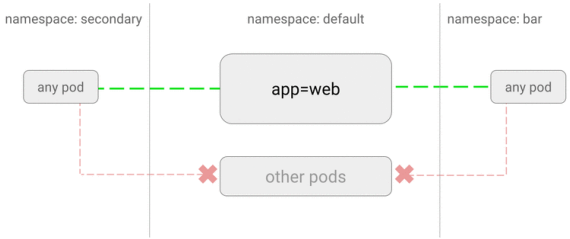
- Each network policy has a **podSelector** field, which selects a group of (zero or more) pods
- When a pod is selected by a network policy, the network policy is said to apply to it
- When the network policy is created, all the pods that it applies to are allowed to make or accept the connections listed in it
- A network policy is a list of allowed connections

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  policyTypes:
  - Ingress
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
~
~
```



Allow to and from (namespaceSelector)

```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: secondary
  name: web-allow-all-namespaces
spec:
  podSelector:
    matchLabels:
      app: web
  policyType:
  - Ingress
  ingress:
  - from:
    - namespaceSelector: {}
~
~
~
```




```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production
~
```

ALLOW traffic from specific pods in a namespace

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-ns-monitoring
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
  - Ingress
  ingress:
    - from:
      - namespaceSelector:     # chooses all pods in namespaces labelled with team=operations
          matchLabels:
            team: operations
        podSelector:           # chooses pods with type=monitoring
          matchLabels:
            type: monitoring
~
~
~
~
```

Egress and Ingress

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

- **Egress rules:**

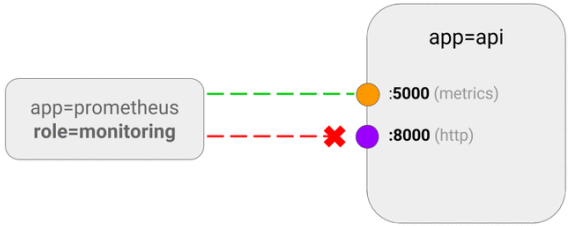
Allows connections from any pod in the "default" namespace with the label "role=db" to CIDR 10.0.0.0/24 on TCP port 5978

- **Ingress rules :**

Allows connections to all pods in the "default" namespace with the label "role=db" on TCP port 6379 from:

- Any pod in the "default" namespace with the label "role=frontend"
- Any pod in a namespace with the label "project=myproject"
- IP addresses in the ranges 172.17.0.0–172.17.0.255 and 172.17.2.0–172.17.255.255 (ie, all of 172.17.0.0/16 except 172.17.1.0/24)

ALLOW traffic only to a port of a Pod




```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: api-allow-5000
spec:
  podSelector:
    matchLabels:
      app: apiserver
  policyTypes:
  - Ingress
  ingress:
  - ports:
    - port: 5000
    from:
    - podSelector:
        matchLabels:
          role: monitoring
~
~
```

Security Context

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine-user-context
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      runAsUser: 405
~
~
~
~
~
```

```
apiVersion: v1
kind: Pod
metadata:
  name: alpine-nonroot
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      runAsNonRoot: true
~
~
~
~
~
~
```

```
apiVersion: v1
kind: Pod
metadata:
  name: privileged-pod
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      privileged: true
~
~
```

Security Context

```
apiVersion: v1
kind: Pod
metadata:
  name: kernelchange-pod
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      capabilities:
        add:
        - SYS_TIME
~
~
~
~
~
~
~
```

```
apiVersion: v1
kind: Pod
metadata:
  name: remove-capabilities
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      capabilities:
        drop:
        - CHOWN
~
~
~
~
~
~
```

```
apiVersion: v1
kind: Pod
metadata:
  name: readonly-pod
spec:
  containers:
  - name: main
    image: alpine
    command: ["/bin/sleep", "999999"]
    securityContext:
      readOnlyRootFilesystem: true
    volumeMounts:
    - name: my-volume
      mountPath: /volume
      readOnly: false
  volumes:
  - name: my-volume
    emptyDir:
~
~
~
~
~
```

Drawbacks & Solution

- The drawback of Docker and Kubernetes environment variables is that they are tied to the container or deployment
- If we wish to change them, we have to rebuild the container or modify the deployment
- Even worse, if we wish to use the variable with multiple containers or deployments, we have to duplicate the data!
- Kubernetes has solved this problem with Secrets (for confidential data) and ConfigMaps (for non-confidential data)

Env Variable - ConfigMap

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
  namespace: default
data:
  mydata: hello_world

~
~
~
~
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cm-pod
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    env:
      - name: cm
        valueFrom:
          configMapKeyRef:
            name: my-config
            key: mydata

~
~
```

Cluster Resource Limit

Resource Quota

- Defines resource consumption quota per namespace
- The quota system keeps track of resources created by the users and ensures that the hard limits are not exceeded
- Compute resources like CPU and Memory can be limited in a namespace
- Number of objects of a kind can be limited per namespace
- When a cluster is shared among several teams, it becomes necessary to limit the amount of resources that can be used by them

