# Kubernetes Essential

# Agenda

- Assignment Review & Guides
- Kubernetes concepts(Namespace, Pod, Deployment, Labels/Selector, ReplicaSet)
- Working with Pod
- Lab: Deployment Rolling Update and Rollback

# API OVERVIEW

- The **REST API** is the true **keystone** of Kubernetes.
- **Everything** within Kubernetes is as **an API Object**.
- Referenced within an object as the **apiVersion** and **kind**.

Format:
/apis/<group>/<version>/<resource>

Examples:
/apis/apps/v1/deployments
/apis/batch/v1beta1/cronjobs

# OBJECT MODEL

- Objects are a “record of intent” or a persistent entity that represent the desired state of the object within the cluster.

- All objects **MUST** have apiVersion, kind, and poses the nested fields metadata.name, metadata.namespace, and metadata.uid.

# OBJECT EXPRESSION - YAML

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

- Three basic datatypes:
  - **mappings** - hash or dictionary,
  - **sequences** - array or list
  - **scalars** - string, number, boolean etc

```
apiVersion: v1
kind: Pod
metadata:
  name: sample
  namespace: test
spec:
  containers:
  - name: container1
    image: nginx
  - name: container2
    image: alpine
```

# YAML VS JSON

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: pod-example
spec:
    containers:
    - name: nginx
      image: nginx:stable-alpine
        ports:
        - containerPort: 80
```

```json
{
 "apiVersion": "v1",
 "kind": "Pod",
 "metadata": {
     "name": "pod-example"
 },
 "spec": {
     "containers": [
         {
             "name":"nginx",
             "image": "nginx:stable-alpine",    [
             "ports":
               { "containerPort": 80 }
             ]
         }
     ]
 }
}
```

Are you wondering about the YAML schema?
kubectl explain is your friend ;)

What about kinds or versions ?
kubectl api-versions is your friend ;)

# CORE CONCEPTS

# NAMESPACES

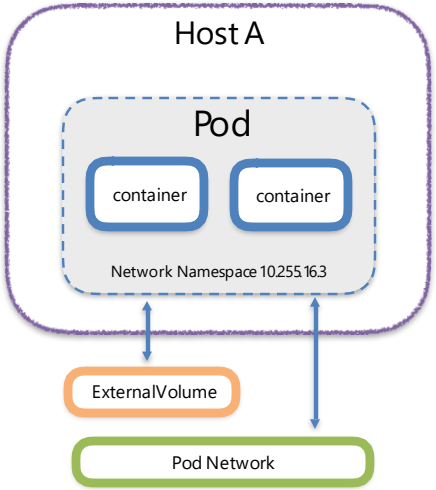

Namespaces are a logical cluster or environment, and are the primary method of partitioning a cluster or scoping access.

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```

```
$ kubectl get ns --show-labels
NAME              STATUS     AGE          LABELS
default           Active     11h          <none>
kube-public       Active     11h          <none>
kube-system       Active     11h          <none>
prod              Active     6s           app=MyBigW
                                          ebApp
```

# PODS

- **Atomic unit** or smallest "*unit of work*" of Kubernetes.
- Foundational building block of Kubernetes Workloads.
- 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:
  name: sample
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
      name: http
      protocol: TCP
    env:
    - name: MYVAR
      value: isAwesome
    command: ["/bin/sh", "-c"]
    args: ["echo ${MYVAR}"]
    readinessProbe:
      tcpSocket:
        port: http
        initialDelaySeconds: 10
        periodSeconds: 10
    livenessProbe:
      httpGet:
        path: /
        port: http
        initialDelaySeconds: 30
        periodSeconds: 60
```

Name of the container

Container Image

Array of environment variables

Array of ports to expose

Entrypoint Array ~ Docker ENTRYPOINT

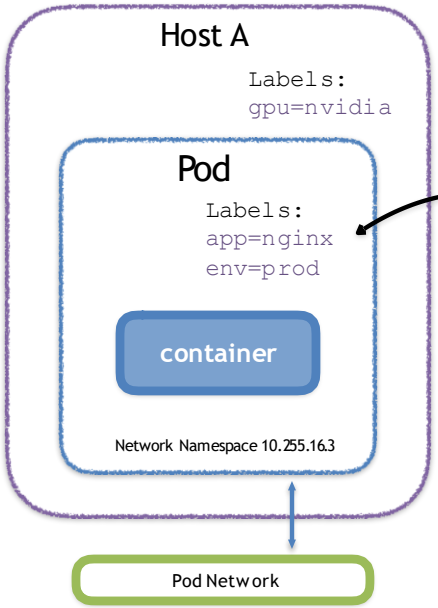
Arguments to pass to the command ~ Docker CMD

