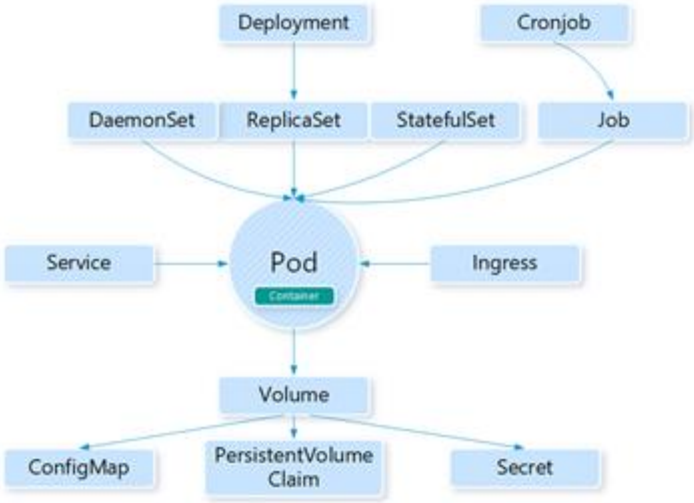
Kubernetes Essential

Agenda

- ➢ Assignment Review & Guides
- ➢ Kubernetes concepts
- ➢ Working with Pod
- ➢ Lab: Deployment Rolling Update and Rollback

API Overview

- Everything within Kubernetes is as **an API Object**.
- Referenced within an object as the **apiVersion** and **kind**.



```yaml
apiVersion: v1
kind: Pod
metadata:
 creationTimestamp: null
 labels:
   run: nginx
 name: nginx
spec:
 containers:
 - image: nginx:alpine
   name: nginx
   resources: {}
status: {}
```

Object model

- Objects represent the desired state of the object within the cluster.

- All objects **required**:
  - ○ apiVersion: version of the Kubernetes API to create the object
  - ○ kind: kind of object to create
  - ○ metadata: data that helps uniquely identify the object, including a name string, UID, and optional namespace.
  - ○ spec: desired state of the object

```
apiVersion: v1
kind: Pod
metadata:
 creationTimestamp: null
 labels:
   run: nginx
 name: nginx
spec:
 containers:
 - image: nginx:alpine
   name: nginx
   resources: {}
 dnsPolicy: ClusterFirst
 restartPolicy: Always
status: {}
```

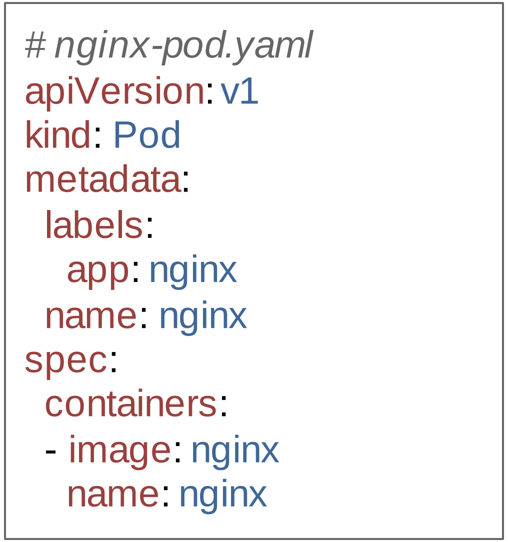
Object expression with YAML

- Files or other representations of Kubernetes Objects are generally represented in YAML.