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
  namespace: quotas
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
~
~
~
~
~
~
~
~
~
~
~
~
~
```

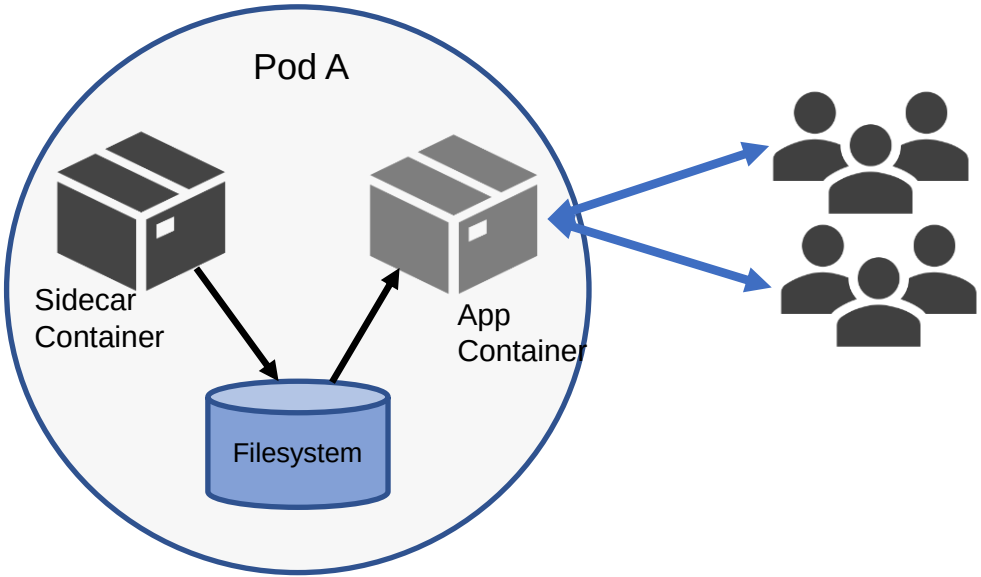
Limiting Resources

- Containers in a Pod can specify the resource requirements
- **Limits:** Specifies the max the container would go up to
- **Requests:** The minimum need for the container

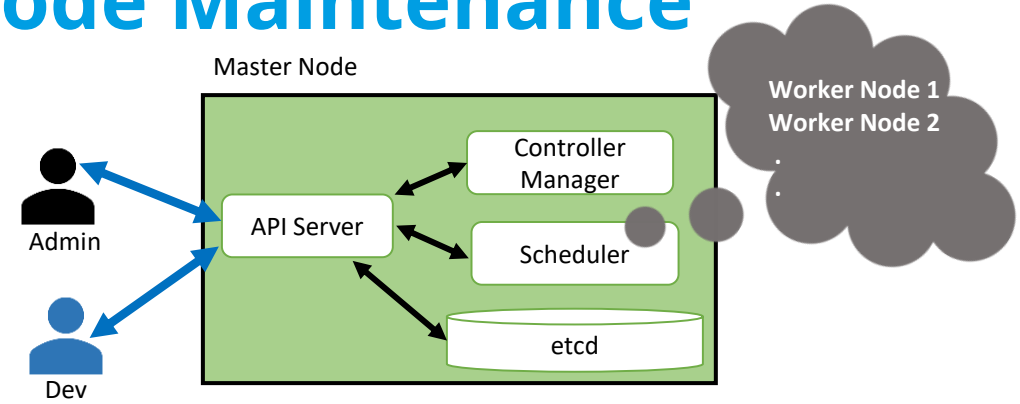
```
apiVersion: v1
kind: Pod
metadata:
  name: quota-pod
  namespace: quotas
spec:
  containers:
  - name: quota-container
    image: nginx
    resources:
      limits:
        memory: "800Mi"
        cpu: "1000m"
      requests:
        memory: "600Mi"
        cpu: "350m"
~
~
~
```

Multi Container - Sidecar Pattern

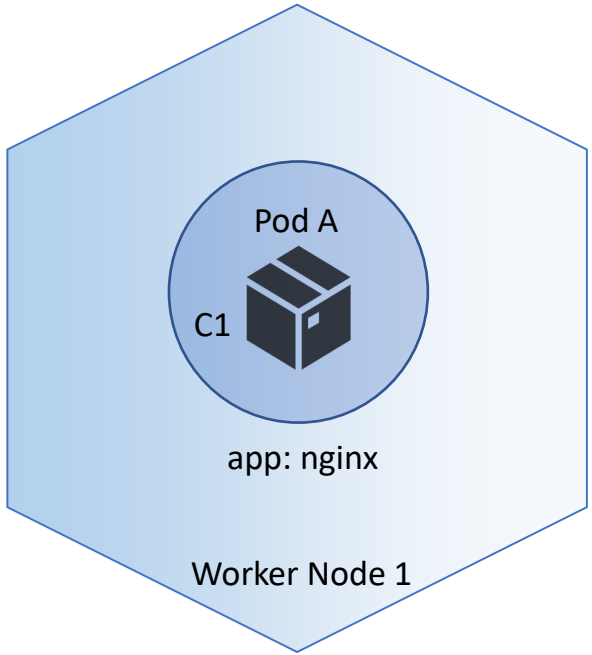
Multi Container - Sidecar Pattern



Node Maintenance



- Use **kubectl cordon** to mark the node as unschedulable
- Use **kubectl drain** to gracefully terminate all pods on the node while marking the node as unschedulable
- Use **kubectl uncordon** to mark the node schedulable again



K8s Control Plane Log Path

- **Daemon logs – Kubelet and Docker**
  - Every node @ /var/log/syslog
- **Kube-system Pod logs on Master**
  - Master node @ /var/log/pods (or)
  - Master node @ /var/lib/docker/containers find respective container logs
- **Kube-system and App Pod logs on Worker**
  - Every node @ /var/log/pods (or)
  - Every node @ /var/lib/docker/containers find respective container logs

kube-system Process Logs

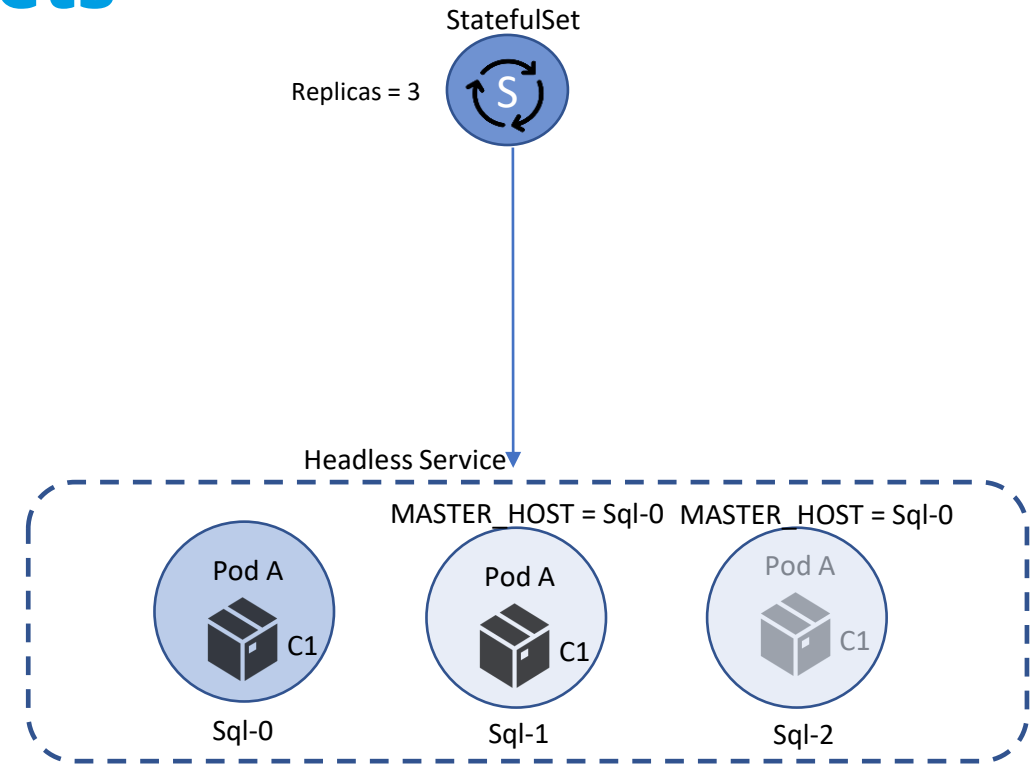
```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl logs kube-proxy-49gft -n kube-system
W0718 14:33:36.398781       1 server_others.go:559] Unknown proxy mode "", assuming iptables proxy
I0718 14:33:36.410772       1 node.go:136] Successfully retrieved node IP: 10.0.0.5
I0718 14:33:36.410937       1 server_others.go:186] Using iptables Proxier.
W0718 14:33:36.410965       1 server_others.go:436] detect-local-mode set to ClusterCIDR, but no cluster CIDR defined
I0718 14:33:36.410972       1 server_others.go:447] detect-local-mode: ClusterCIDR , defaulting to no-op detect-local
I0718 14:33:36.411420       1 server.go:583] Version: v1.18.6
I0718 14:33:36.411959       1 conntrack.go:100] Set sysctl 'net/netfilter/nf_conntrack_max' to 131072
I0718 14:33:36.411995       1 conntrack.go:52] Setting nf_conntrack_max to 131072
I0718 14:33:36.412168       1 conntrack.go:100] Set sysctl 'net/netfilter/nf_conntrack_tcp_timeout_established' to 86400
I0718 14:33:36.412247       1 conntrack.go:100] Set sysctl 'net/netfilter/nf_conntrack_tcp_timeout_close_wait' to 3600
I0718 14:33:36.414454       1 config.go:315] Starting service config controller
I0718 14:33:36.414484       1 shared_informer.go:223] Waiting for caches to sync for service config
I0718 14:33:36.414883       1 config.go:133] Starting endpoints config controller
```

