Kubernetes

FROM ZERO TO ~~HERO~~ Seasoned Beginner

Seshagiri Sriram
Based on work of Alejandro Galue
& Others

WHY CONTAINERS

- Build, Ship, and Run any App, Anywhere.
- No more dependency problems; e.x. no RPM/DEB or dynamic libraries incompatibility issues.
- No more: "but it works on my machine", as the same image used in dev is deployed to prod.
- Serverless implementations are mainly backed by containers.

WHAT IS KUBERNETES

- Container-orchestration system for

  - automating deployment
  - scaling
  - operations

  of application **containers** across **clusters** of hosts

- Designed by Google, started in 2014 (previously *Borg*)
  now maintained by Cloud Native Computing Foundation

- v1.0 available since 2015, currently at v1.18

WHAT IS KUBERNETES

- A distributed cluster technology that manages container-based systems in a declarative manner using an API (a container orchestrator).

- Designed from the ground-up as a loosely coupled collection of components centered around deploying, maintaining and scaling workloads.

- Abstracts away the underlying hardware of the nodes and provides a uniform interface for workloads to be both deployed and consume the shared pool of resources.

- Works as an engine for resolving state by converging actual and the desired state of the system (self-healing).

WHY KUBERNETES

- Be able to manage hundreds or thousands of containers, on a fleet of hundred or thousands of nodes.

- Be able to deploy an application without worrying about the hardware infrastructure.

- Be able to guarantee that applications will stay running according with the specifications.

- Be able to facilitate scaling and upgrades not only for the workloads but for the k8s cluster itself.

I ❤ Docker - why k8s?

- I run Google
- Ok, I don't, but all the cool kids run k8s !!
- My containers no longer fit to a single node

Kubernetes enables to seamlessly run Docker containers over multiple computing nodes

(as **Docker Swarm** does as well btw)

HOW TO GET A CLUSTER

- Minikube / Microk8s / K3s
  Kubernetes on a single machine
- KOPS / kubeadm
  Kubernetes on multiple machines
- AWS / Azure / Google Cloud
  Managed Kubernetes in the cloud

SOME TERMINOLOGY

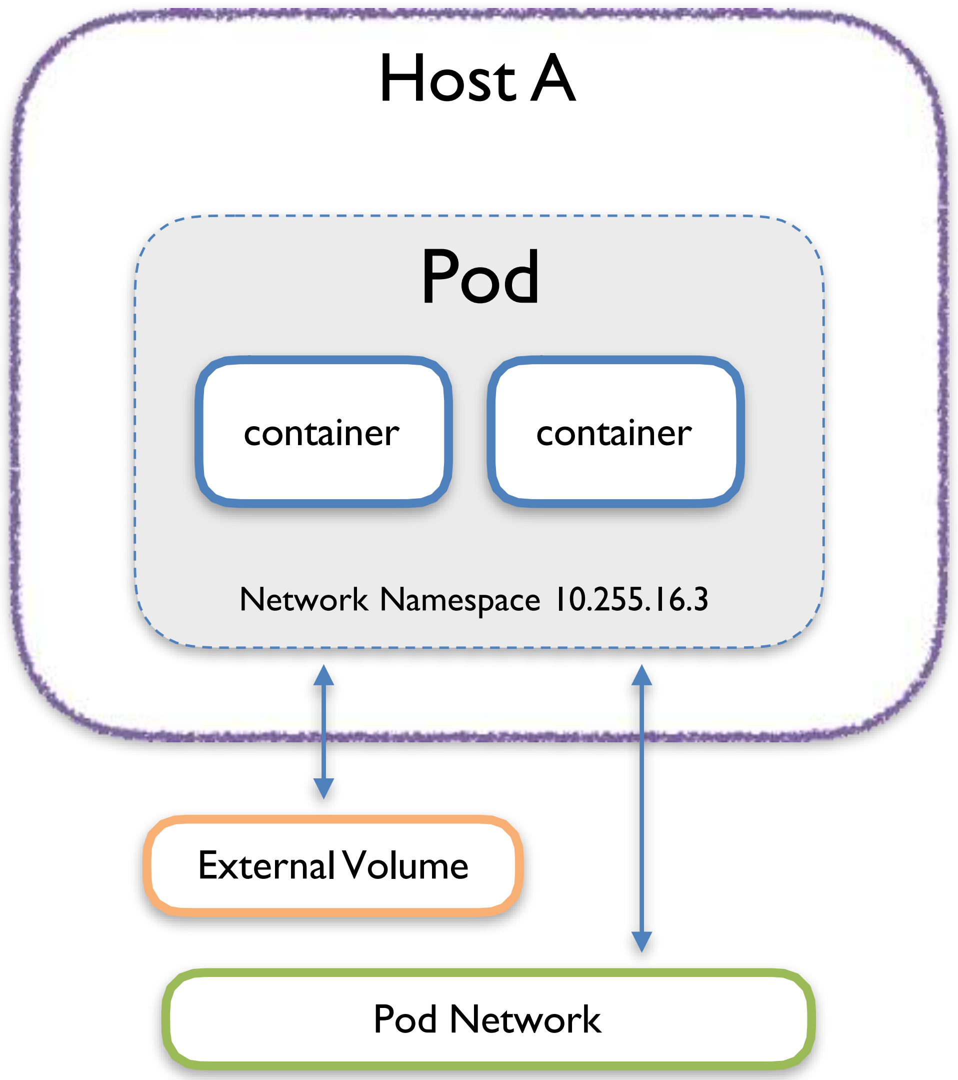
Pod	Wraps a docker container
Replicaset	Replicates a POD
Deployment	Defines a replicaset with other features
Service	Load Balancer for a set of pods
Ingress	Enables Inbound Connections
ConfigMap/Secrets	Configuration Data
PersistentVolume	Mountable Volumes for persistent data

A COUPLE OF KEY CONCEPTS…

PODS

Ephemeral

- **Atomic unit** or 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**.