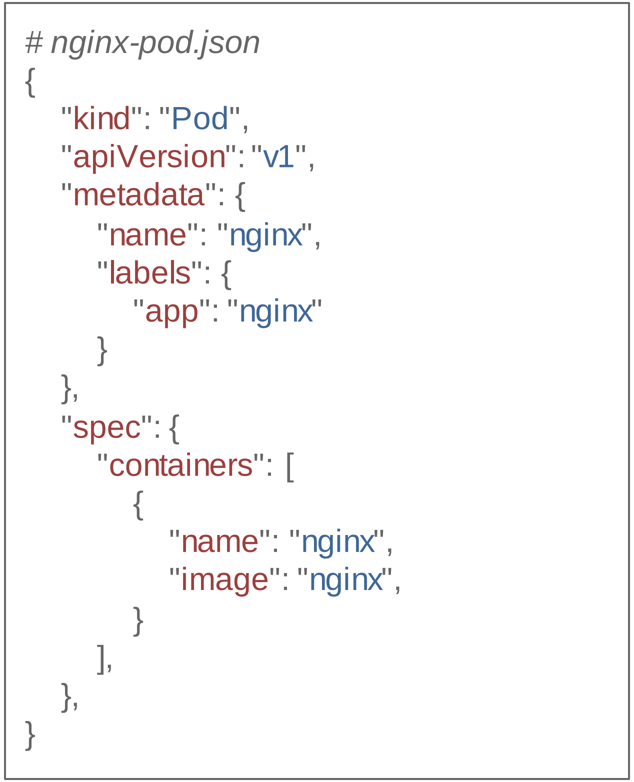
- Three basic data types:
  - List
  - Map
  - String, number, boolean, etc

```
# nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```

YAML and JSON



```
# nginx-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  containers:
  - image: nginx
    name: nginx
```



```
# nginx-pod.json
{
    "kind": "Pod",
    "apiVersion": "v1",
    "metadata": {
        "name": "nginx",
        "labels": {
            "app": "nginx"
        }
    },
    "spec": {
        "containers": [
            {
                "name": "nginx",
                "image": "nginx",
            }
        ],
    },
}
```

CORE CONCEPTS

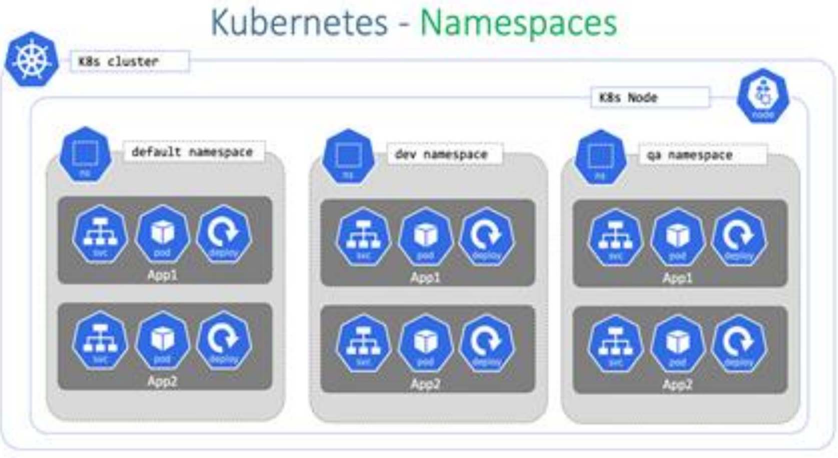
Namespaces

namespaces is used to isolate groups of resources within a single cluster:

- Names of resources need to be unique within a namespace, but not across namespaces.
- There are namespaced objects and cluster-wide objects.

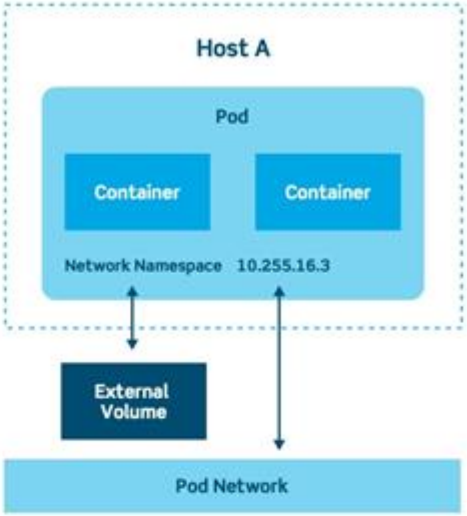
```
[user@mate ~]$ kubectl create ns my-namespace
namespace/my-namespace created
```

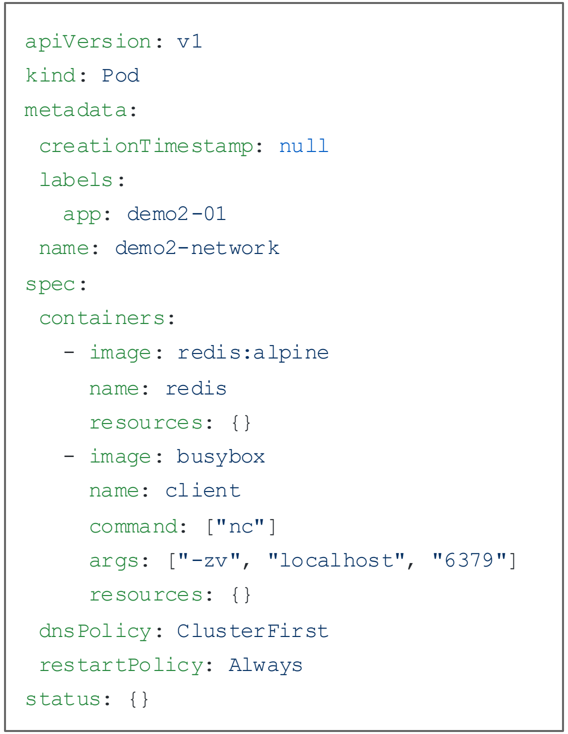
```
[user@mate ~]$ kubectl get ns
NAME                 STATUS   AGE
default              Active   71m
kube-node-lease      Active   71m
kube-public          Active   71m
kube-system          Active   71m
local-path-storage   Active   71m
```



Pods

- Smallest unit of work of Kubernetes.
- Pods are one or MORE containers that share volumes, a network namespace, and are a part of a **single context**.