```
root@kubeadm-master:/home/ubuntu#
root@kubeadm-master:/home/ubuntu# kubectl logs kube-scheduler-kubeadm-master -n kube-system
I0718 14:27:03.854038       1 registry.go:150] Registering EvenPodsSpread predicate and priority function
I0718 14:27:03.854115       1 registry.go:150] Registering EvenPodsSpread predicate and priority function
I0718 14:27:04.946783       1 serving.go:313] Generated self-signed cert in-memory
W0718 14:27:09.252268       1 authentication.go:349] Unable to get configmap/extension-apiserver-authentication in kube-system.  Usually fixed by 'kubectl create rolebinding
 -n kube-system ROLEBINDING_NAME --role=extension-apiserver-authentication-reader --serviceaccount=YOUR_NS:YOUR_SA'
W0718 14:27:09.252316       1 authentication.go:297] Error looking up in-cluster authentication configuration: configmaps "extension-apiserver-authentication" is forbidden:
User "system:kube-scheduler" cannot get resource "configmaps" in API group "" in the namespace "kube-system"
W0718 14:27:09.252328       1 authentication.go:298] Continuing without authentication configuration. This may treat all requests as anonymous.
W0718 14:27:09.252336       1 authentication.go:299] To require authentication configuration lookup to succeed, set --authentication-tolerate-lookup-failure=false
I0718 14:27:09.276928       1 registry.go:150] Registering EvenPodsSpread predicate and priority function
I0718 14:27:09.276957       1 registry.go:150] Registering EvenPodsSpread predicate and priority function
I0718 14:27:09.281087       1 secure_serving.go:178] Serving securely on 127.0.0.1:10259
I0718 14:27:09.281359       1 configmap_cafile_content.go:202] Starting client-ca::kube-system::extension-apiserver-authentication::client-ca-file
I0718 14:27:09.281377       1 shared_informer.go:223] Waiting for caches to sync for client-ca::kube-system::extension-apiserver-authentication::client-ca-file
I0718 14:27:09.281542       1 tlsconfig.go:240] Starting DynamicServingCertificateController
```

Application Logs

```
root@kubeadm-master:/home/ubuntu/Kubernetes-basics#
root@kubeadm-master:/home/ubuntu/Kubernetes-basics# kubectl logs demo-pod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
```

StatefulSets



StatefulSets

- Used for Stateful application and Distributed systems
- For a StatefulSet with N replicas, Pods are created sequentially, in order from {0..N-1}
- Pods in a StatefulSet have a unique ordinal index and a stable network identity
- The RollingUpdate strategy updates all Pods in a StatefulSet, in reverse ordinal order
- The controller deletes one Pod at a time, in reverse order with respect to its ordinal index

Headless Service

- Don't need load-balancing and a single Service IP
- Explicitly specifying "None" for the cluster IP
- Allows us to interact directly with the Pods instead of a proxy
- A headless service is a service with a service IP but instead of load-balancing it will return the IPs of our associated Pods