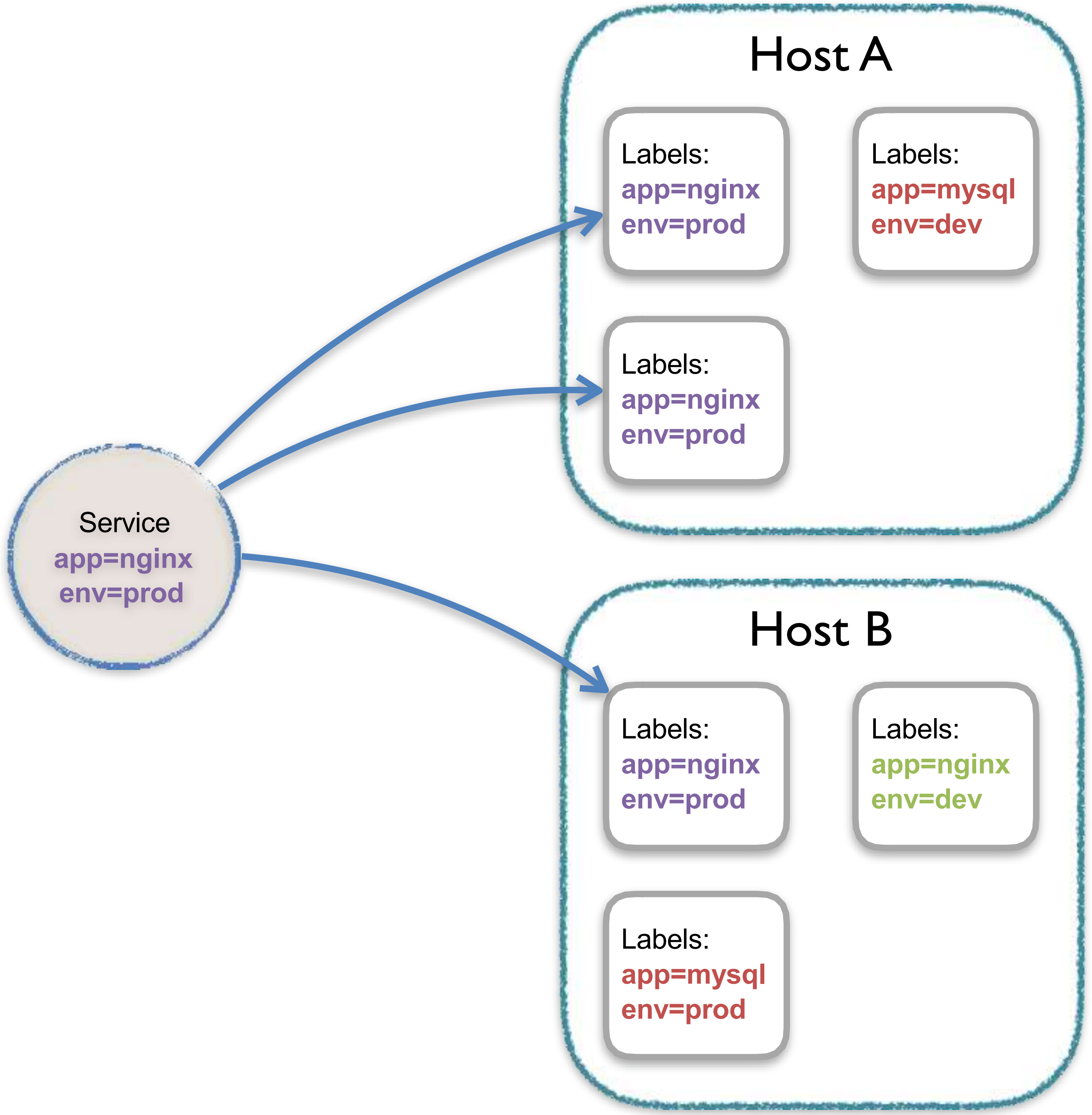


SERVICES

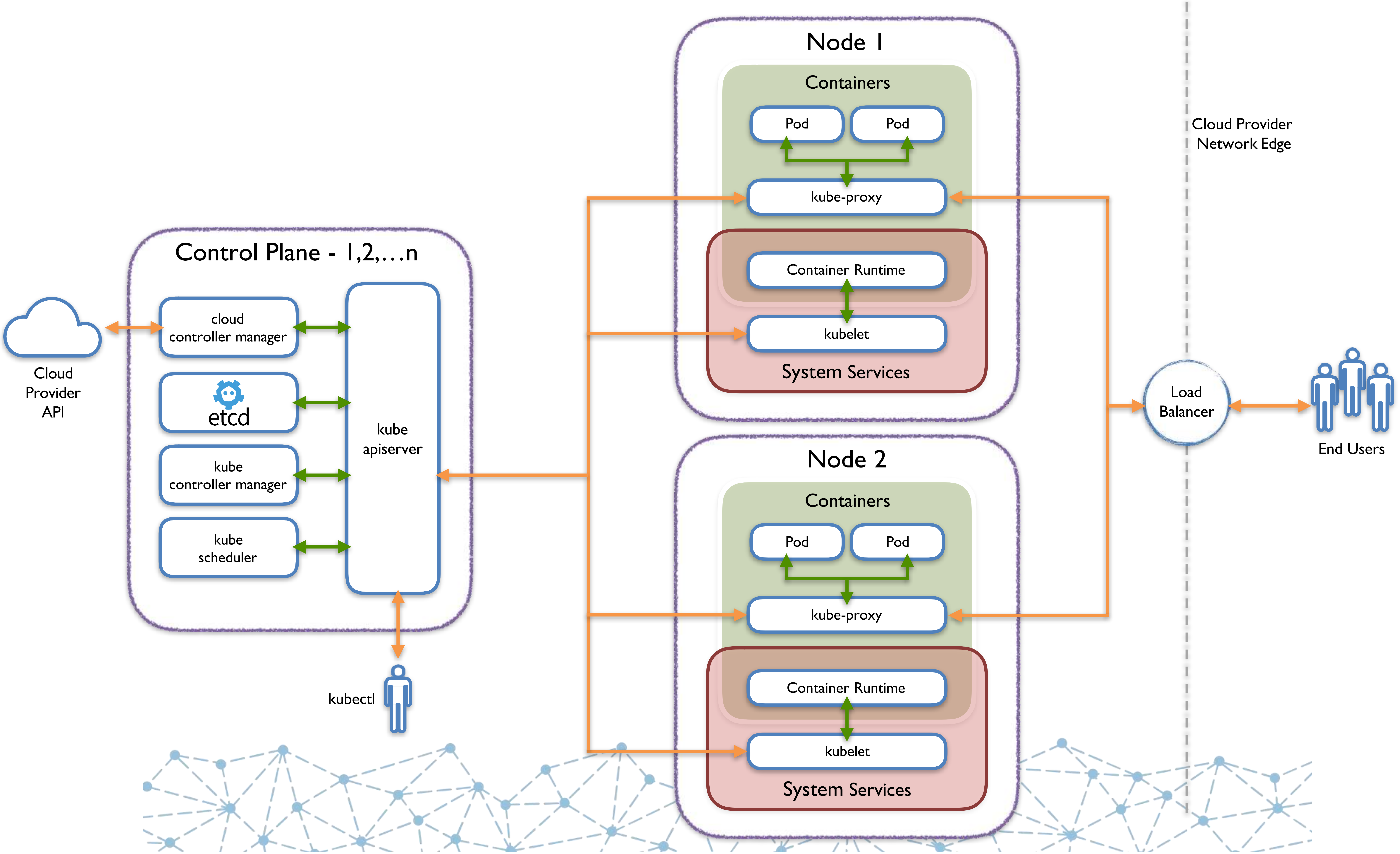
NOT Ephemeral

- **Unified method of accessing** the exposed workloads of Pods.

- **Durable resource**
  - Static cluster IP
  - static namespaced DNS name

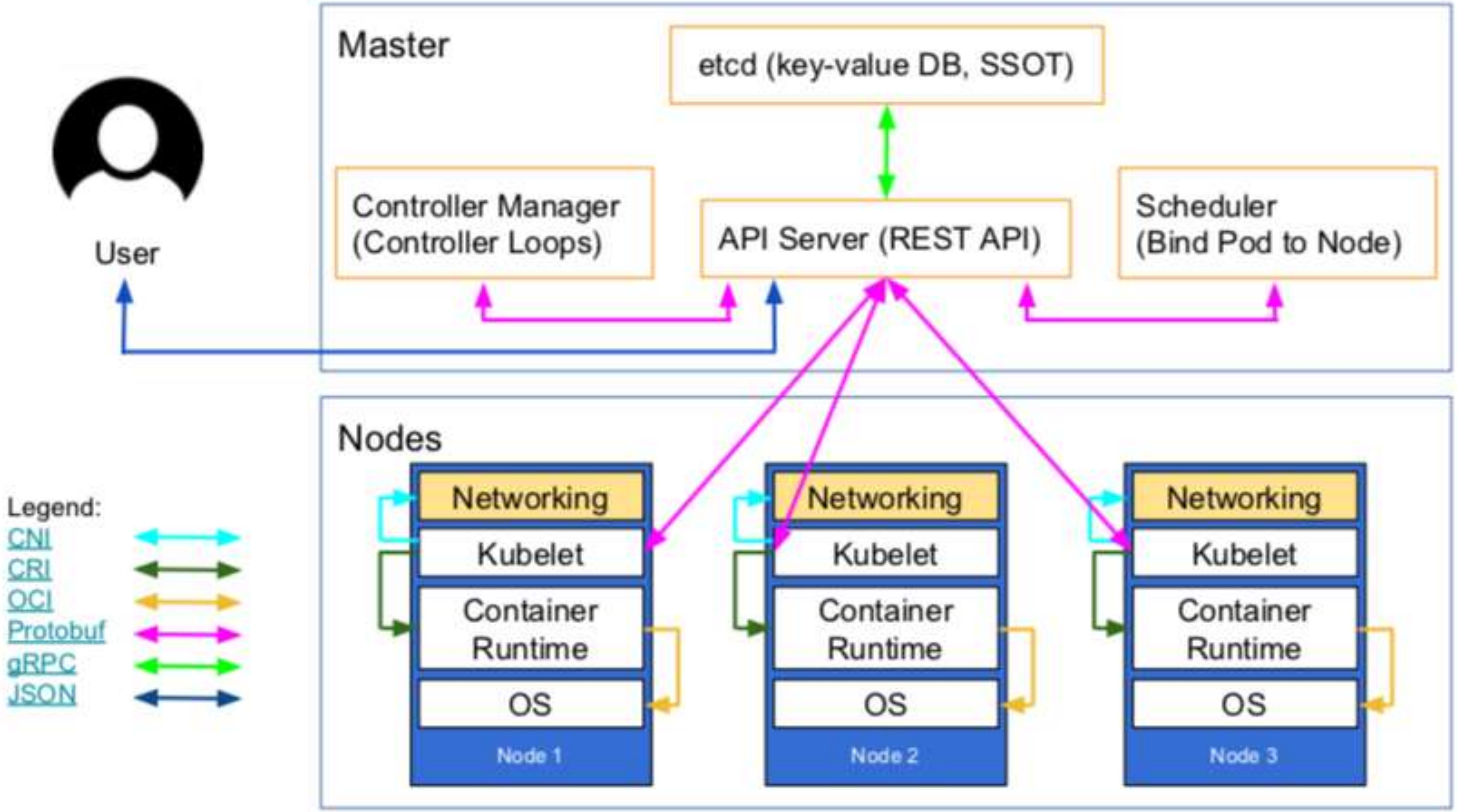


ARCHITECTURE OVERVIEW


Node 1
Containers
Pod
Pod
kube-proxy
Container Runtime
kubelet
System Services
Cloud Provider
Network Edge
Control Plane - 1,2,…n
cloud
controller manager
etcd
kube
controller manager
kube
scheduler
kube
apiserver
Cloud
Provider
API
kubectl
Load
Balancer
End Users
Node 2
Containers
Pod
Pod
kube-proxy
Container Runtime
kubelet
System Services

K8S ARCHITECTURE

master



KUBE-APISERVER

master

- Provides a forward facing REST interface into the kubernetes control plane and datastore.

- All clients and other applications interact with kubernetes **strictly** through the API Server.