```yaml
apiVersion: v1
kind: Pod
metadata:
 creationTimestamp: null
 labels:
   app: demo2-01
 name: demo2-network
spec:
 containers:
   - image: redis:alpine
     name: redis
     resources: {}
   - image: busybox
     name: client
     command: ["nc"]
     args: ["-zv", "localhost", "6379"]
     resources: {}
 dnsPolicy: ClusterFirst
 restartPolicy: Always
status: {}
```

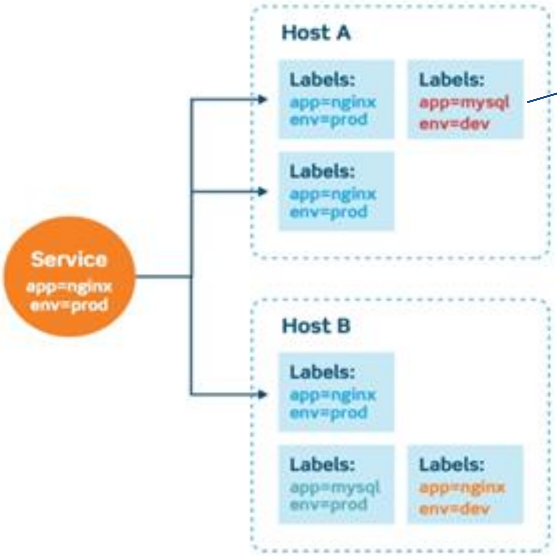
Pods

```
user@mate:~$ kubectl run new-nginx --image nginx --namespace my-ns
pod/new-nginx created
```