Tells when the application is ready to receive requests

Checks if the container is still alive (running)

# LABELS

- key-value pairs that are used to identify, describe and group together related sets of objects or resources.
- NOT characteristic of uniqueness.
- Have a strict syntax with a slightly limited character set*.



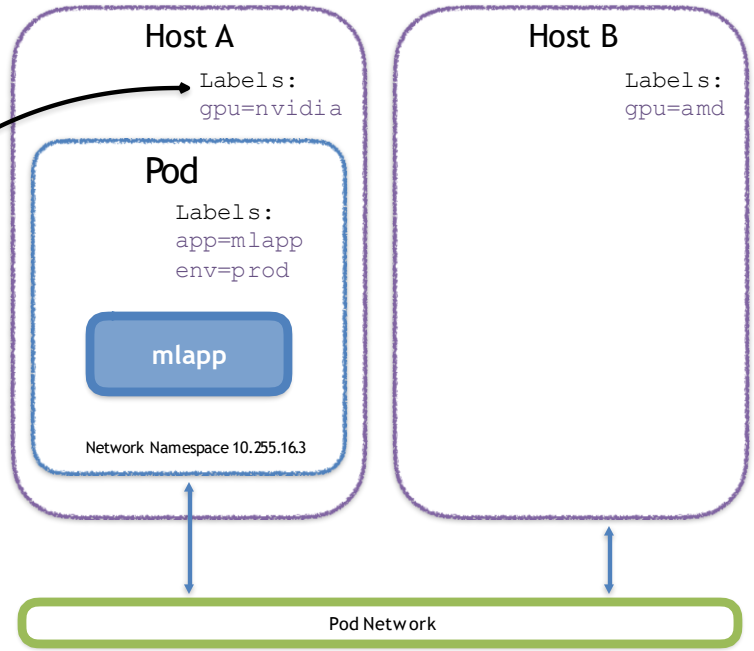

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#syntax-and-character-set

# 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
```



# SERVICES

- **Unified method of accessing** the exposed workloads of Pods.
- **Durable resource** (unlike Pods)
  - static cluster-unique IP
  - static namespaced DNS name

There are 4 major service types:

- ClusterIP (default)
- NodePort
- LoadBalancer
- ExternalName

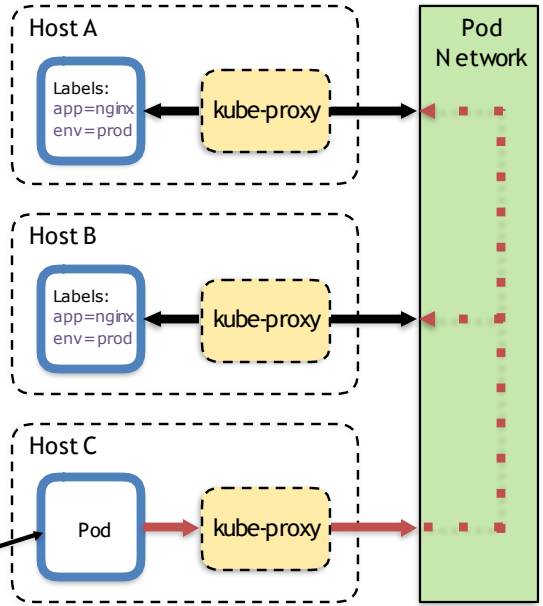
**<service name>.<namespace>.svc.cluster.local**

# CLUSTER IP SERVICE

- The Pod on host C requests the service.
- Hits host iptables and it load-balances the connection between the endpoints residing on Hosts A,B