```
kind: Service
apiVersion: v1
metadata:
  name: elasticsearch
  namespace: kube-logging
  labels:
    app: elasticsearch
spec:
  selector:
    app: elasticsearch
  clusterIP: None
  ports:
    - port: 9200
      name: rest
    - port: 9300
      name: inter-node
~
~
~
```

DaemonSet

- A daemonset ensures that an instance of a specific pod is running on all nodes in a cluster
- It creates pods on every cluster node and these pods are garbage collected when nodes are removed from the cluster
- We can use daemonsets for running daemons and other tools that need to run on all nodes of a cluster
- Cluster bootstrapping is another use case. The DaemonSet controller can make Pods even when the scheduler has not been started
- We can perform rolling update on DaemonSet

Node specific Daemons

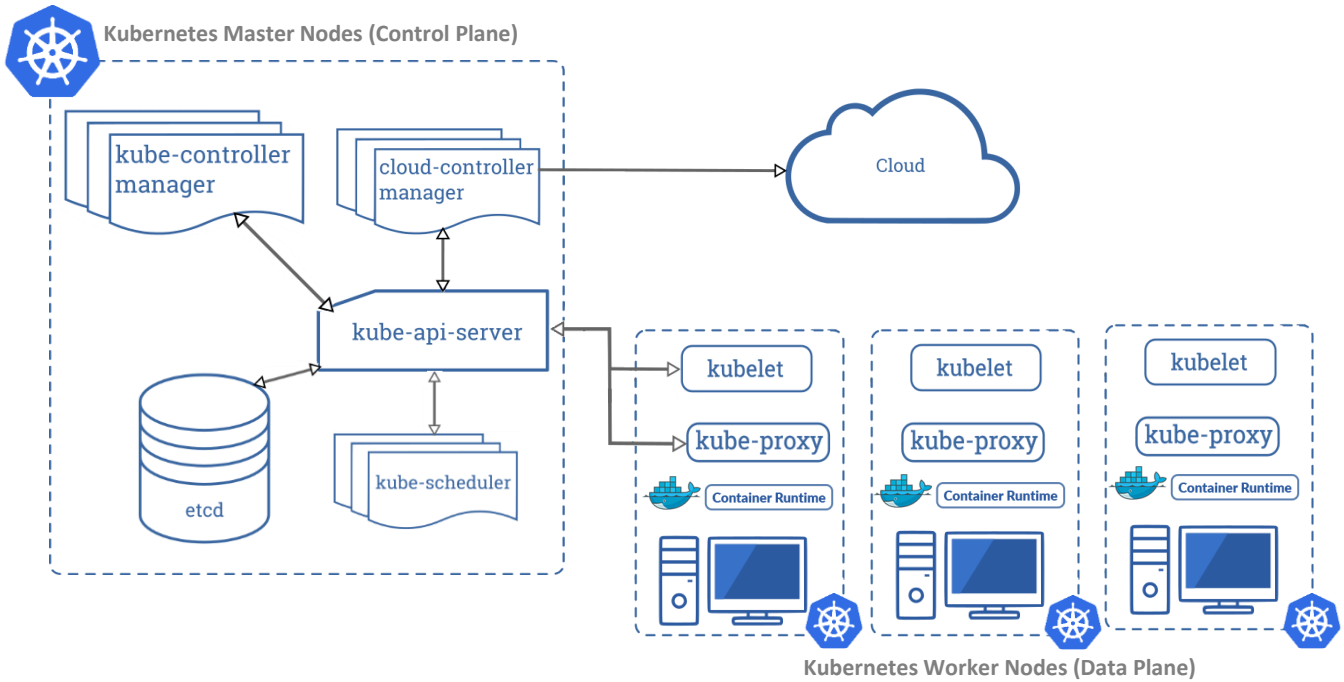
- If we plan to have different logging, monitoring, or storage solutions on different nodes of the cluster
- We want to deploy the daemons only to a specific set of nodes instead of all Node selector can be used to specify a subset of nodes for the daemon set
- DaemonSet controller will create Pods on nodes which match that node affinity

Managed Kubernetes on Cloud

Kubernetes Adoption in Cloud

- AWS - EKS
- Microsoft Azure - AKS
- Google Cloud Platform - GKE
- Oracle Cloud - OKE
- Digital Ocean - DOKS

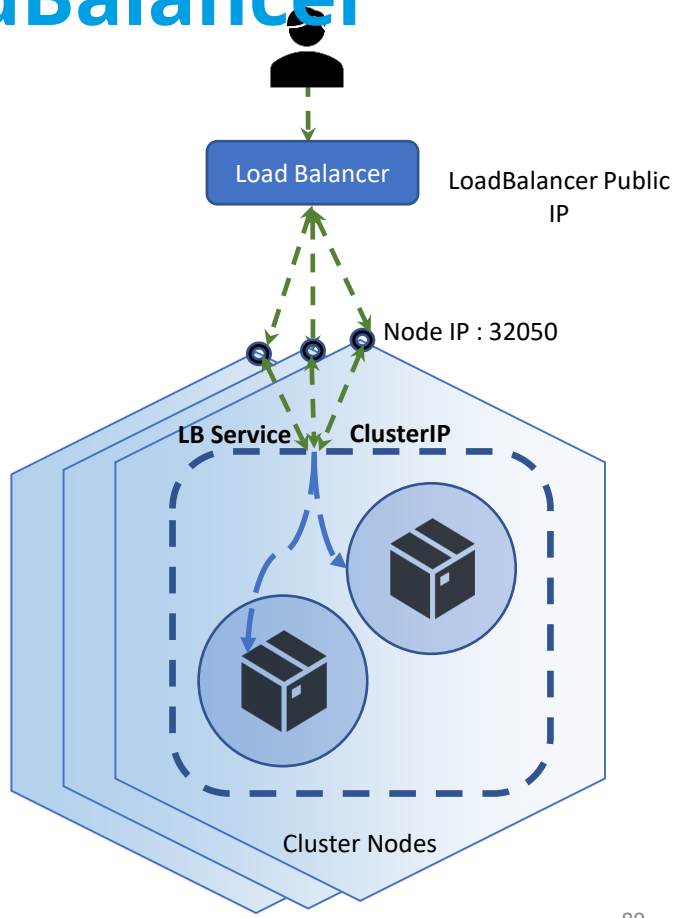
Kubernetes Architecture



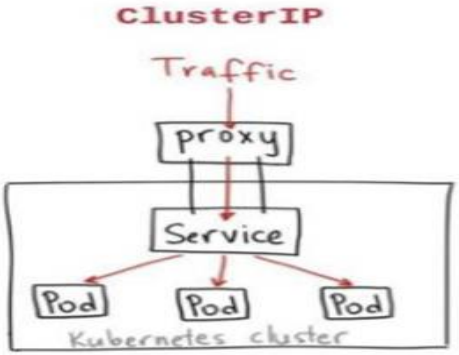
Advanced Networking

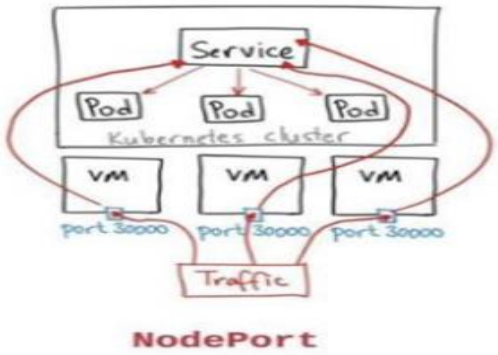
Kubernetes Service - LoadBalancer

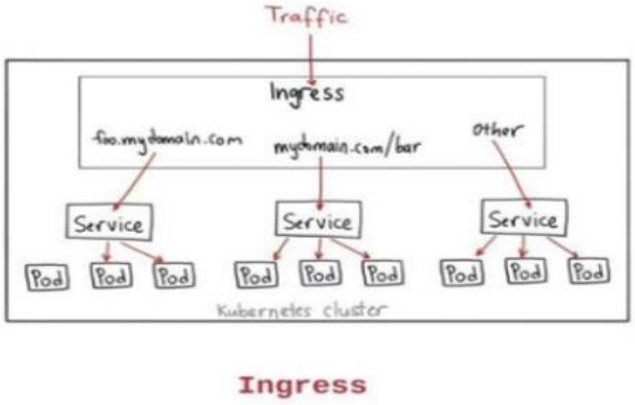
- Exposes the Pod to outside the cluster
- Assigns a Private IP Address in Cluster IP range
- Assigns a port on the worker nodes to expose it outside cluster
- Create a Load Balancer in Cloud

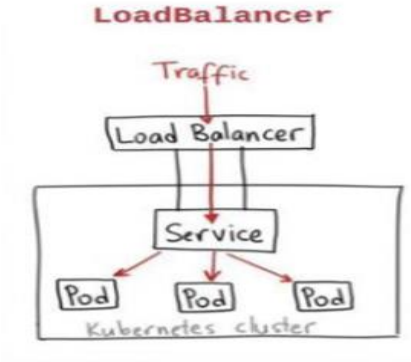


K8s Service Types

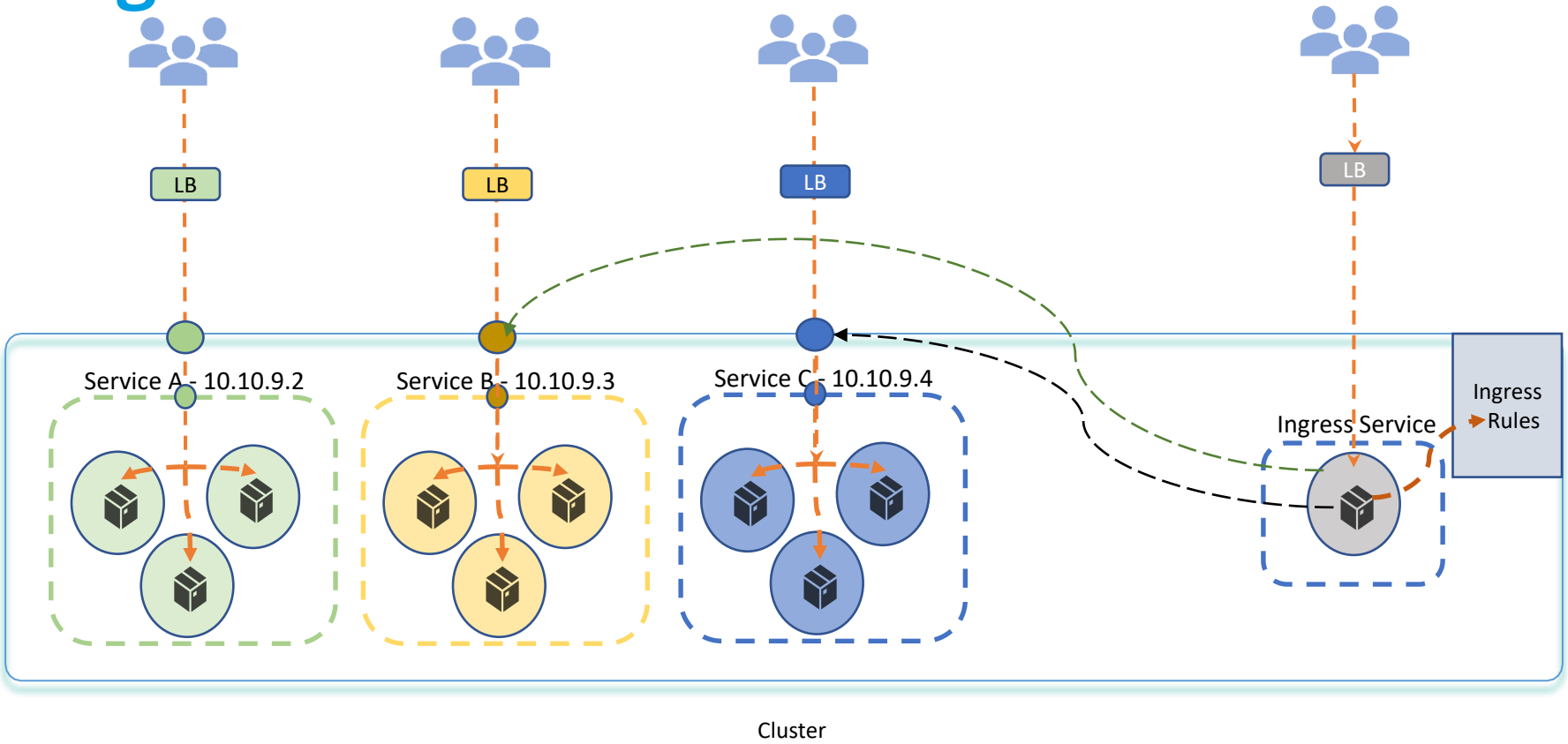








Ingress Controller



Ingress Controller