```
user@mate:~$ kubectl get pod --namespace my-ns
NAME         READY    STATUS     RESTARTS    AGE
new-nginx    1/1      Running    0           111s
```

Labels

- *Labels* are key-value pairs that are used to identify, describe and group together related sets of objects or resources.

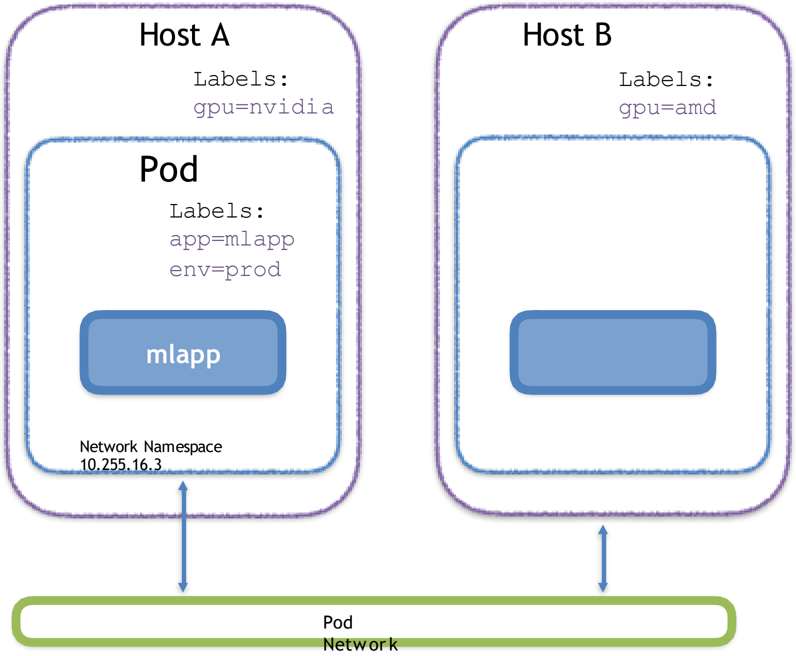


```
apiVersion: v1
kind: Pod
metadata:
 creationTimestamp: null
 labels:
   app: mysql
   env: dev
 name: demo2-network
spec:
 containers:
   - image: mysql
     name: mysql
     resources: {}
   - image: busybox
     name: client
     command: ["nc"]
     args: ["-zv", "localhost", "3306"]
     resources: {}
 dnsPolicy: ClusterFirst
 restartPolicy: Always
status: {}
```

Selectors

Selectors use *labels* to filter or select objects, and are used throughout Kubernetes.

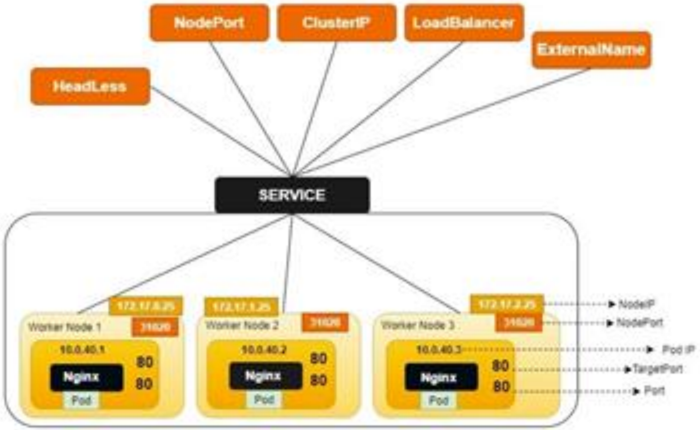
```
apiVersion: v1
kind: Pod
metadata:
  name:mlapp
  labels:
    app: mlapp
    env: prod
spec:
  nodeSelector:
  - gpu: nvidia
  containers:
  - name: nginx
    image: tensorflow/tensorflow
```



Service

Service is a method for exposing a network application that is running as one or more Pods in the cluster:

- unique static cluster-wide IP
- static namespaced DNS name



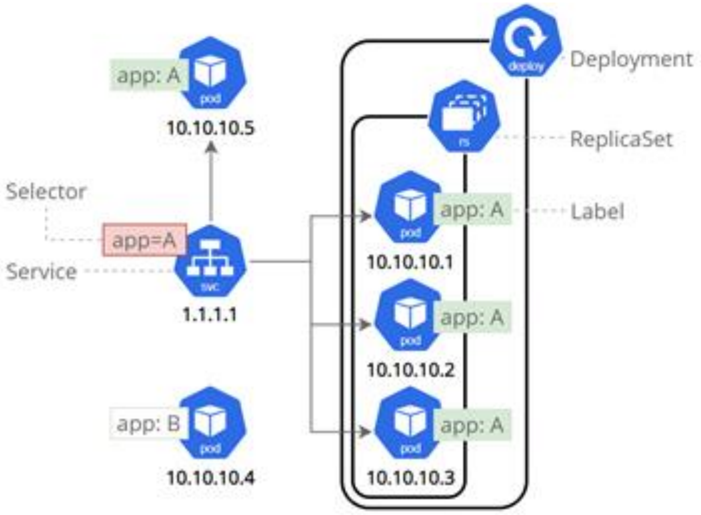
There are 4 types:

- ClusterIP
- NodePort
- LoadBalancer
- ExternalName

<service name>.<namespace>.svc.cluster.local

ClusterIP Service

ClusterIP service exposes the Service on a cluster-internal IP, uses labels and selectors to associate with pods. This Service only reachable from within the cluster.



ClusterIP Service