```
Name:
Selector:       example-prod
Type:           app=nginx,env=prod
IP:             ClusterIP
Port:           10.96.28.176
TargetPort:     <unset>  80/TCP
Endpoints:      80/TCP
                10.255.16.3:80,
                10.255.16.4:80
```

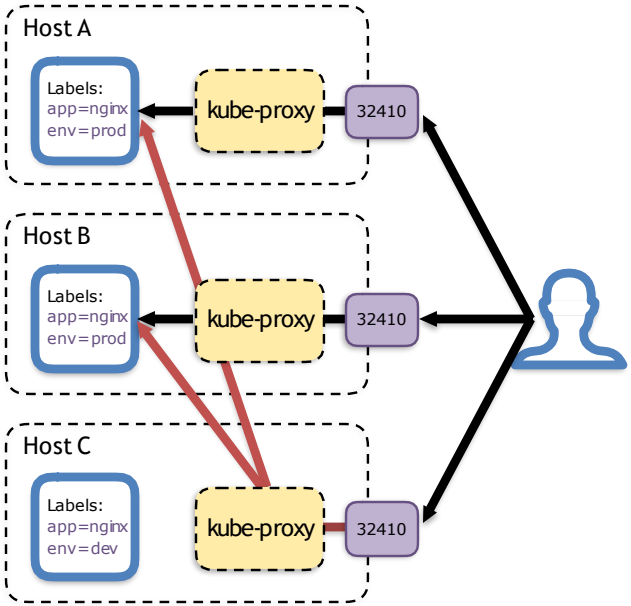
```
/ # nslookup example-prod.default.svc.cluster.local

Name:      example-prod.default.svc.cluster.local
Address 1: 10.96.28.176 example-prod.default.svc.cluster.local
```



# NODE PORT SERVICE

- User can hit any host in cluster on **NodePort** IP and get to service.
- Does introduce extra hop if hitting a host without instance of the pod.

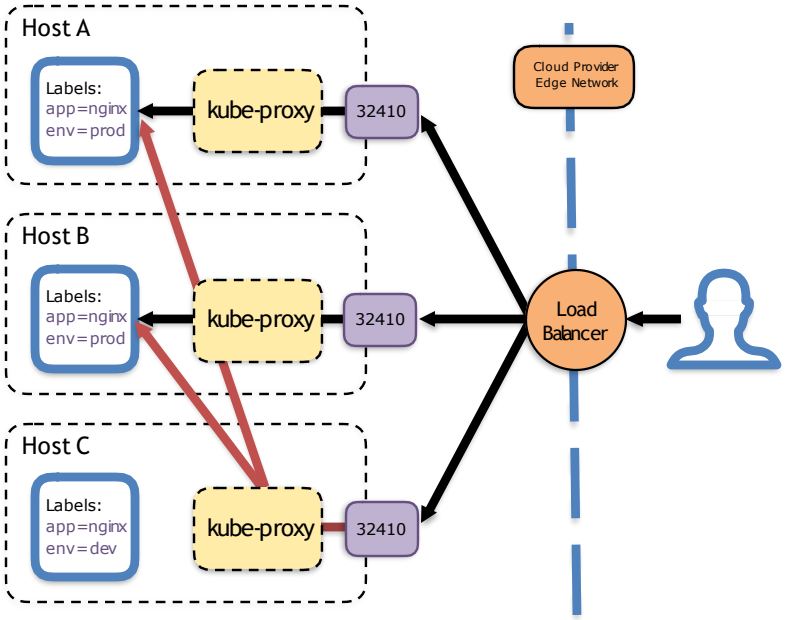
```
Name:           example-prod
Selector:       app=nginx,env=prod
Type:           NodePort
IP:             10.96.28.176
Port:           <unset>  80/TCP
TargetPort:     80/TCP
NodePort:       <unset>  32410/TCP
Endpoints:      10.255.16.3:80,
                10.255.16.4:80
```



# LOAD BALANCER SERVICE

- **LoadBalancer** services extend **NodePort**.
- Works in conjunction with an external system to map a cluster external IP to the exposed service.