- Intelligent Load Balancer
- Layer 7 Load Balancer
- Nginx or Nginx +
- It doesn’t run as part of kube-controller-manager
- With Ingress, users don't connect directly to a Service
- Users reach the Ingress endpoint, and, from there, the request is forwarded to the respective Service

Ingress

- An API object that manages external access to the services in a cluster, typically HTTP
- Ingress may provide load balancing, SSL termination and name-based virtual hosting
- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the Ingress resource
- For it to work we need to deploy an Ingress controller such as ingress-nginx

Helm
&
Helm Charts

Helm Terminology

- A Chart is a Helm package. It contains all of the resource definitions necessary to run an application, tool, or service inside of a Kubernetes cluster
- A Repository is the place where charts can be collected and shared
- A Release is an instance of a chart running in a Kubernetes cluster

Helm Charts

- Package manager for Kubernetes
- Helm is the best way to find, share, and use software built for Kubernetes
- Helm helps us manage Kubernetes applications — Helm Charts
- Helm Chart is collection of files that describe a related set of K8s resources
- Helps us define, install, and upgrade even the most complex Kubernetes application
- Charts are easy to create, version control, share, and publish

Helm Commands

- helm search
- helm repo add
- helm install
- helm list
- helm status
- helm show values
- helm install
- helm upgrade
- helm get values
- helm rollback
- helm uninstall

Yaml Definitions

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 8Gi
  storageClassName: slow
  selector:
    matchLabels:
      release: "stable"
    matchExpressions:
      - {key: environment, operator: In, values: [dev]}
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: mypd
  volumes:
    - name: mypd
      persistentVolumeClaim:
        claimName: myclaim
```