- Acts as the gatekeeper to the cluster by handling authentication and authorization, request validation, mutation, and admission control in addition to being the front-end to the backing datastore.

etcd

master

- etcd acts as the cluster datastore.
- Purpose in relation to Kubernetes is to provide a strong, consistent and highly available key-value store for persisting cluster state.
- Stores objects and config information.
- Uses “Raft Consensus” among a quorum of systems to create a fault-tolerant consistent “view” of the cluster.

KUBE-CONTROLLER-MANAGER

master

- Serves as the primary daemon that manages all core component control loops.

- Monitors the cluster state via the apiserver and steers the cluster towards the desired state.

KUBE-SCHEDULER

master

- Component on the master that watches newly created pods that have no node assigned, and selects a node for them to run on.

- Factors taken into account for scheduling decisions include individual and collective resource requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference and deadlines.

Optional

CLOUD-CONTROLLER-MANAGER

master

- Daemon that provides cloud-provider specific knowledge and integration capability into the core control loop of Kubernetes.

- The controllers include Node, Route, Service, and add an additional controller to handle things such as **PersistentVolume** Labels.

KUBE-PROXY

worker

- Manages the network rules on each node.

- Performs connection forwarding or load balancing for Kubernetes cluster services.

KUBELET

worker

- An agent that runs on each node in the cluster. It makes sure that containers are running in a pod.

- The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy.

CONTAINER RUNTIME ENGINE

worker

A container runtime is a CRI (Container Runtime Interface) compatible application that executes and manages containers.

- **containerd** (Docker)
- **cri-o**
- **rkt**
- **kata** (formerly clear and hyper)
- **virtlet** (VM CRI compatible runtime)

KUBERNETES NETWORKING

- **Pod Network**

  - Cluster-wide network used for pod-to-pod communication managed by a CNI (Container Network Interface) plugin.

- **Service Network**

  - Cluster-wide range of Virtual IPs managed by **kube-proxy** for service discovery.

FUNDAMENTAL NETWORKING RULES

- All containers within a pod can communicate with each other unimpeded.
- All Pods can communicate with all other Pods without NAT.
- All nodes can communicate with all Pods (and vice-versa) without NAT.
- The IP that a Pod sees itself as is the same IP that others see it as.

What if I want to limit communication within Pods ?

Learn about **Network Policies** (and make sure the chosen CNI supports it)

API

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
```

API OVERVIEW

Refer: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/

Custom Resource Definitions are handled by: apiextensions.k8s.io

Pod Templates are handled by: **core.k8s.io**

Read More here: https://pkg.go.dev/k8s.io/api

API OVERVIEW

All YAML files do start with an apiVersion.

This could potentially change from Version to Version of Kubernetes.

Example for a deployment

```
apiVersion: apps/v1beta1              # for >= Kubernetes 1.6
apiVersion: extensions/v1beta1  # for < Kubernetes 1.6
```

API OVERVIEW

APIS are grouped to help extend functionality

Core or legacy are specified like so: apiVersion: v1

For others: use apiVersion: $GROUP_NAME/$VERSION

Refer: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.32/#-strong-api-groups-strong-

Enabling or disabling examples:

```
--runtime-config=batch/v1=false
--runtime-config=storage.k8s.io/v1beta1/csistoragecapacities
```

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 MODEL

- The difference between an object and a resource?
- A Kubernetes object is a persistent entities in the Kubernetes system
- A Kubernetes resource is an endpoint in the Kubernetes API that stores a collection of API objects of a certain kind; for example, the built-in pods resource contains a collection of Pod objects.
- Using this definition, **pods are objects, services are resources.**

OBJECT EXPRESSION - YAML

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

- Three basic data types:
  - **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 ;)

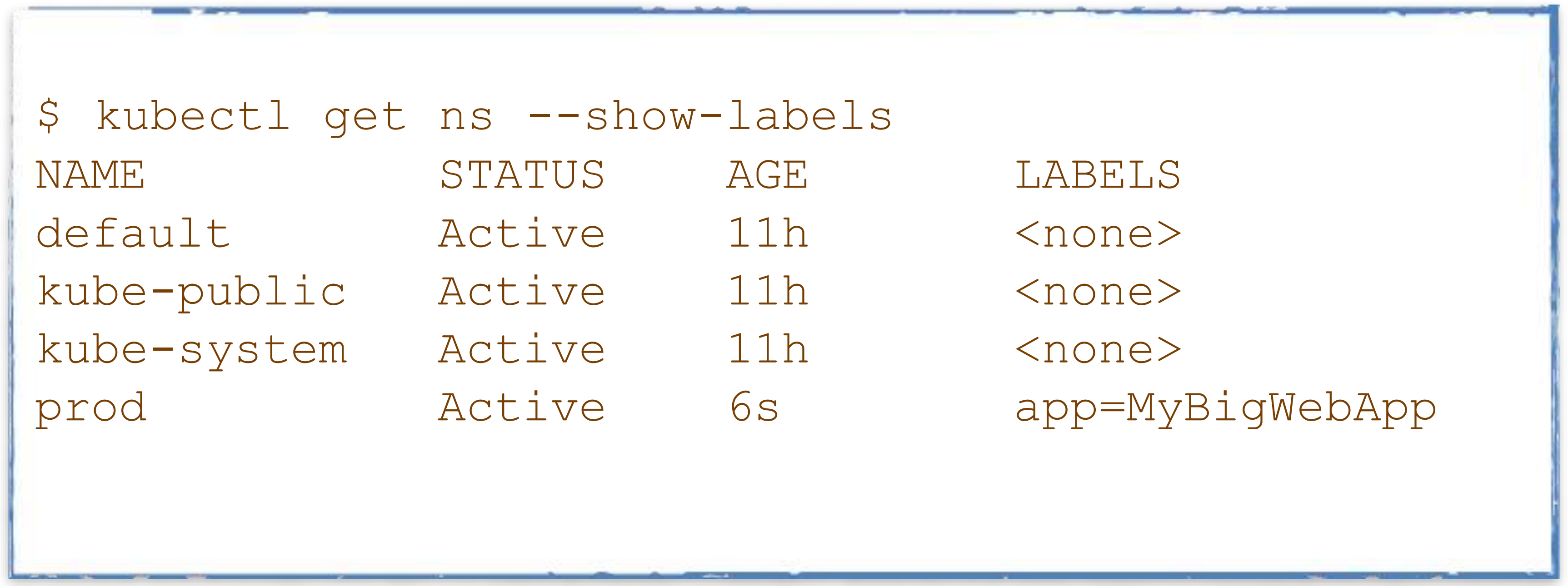
CORE CONCEPTS

Known as Projects in OpenShift

NAMESPACES

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

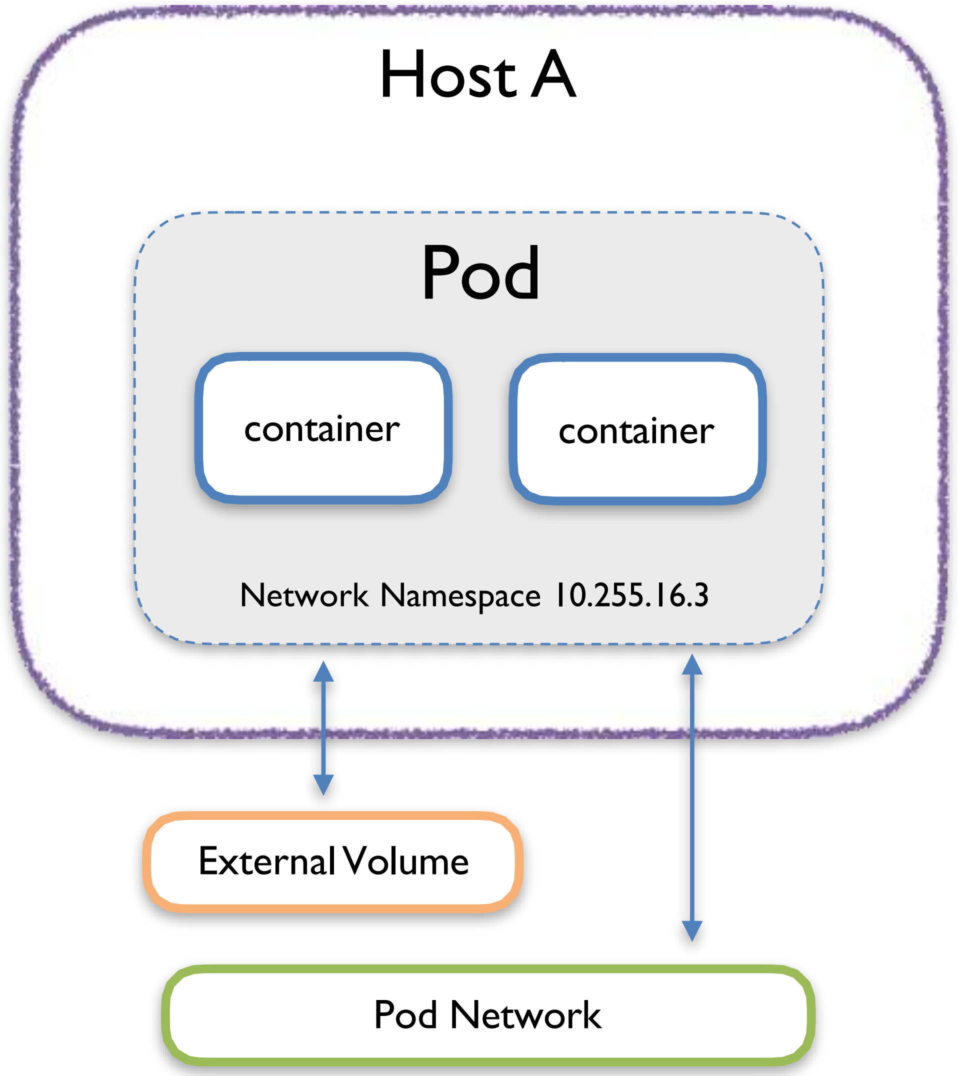
```
apiVersion: v1
kind: Namespace
metadata:
  name: prod
  labels:
    app: MyBigWebApp
```