```
Name:          example-prod
Selector:      app=nginx,env=prod
Type:          LoadBalancer
IP:            10.96.28.176
LoadBalancer
Ingress:       172.17.18.43
Port:          <unset>  80/TCP
TargetPort:    80/TCP
NodePort:      <unset>  32410/TCP
Endpoints:     10.255.16.3:80,
               10.255.16.4:80
```



# WORKLOADS

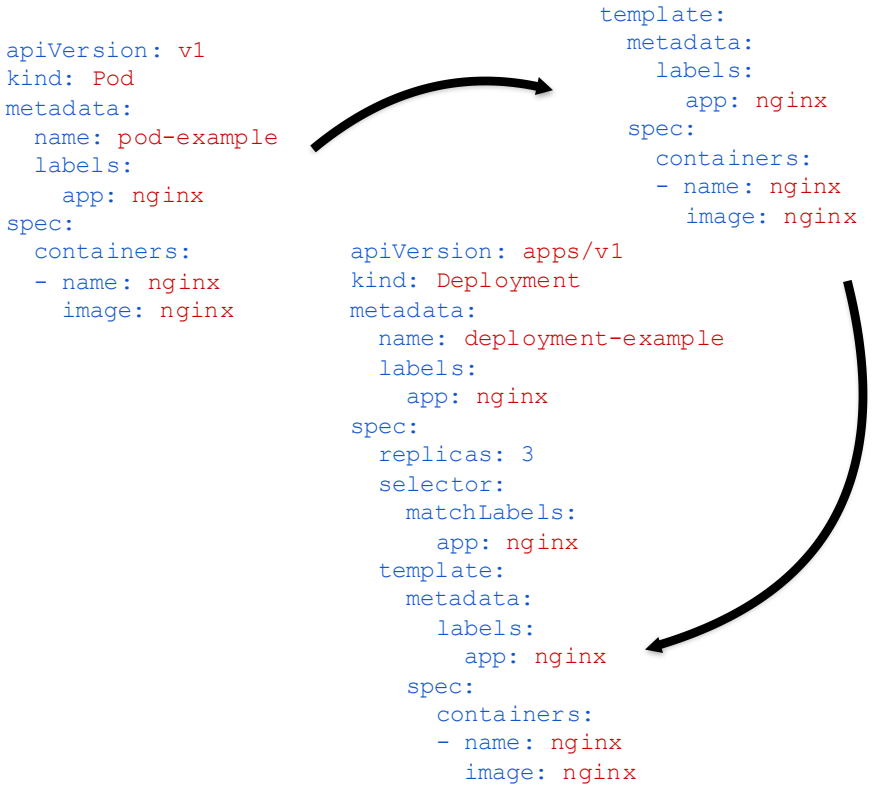
# WORKLOADS

- 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.

- ReplicaSet
- Deployment
- DaemonSet
- StatefulSet
- Job
- CronJob

# POD TEMPLATE

- Workload Controllers manage instances of Pods based off a provided template.
- Pod Templates are Pod specs with limited metadata.
- Controllers use Pod Templates to make actual pods



# RESOURCE MODEL

- **Request**: amount of a resource allowed to be used, with a strong guarantee of availability.

CPU (seconds/second), RAM (bytes)

Scheduler will not over-commit requests

Limit: max amount of a resource that can be used, regardless of guarantees
- Scheduler ignores limits

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-example
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

# INIT CONTAINERS

If we compare a container template with a Class definition in Java, an initContainer would be the constructor of a class; while a running Pod, would be an instance of thatclass.

Can be used for everything that should happen before the containers within a Pod start running. For example: wait for dependencies, initialize volumes or databases, verify requirements, etc.

# REPLICA SET

- Primary method of managing pod replicas and their lifecycle.
- Includes their scheduling, scaling, and deletion.
- Their job is simple: **Always ensure the desired number of pods are running.**


- **replicas**: The desired number of instances of the Pod.
- **selector**: The label selector for the **ReplicaSet** will manage **ALL** Pod instances that it targets; whether it's desired or not.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: rs-example
spec:
  replicas: 3
  selector:
    matchLabels:
      app:  nginx
      env: prod
  template:
    <pod template>
```

# DEPLOYMENT

- Declarative method of managing Pods via ReplicaSets.
- Provide rollback functionality and update control.
- Updates are managed through the **pod-template-hash label**.
- Each iteration creates a unique label that is assigned to both the **ReplicaSet** and subsequent Pods.