```
# Create a ClusterIP service in my-ns namespace with port 8080 map to port 80
thainm5hls@xps:~$ kubectl create service clusterip --namespace my-ns new-cs --
tcp=8080:80
service/new-cs created

# Edit created service
thainm5hls@xps:~$ kubectl edit -n my-ns svc new-cs
```

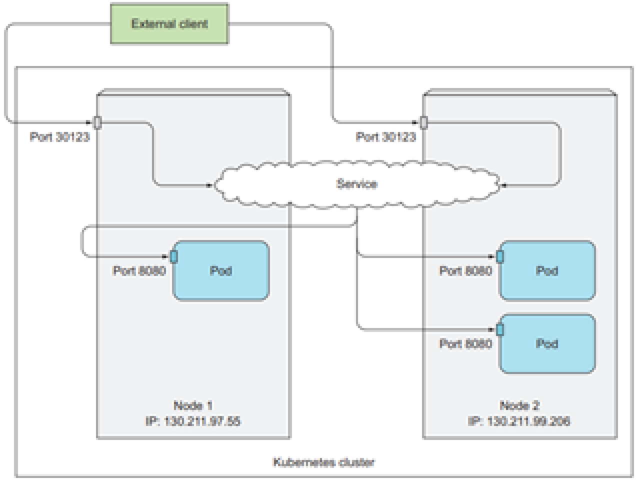
```
# Get list of services in my-ns namespace
thainm5hls@xps:~$ kubectl get svc -n my-ns
NAME      TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
new-cs    ClusterIP   10.96.163.21   <none>        8080/TCP   8m17s
```

```
# Get detailed information of service new-cs in namespace my-ns
thainm5hls@xps:~$ kubectl describe svc -n my-ns new-cs
```

NodePort Service

- *NodePort service* makes Kubernetes reserve a port on all its nodes (the same port number) and forward incoming connections to the pods that are part of the service.

- Similar to a ClusterIP service, but a NodePort service can be accessed not only through the service's internal cluster IP, but also through any node's IP and the reserved node port.



NodePort Service

```
# Create NodePort service nodeport-svc in namespace my-ns with service port 8080 map to backend port 80
thainm5hls@xps:~$ kubectl create service nodeport --namespace my-ns nodeport-svc --tcp=8080:80
service/nodeport-svc created
```

```
# List all services in namespace my-ns
thainm5hls@xps:~$ kubectl get svc -n my-ns
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
new-cs         ClusterIP   10.96.163.21    <none>        8080/TCP         23m
nodeport-svc   NodePort    10.96.197.227   <none>        8080:30175/TCP   9s
```

```
# Get detailed information of created NodePort service
thainm5hls@xps:~$ kubectl describe svc -n my-ns nodeport-svc
```

LOAD BALANCER SERVICE

- Kubernetes clusters running on cloud providers usually support the automatic provision of a load balancer from the cloud infrastructure.
- The load balancer will have unique publicly accessible IP address and will redirect all connections to service.
- If Kubernetes is running in an environment that doesn't support LoadBalancer services, the load balancer will not be provisioned, but the service will still behave like a NodePort service.

WORKLOADS

Workloads

- Workloads within Kubernetes are higher level objects that manage Pods or other higher level objects.

- In **ALL CASES** a Pod Template is included, and acts the base tier of management.

- Types of workload:
  - ReplicaSet
  - Deployment
  - DaemonSet
  - StatefulSet
  - Job
  - Cronjob

Pod template

- *Pod templates* are Pod specs with limited metadata.
- Workloads use *Pod templates* to make actual pods.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    name: nginx
    resources: {}
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:alpine
        name: nginx
        resources: {}
```

Resource model

- When create a pod, the amount of CPU and memory that a container needs - *requests*, and a hard limit on what it may consume - *limits*, can be specified.
  - CPU unit: core
  - Memory unit: byte

```yaml
apiVersion: v1
kind: Pod
metadata:
 labels:
   run: nginx
 name: nginx
spec:
 containers:
 - image: nginx
   name: nginx
   resources:
     requests:
       cpu: 200m
       memory: 100Mi
```

The container request at least 200 milicore of CPU (⅕ of a single core) and 100Mi (0.1M) of memory.