```
$ 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=MyBigWebApp
```

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

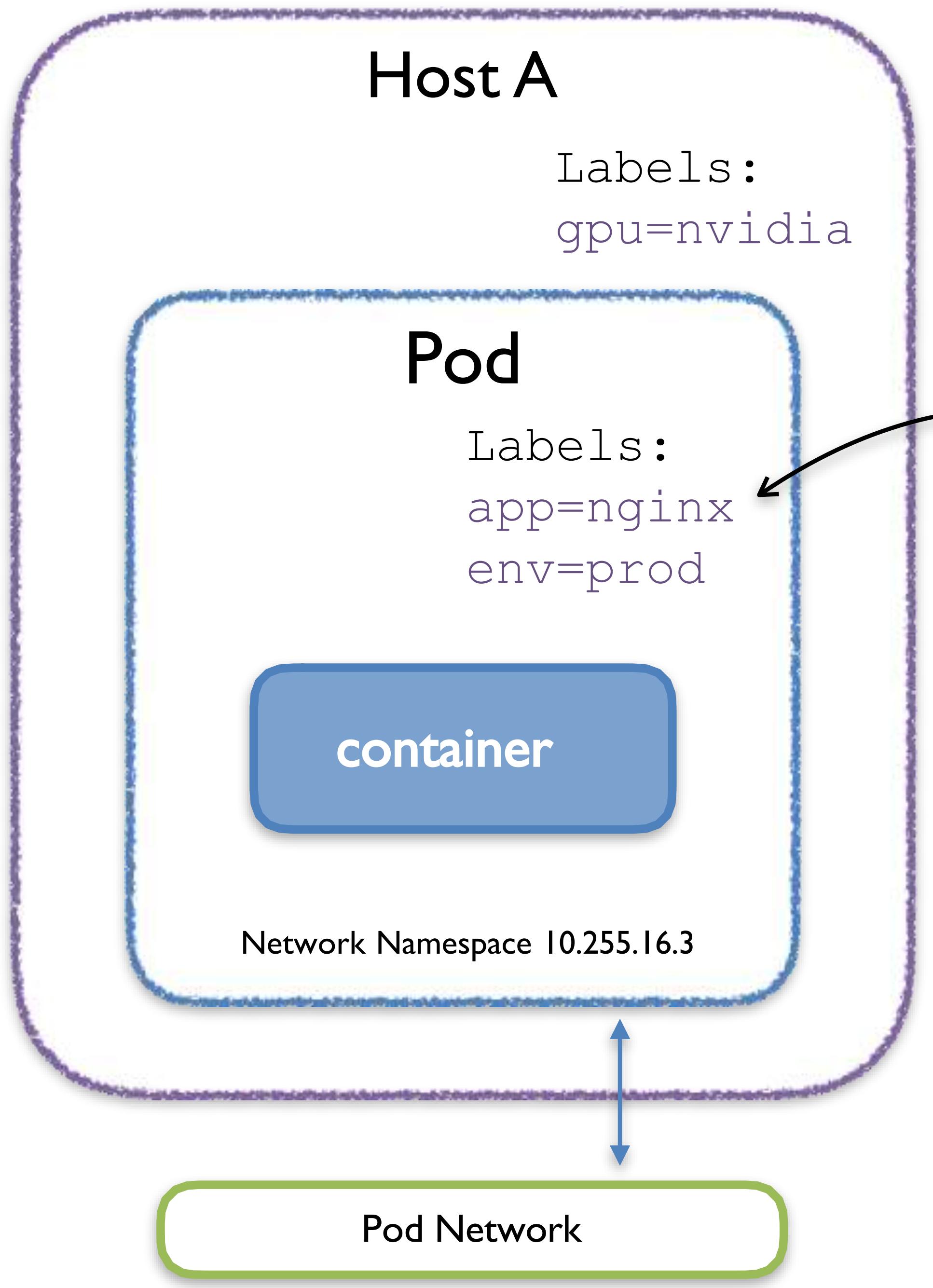
Arguments to pass to the command ~ Docker CMD

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*.



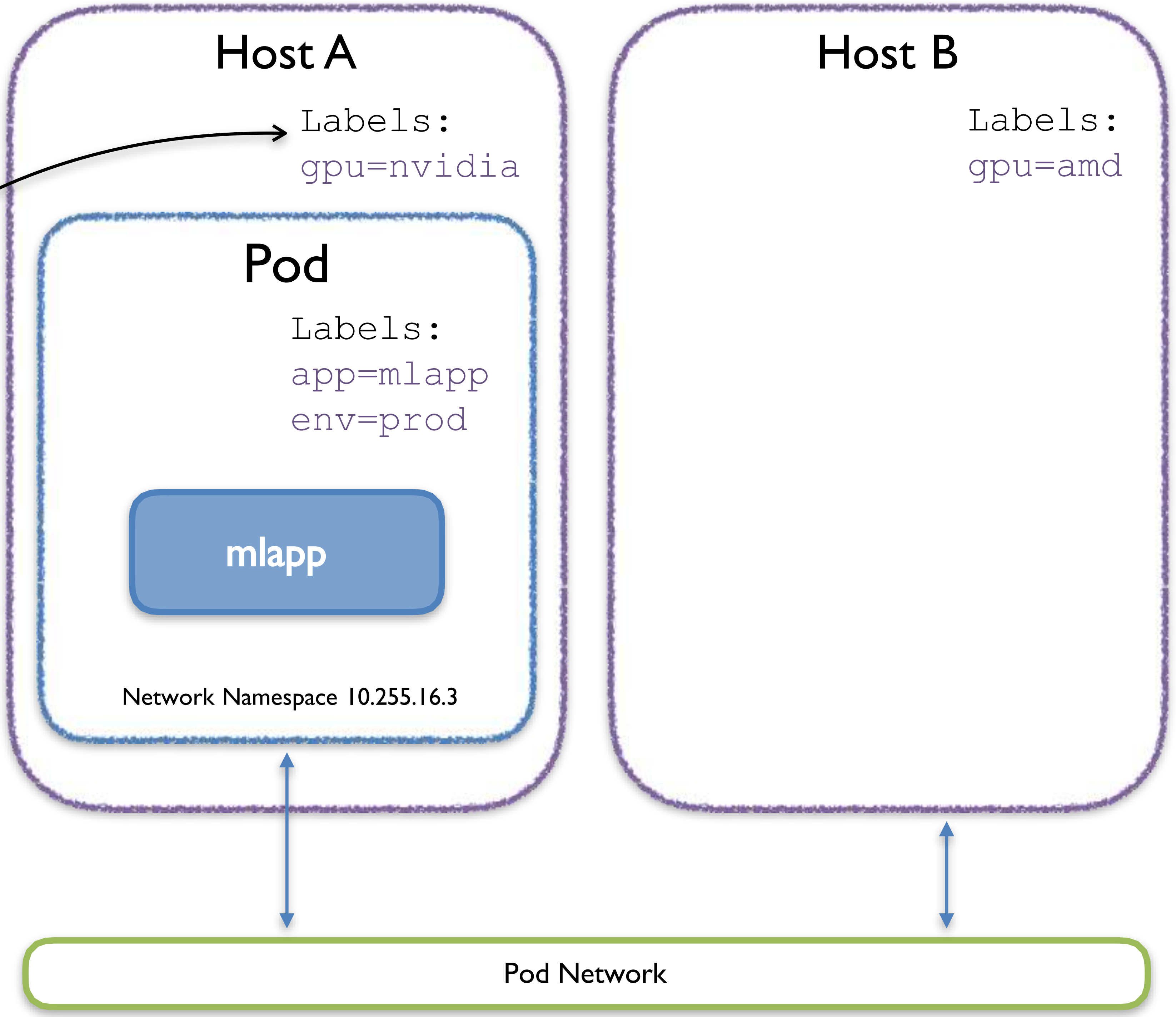
```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-label-example
  labels:
    app: nginx
    env: prod
spec:
  containers:
  - name: nginx
    image: nginx:stable-alpine
    ports:
    - containerPort: 80
```

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
```