Storage Class

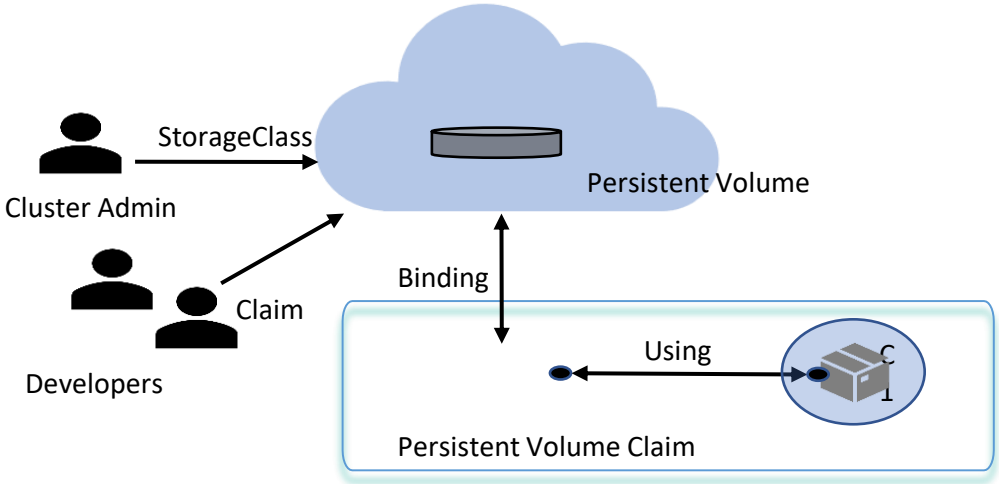
- A StorageClass provides a way for administrators to describe the "classes" of storage they offer
- Different classes might map to quality-of-service levels, or to backup policies
- Each StorageClass

```
mamta@Azure:~/Kubernetes$ kubectl describe sc managed-premium
Name:            managed-premium
IsDefaultClass:  No
Annotations:     kubectl.kubernetes.io/last-applied-configuration={"allowVolumeExpansion":true,"apiV
ersion":"storage.k8s.io/v1beta1","kind":"StorageClass","metadata":{"annotations":{},"labels":{"kuber
netes.io/cluster-service":"true"},"name":"managed-premium"},"parameters":{"cachingmode":"ReadOnly","
kind":"Managed","storageaccounttype":"Premium_LRS"},"provisioner":"kubernetes.io/azure-disk"}

Provisioner:           kubernetes.io/azure-disk
Parameters:            cachingmode=ReadOnly,kind=Managed,storageaccounttype=Premium_LRS
AllowVolumeExpansion:  True
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     Immediate
Events:                <none>
mamta@Azure:~/Kubernetes$
```

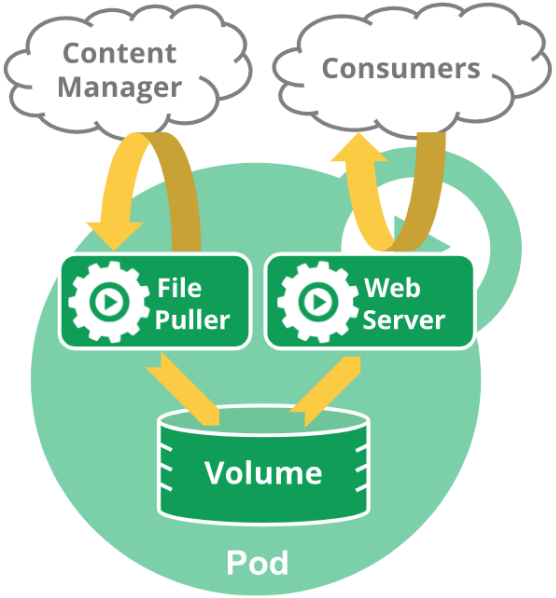
Dynamic Volume Provisioning

- Dynamic volume provisioning allows storage volumes to be created on-demand
- Dynamic volume provisioning is based on the API object StorageClass
- To enable dynamic provisioning, a cluster administrator pre-creates one or more StorageClass objects for users
- StorageClass objects define which provisioner should be used and what parameters should be passed to that provisioner when dynamic provisioning is invoked

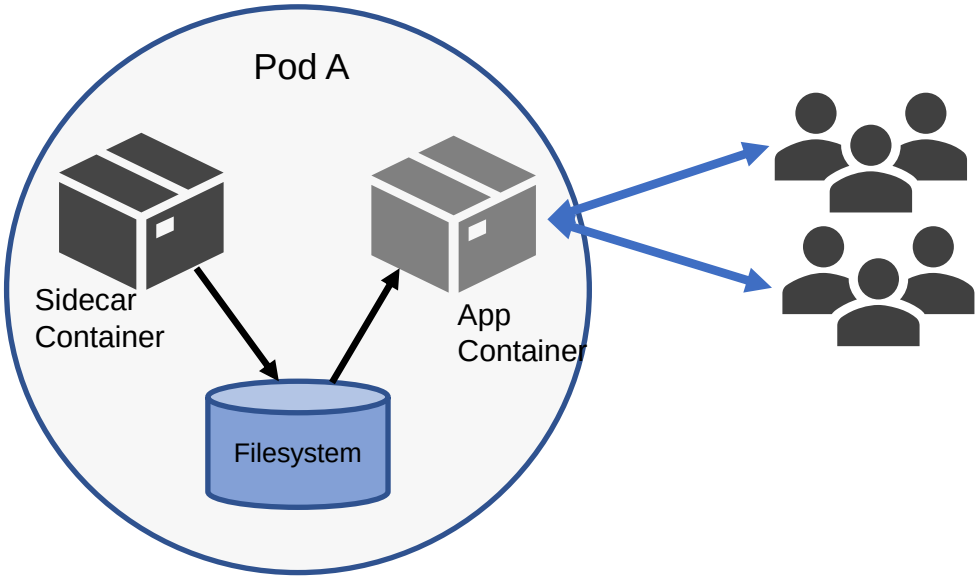


Multi-Containers Pod

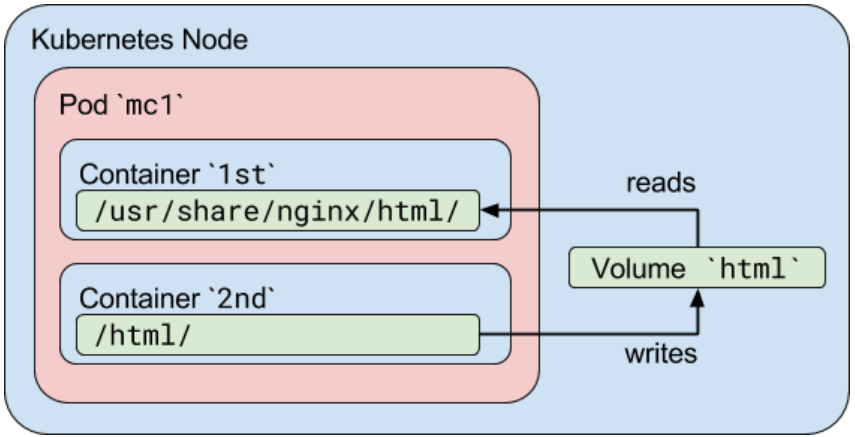
- Pods are designed to support multiple cooperating processes (as containers) that form a cohesive unit of service
- The containers in a Pod are automatically co-located and co-scheduled on the same physical or virtual machine in the cluster
- The containers can share resources and dependencies, communicate with one another, and coordinate when and how they are terminated
- These containers can efficiently communicate, ensuring data locality. Also, Pods enable us to manage several tightly coupled