Resource model

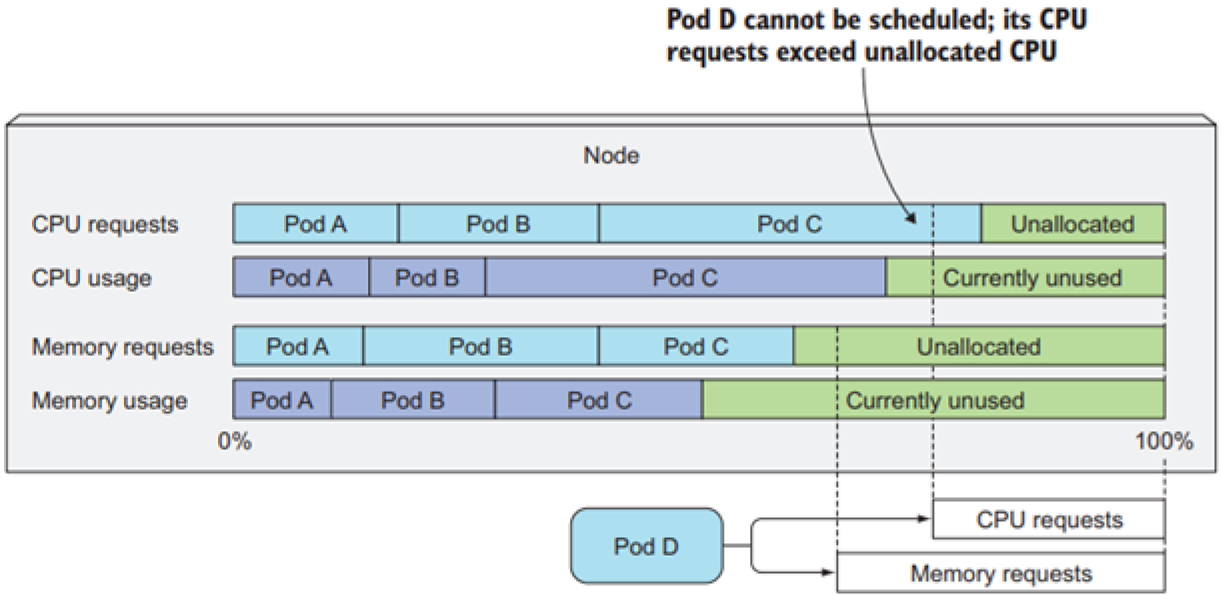


Figure 14.1 The Scheduler only cares about requests, not actual usage.

Init containers

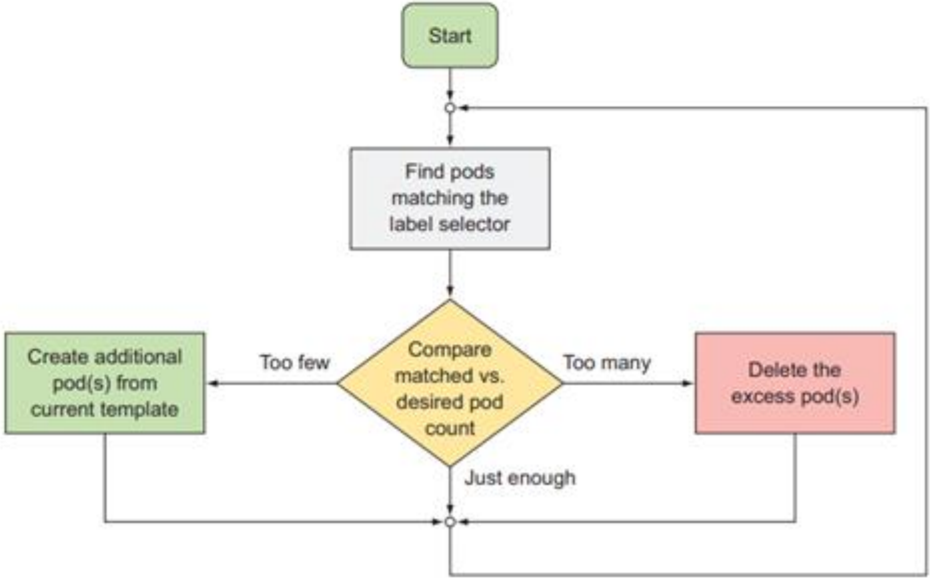
Init containers run before app container are started.