Host A

```
Labels:
gpu=nvidia
```

Pod

```
Labels:
app=mlapp
env=prod
```

mlapp

Network Namespace 10.255.16.3

Host B

```
Labels:
gpu=amd
```

Pod Network

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

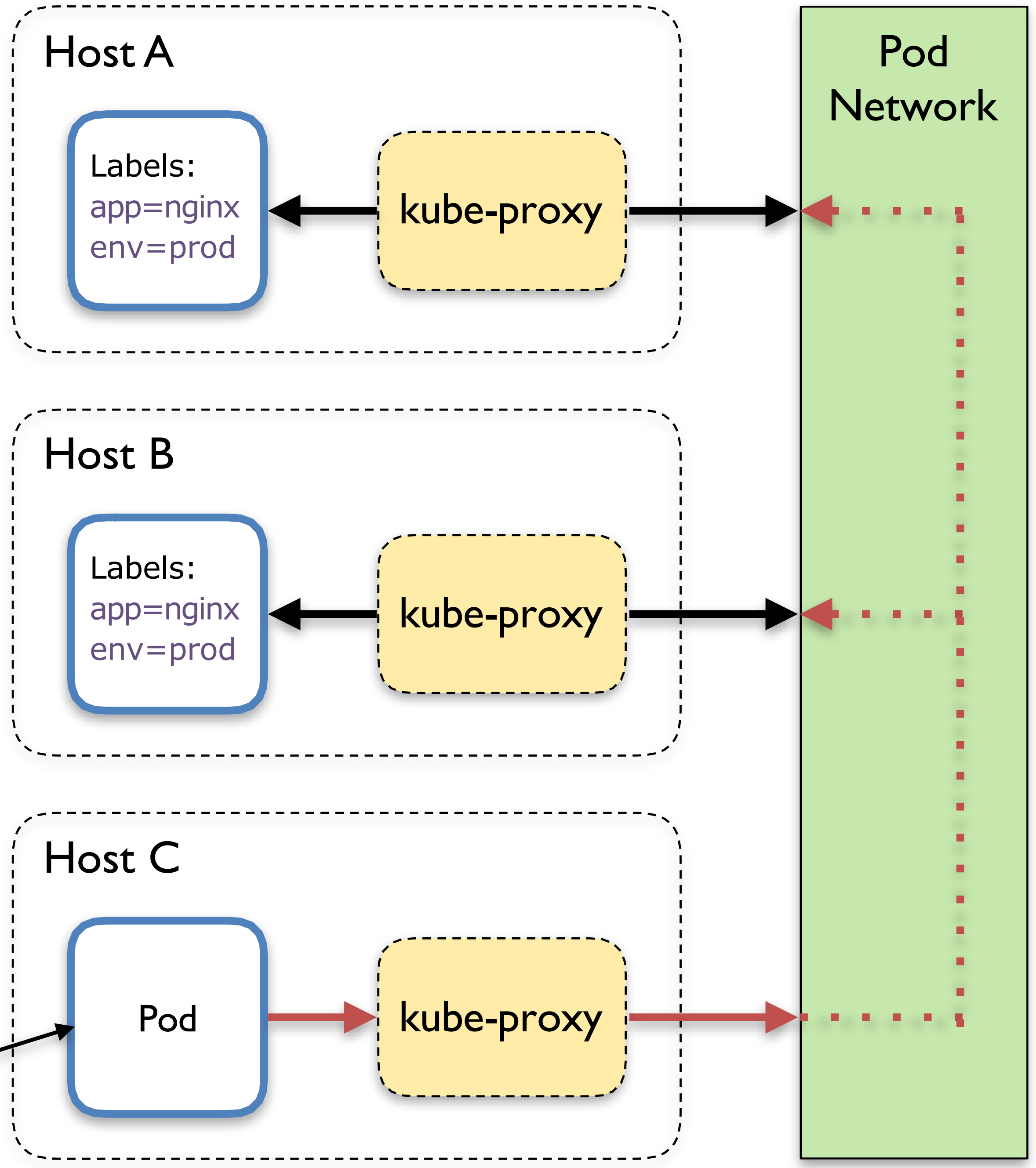
<service name>.<namespace>.svc.cluster.local