Multi Container - Sidecar Pattern



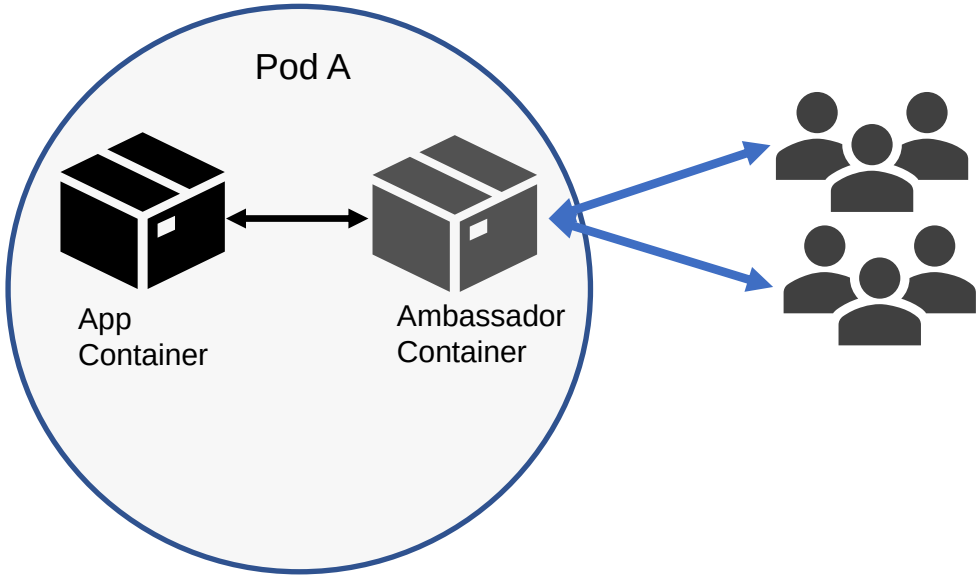
Multi Container - Sidecar Pattern




```
apiVersion: v1
kind: Pod
metadata:
  name: mc1
spec:
  volumes:
  - name: html
    emptyDir: {}
  containers:
  - name: 1st
    image: nginx
    volumeMounts:
    - name: html
      mountPath: /usr/share/nginx/html
  - name: 2nd
    image: debian
    volumeMounts:
    - name: html
      mountPath: /html
    command: ["/bin/sh", "-c"]
    args:
      - while true; do
          date >> /html/index.html;
          sleep 1;
        done
~
```

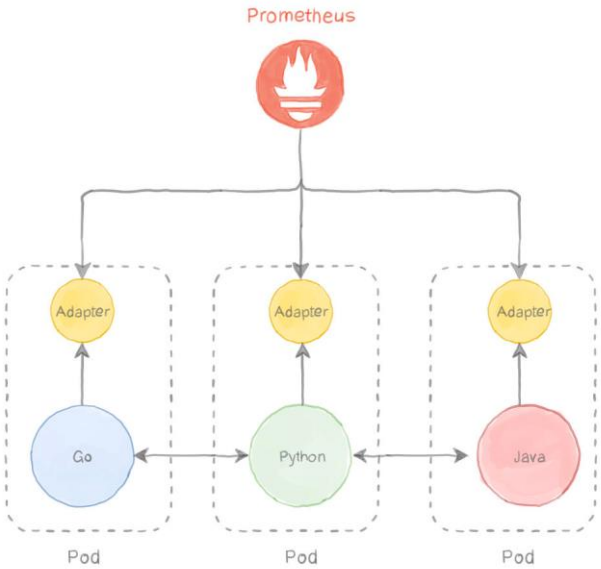
Multi Container - Ambassador Pattern

- An Ambassador container is in charge of proxying connections from the application container to other services
- Ambassador container acts as a client proxy, it proxies a local connection to the world



Multi Container - Adapter Pattern

- Adapter containers standardize and normalize output
- Using the adapter pattern means keeping communication between containers consistent
- Analyzing logs from different sources can be a pain if you don't have a standard format
- When you have a container that works as an adapter, it will receive raw logs
- It will standardize and store data in a centralized place
- Widely used for Logging and Monitoring



Jobs
and
Cron Jobs

Cron Job

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hw
            image: busybox
            args:
            - /bin/sh
            - -c
            - echo Hello World!
          restartPolicy: OnFailure
~
~
```

```
$ kubectl get cronjobs
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     1        7s              12s
```

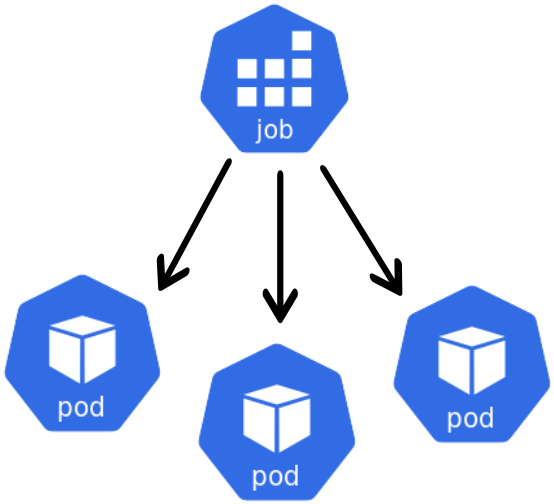
```
# ┌──────────── minute (0 - 59)
# │ ┌──────────── hour (0 - 23)
# │ │ ┌──────────── day of the month (1 - 31)
# │ │ │ ┌──────────── month (1 - 12)
# │ │ │ │ ┌──────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                    7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

Cron Job - Params

- **concurrencyPolicy:** It defines how will the concurrent executions of a jobs be treated that is created by this cron job
  - **Allow** (default): Concurrently running jobs are allowed
  - **Forbid**: concurrent runs are not allowed; if it is time for a new job run and the previous job run hasn't finished yet, the cron job skips the new job run
  - **Replace**: If it is time for a new job run and the previous job run hasn't finished yet, the cron job replaces the currently running job run with a new job run
- **Jobs History Limits:** These fields defines how many completed and failed jobs should be kept in the history

Jobs

- A Job creates one or more Pods and ensures that a specified number of them successfully terminate
- As pods successfully complete, the Job tracks the successful completions
- When a specified number of successful completions is reached, the task (ie, Job) is complete
- Deleting a Job will clean up the Pods it created
- The Job object will start a new Pod if the first Pod fails or is deleted