Init containers are exactly like regular containers, except:

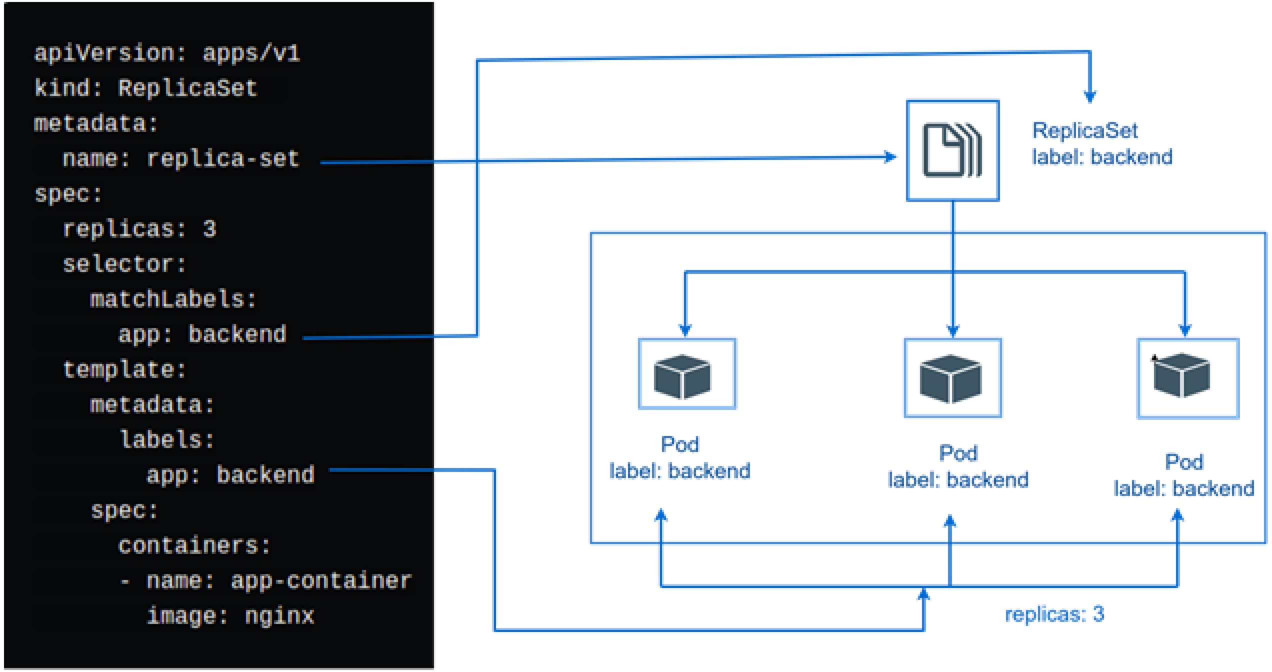
- Init containers always run to completion.
- Each init container must complete successfully before the next one starts.

ReplicaSet

- *ReplicaSet* ensures pods are always kept running.
- If the pod disappears for any reason, the *ReplicaSet* notices the missing pod and creates a replacement pod.