- **revisionHistoryLimit**: The number of previous iterations of the Deployment to retain.
- **strategy**: Describes the method of updating the Pods based on the **type**. Valid options are:
  - **Recreate**: All existing Pods are killed before the new ones are created.
  - **RollingUpdate**: Cycles through updating the Pods according to the parameters: **maxSurge** and **maxUnavailable**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    <pod template>
```

# DAEMON SET

- Ensure that all nodes matching certain criteria will run an instance of the supplied Pod.
- They **bypass** default scheduling mechanisms.
- Are ideal for cluster wide services such as log forwarding, or health monitoring.
- Revisions are managed via a **controller-revision-hash** label.


- **spec.template.spec.nodeSelector**: The primary selector used to target nodes.
- **Default Host Labels**:
  - kubernetes.io/hostname
  - beta.kubernetes.io/os
  - beta.kubernetes.io/arch
- **Cloud Host Labels**:
  - failure-domain.beta.kubernetes.io/zone
  - failure-domain.beta.kubernetes.io/region
  - beta.kubernetes.io/instance-type

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: ds-example
spec:
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
  template:
    spec:
      nodeSelector:
        nodeType: edge
      <pod template>
```

# STATEFULSET

- Tailored to managing Pods that must persist or maintain state.
- Pod identity including **hostname**, **network**, and **storage** **WILL** be persisted.
- Assigned a unique ordinal name following the convention of '<statefulset name>-<ordinal index>'.
- Naming convention is also used in Pod's network Identity and Volumes.
- Pod lifecycle will be ordered and follow consistent patterns.
- Revisions are managed via a **controller-revision-hash** label.


```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: sts-cassandra
spec:
  replicas: 3
  selector:
    matchLabels:
      app: cassandra
  serviceName: cassandra
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      partition: 0
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      containers:
      - name: cassandra-node
        image: cassandra:3.11.4
        env:
        - name: CASSANDRA_SEEDS
          value: sts-cassandra-0.cassandra
        - name: CASSANDRA_CLUSTER_NAME
          value: my-cluster
        ports:
          - containerPort: 7000

          - containerPort: 7199

          - containerPort: 9042

        volumeMounts:
        - name: data
          mountPath: /cassandra_data
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: [ "ReadWriteOnce"
      ]
      storageClassName:
      standard resources:
        requests:
          storage: 100Gi
```

The name of the associated headless service; or a service without a ClusterIP

Pods with an ordinal greater than the partition value will be updated one-by-one in reverse order

Template of the persistent volume(s) request to use for each instance of the StatefulSet.

# JOB

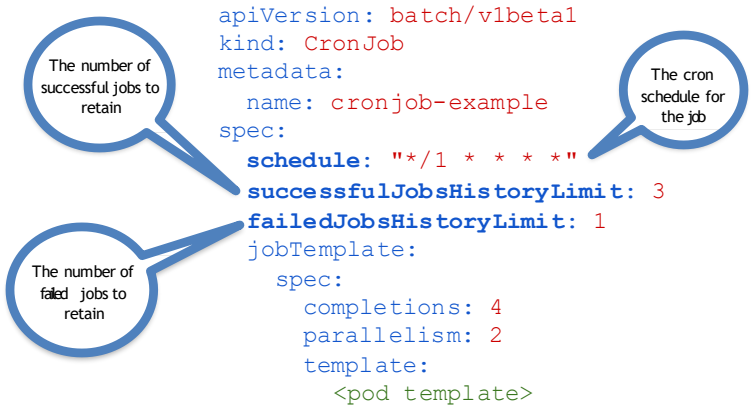
- Job controller ensures one or more pods are executed and successfully terminate.
- Will continue to try and execute the job until it satisfies the completion and/or parallelism condition.
- Pods are NOT cleaned up until the job itself is deleted.


- **backoffLimit**: The number of failures before the job itself is considered failed.
- **completions**: The total number of successful completions desired.
- **parallelism**: How many instances of the pod can be run concurrently.
- **spec.template.spec.restartPolicy**:Jobs only support a restartPolicy of type **Never** or **OnFailure**.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: job-example
spec:
  backoffLimit: 4
  completions: 4
  parallelism: 2
  template:
    spec:
      restartPolicy:Never
        template:
      <pod-template>
```

# CRONJOB

- An extension of the Job Controller, it provides a method of executing jobs on a cron-like schedule.
- CronJobs within Kubernetes use **UTC ONLY.**

# Q&A