Non-Parallel Jobs - Task Types

- Non-parallel Jobs:
  - Normally, only one Pod is started, unless the Pod fails
  - The Job is complete as soon as its Pod terminates successfully
- **.spec.completions** and **.spec.parallelism** both are unset, both are defaulted to 1

```
apiVersion: batch/v1
kind: Job
metadata:
  name: countdown
spec:
  template:
    metadata:
      name: countdown
    spec:
      containers:
      - name: counter
        image: centos:7
        command:
         - "bin/bash"
         - "-c"
         - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
      restartPolicy: Never
~
~
```

Parallel Job - Batch Processing

- Each Job performs some substantial computation, such as rendering a frame of a movie, or processing a range of rows in a database
- If we were rendering a movie, we would set $ITEM to the frame number
- If we were processing rows from a database table, we would set $ITEM to represent the range of database rows to process
- Create Job template and use that to create Jobs

```
apiVersion: batch/v1
kind: Job
metadata:
  name: process-item-$ITEM
  labels:
    jobgroup: jobexample
spec:
  template:
    metadata:
      name: jobexample
      labels:
        jobgroup: jobexample
    spec:
      containers:
      - name: c
        image: busybox
        command: ["sh", "-c", "echo Processing item $ITEM && sleep 5"]
      restartPolicy: Never
~
```

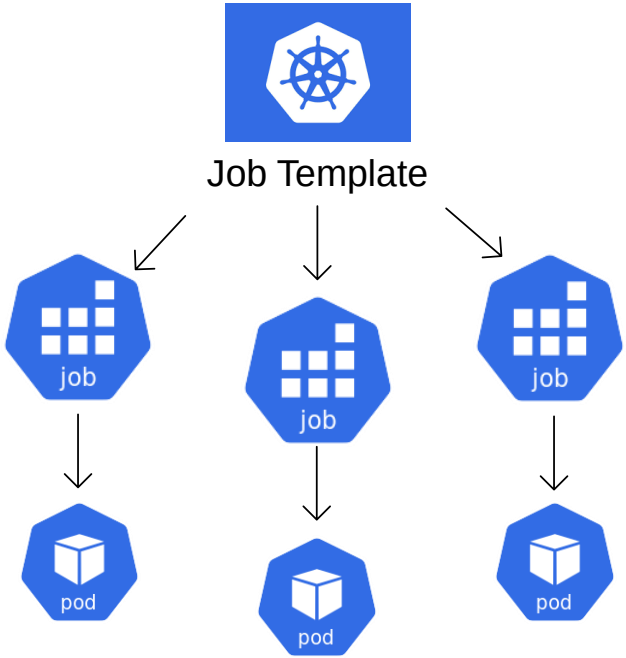
Parallel Job – Batch Processing

```
root@kubeadm-master:/home/ubuntu/new_files#
root@kubeadm-master:/home/ubuntu/new_files# mkdir ./jobs
root@kubeadm-master:/home/ubuntu/new_files# for i in apple banana cherry
> do
>   cat job-tmpl.yaml | sed "s/\$ITEM/$i/" > ./jobs/job-$i.yaml
> done
```

```
root@kubeadm-master:/home/ubuntu/new_files# cd jobs
root@kubeadm-master:/home/ubuntu/new_files/jobs# ls
job-apple.yaml  job-banana.yaml  job-cherry.yaml
```

```
root@kubeadm-master:/home/ubuntu/new_files# kubectl create -f ./jobs
job.batch/process-item-apple created
job.batch/process-item-banana created
job.batch/process-item-cherry created
```

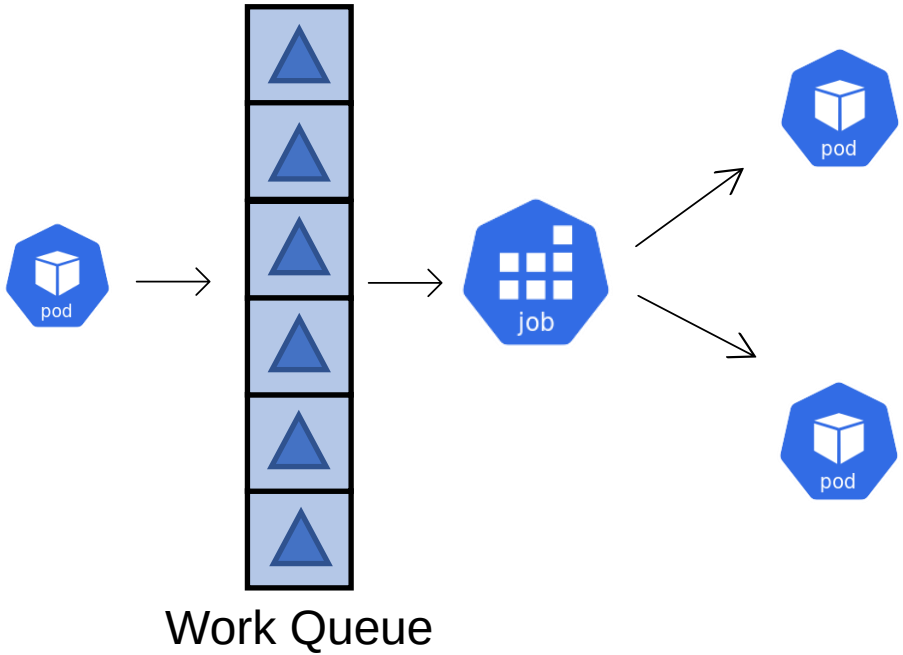
```
root@kubeadm-master:/home/ubuntu/new_files# kubectl get pods -l jobgroup=jobexample
NAME                         READY   STATUS      RESTARTS   AGE
process-item-apple-hntp7     0/1     Completed   0          27s
process-item-banana-sbgfg    0/1     Completed   0          27s
process-item-cherry-b9ch5    0/1     Completed   0          27s
```



Parallel Jobs - Task Types

- **Parallel Jobs with a work queue**:
- Pods must coordinate amongst themselves or an external service to determine what each should work on
- Each Pod is independently capable of determining whether or not all its peers are done, and thus that the entire Job is done
- When any Pod from the Job terminates with success, no new Pods are created
- Each pod is created, it picks up one unit of work from a task queue, completes it, deletes it from the queue, and exits

Parallel Job – Work Queue



Parallel Job – Work Queue

- Create a queue and fill it with messages. Each message represents one task to be done
- Start a Job that works on tasks from the queue
- The Job starts several pods
- Each pod takes one task from the message queue, processes it, and repeats until the end of the queue is reached.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-wq-1
spec:
  completions: 8
  parallelism: 2
  template:
    metadata:
      name: job-wq-1
    spec:
      containers:
      - name: c
        image: mamtaj/job-wq-1
        env:
        - name: BROKER_URL
          value: amqp://guest:guest@rabbitmq-service:5672
        - name: QUEUE
          value: job1
      restartPolicy: OnFailure
~
~
~
~
~
```