Start
Find pods matching the label selector
Compare matched vs. desired pod count
Too few
Create additional pod(s) from current template
Too many
Delete the excess pod(s)
Just enough

ReplicaSet

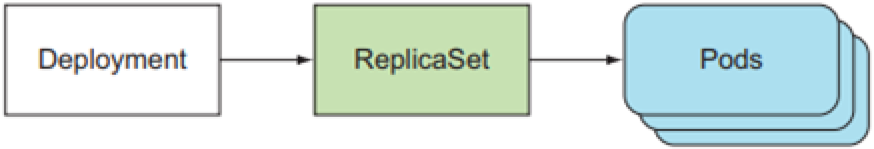


```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: replica-set
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: app-container
        image: nginx
```

Deployment

- *Deployment* provides a declarative way to describe the desired state of your application. Kubernetes then takes care of creating and managing the necessary *ReplicaSets* and Pods to ensure that the desired state is achieved.

- Each update creates a *pod-template-hash* label by hashing *podTemplate* and then assign to both the *ReplicaSet* and subsequent *Pods*.



Deployment

- *.spec.strategy:* specifies the strategy used to replace old Pods by new ones. Available options:
  - *Recreate*: All existing Pods are killed before new ones are created.
  - *RollingUpdate* (default): Updates Pods in a rolling update fashion.

- *.spec.revisionHistoryLimit:* specifies the number of old ReplicaSets to retain to allow rollback. By default, 10 old *ReplicaSets* will be kept.

Deployment

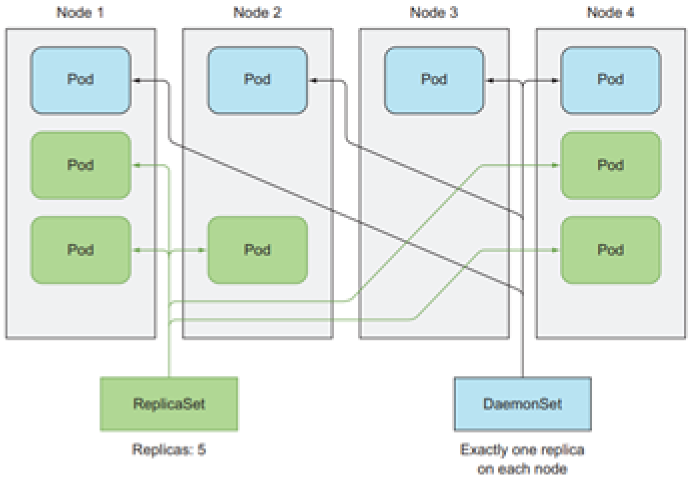
```
thainm5hls@xps:~/$ kubectl create deployment nginx-deployment --image nginx:1.16 --replicas 3
deployment.apps/nginx-deployment created
```

```
thainm5hls@xps:~/$ kubectl get pod
NAME                               READY   STATUS                RESTARTS   AGE
nginx-deployment-66fff56987-gs9bs  1/1     Running               0          30s
nginx-deployment-66fff56987-nsnms  1/1     Running               0          30s
nginx-deployment-66fff56987-tgg9p  1/1     Running                    0
30s
```

```
thainm5hls@xps:~/$ kubectl get deployments.apps
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           118s
```

DaemonSet

- *DaemonSet* ensures there is exactly one instance of a pod running on each node, even with a new added node, and is ideal for cluster wide services such as log forwarding, or health monitoring.
- *DaemonSet* **skips** default scheduling mechanisms.



JOB

- *Job* ensures one or more pods are executed and successfully terminate.
- If :
  - The process running inside the Pod’s container finish successfully, the pod is considered completed.
  - The process fails and returns an error exit code, the Job can be configured to either restart the container or not.
  - A node failure, the pods on that node that are managed by a Job will be rescheduled to other nodes.
- Pods are NOT cleaned up until the job itself is deleted.

JOB

- *Job* pods can’t use the default policy, because they’re not meant to run indefinitely. Therefore, you need to explicitly set the restart policy to either *OnFailure* or *Never.*

- Some helpful attributes:
  - ***.spec.backoffLimit***: The number of failures before the job itself is considered failed.
  - ***.spec.parallelism***: How many instances of the pod can be run concurrently.
  - ***.spec.completion***: How many instances of the pod can be run concurrently.

Cronjob

- An extension of the Job Controller, it provides a method of executing jobs on a cron-like schedule.

Q&A