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

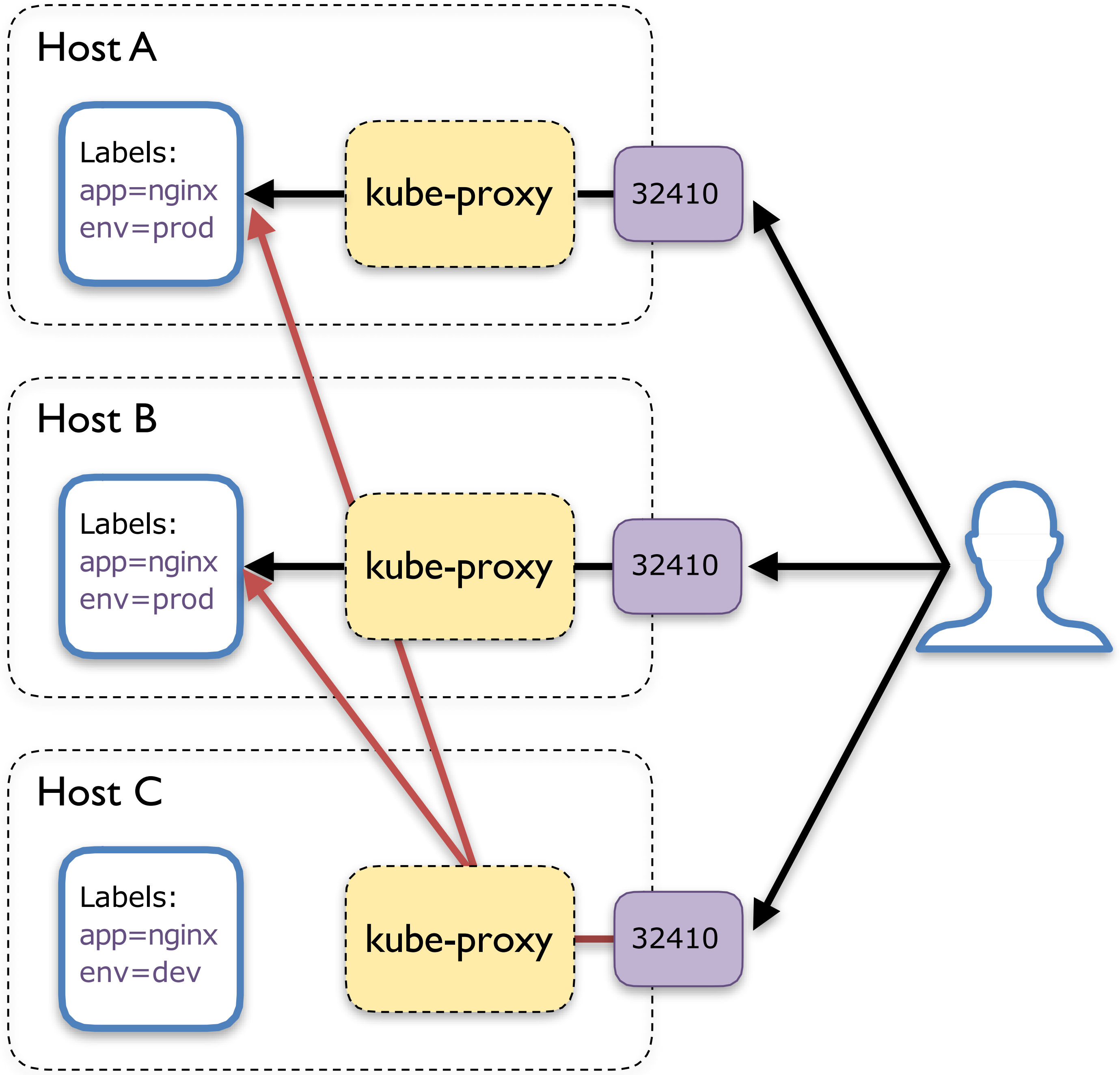
```
/ # nslookup example-prod.default.svc.cluster.local

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.

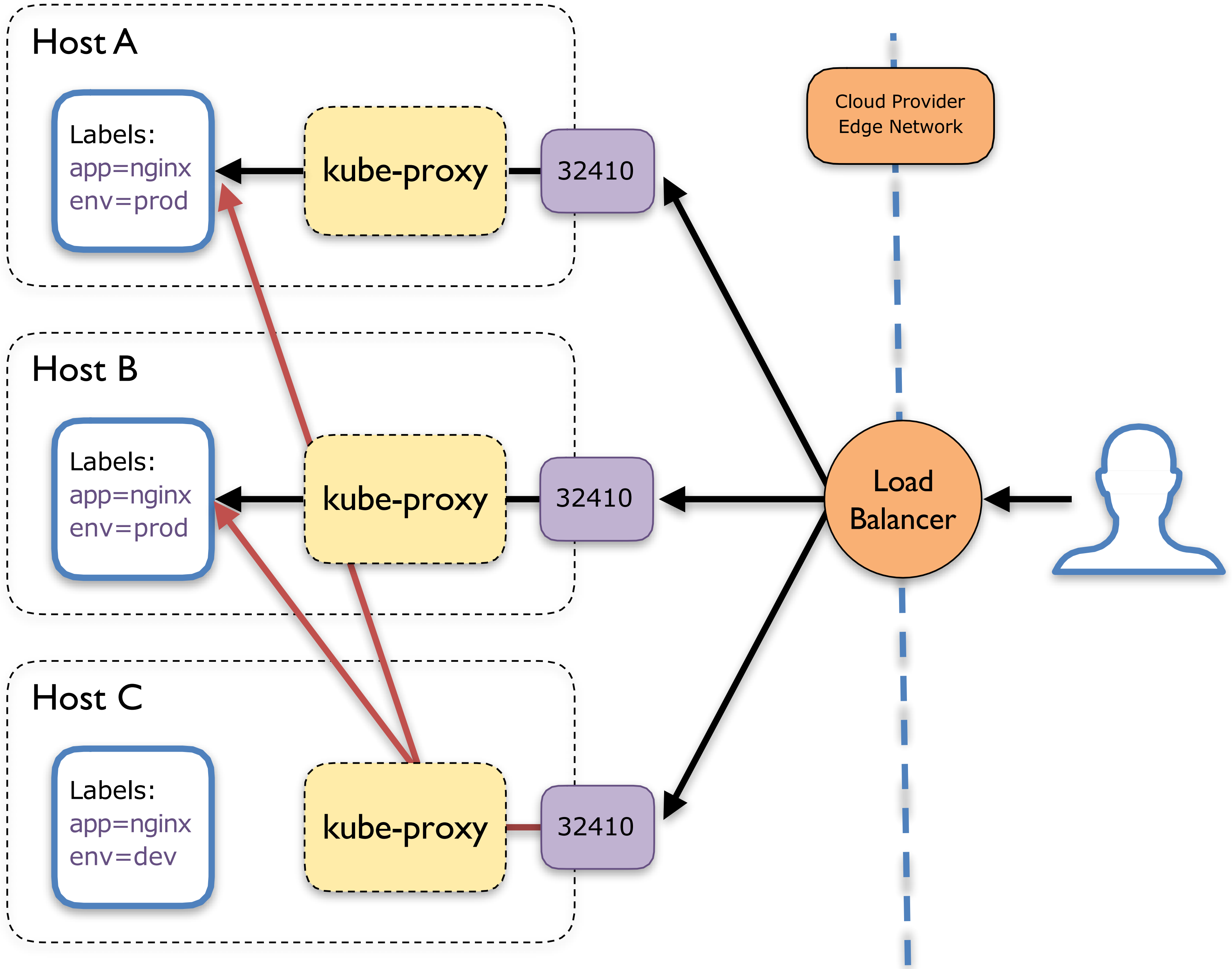
```
Name:            example-prod
Selector:        app=nginx,env=prod
Type:            NodePort
IP:              10.96.28.176
Port:            <unset>  80/TCP
TargetPort:      80/TCP
NodePort:        <unset>  32410/TCP
Endpoints:       10.255.16.3:80,
                 10.255.16.4:80
```



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
```

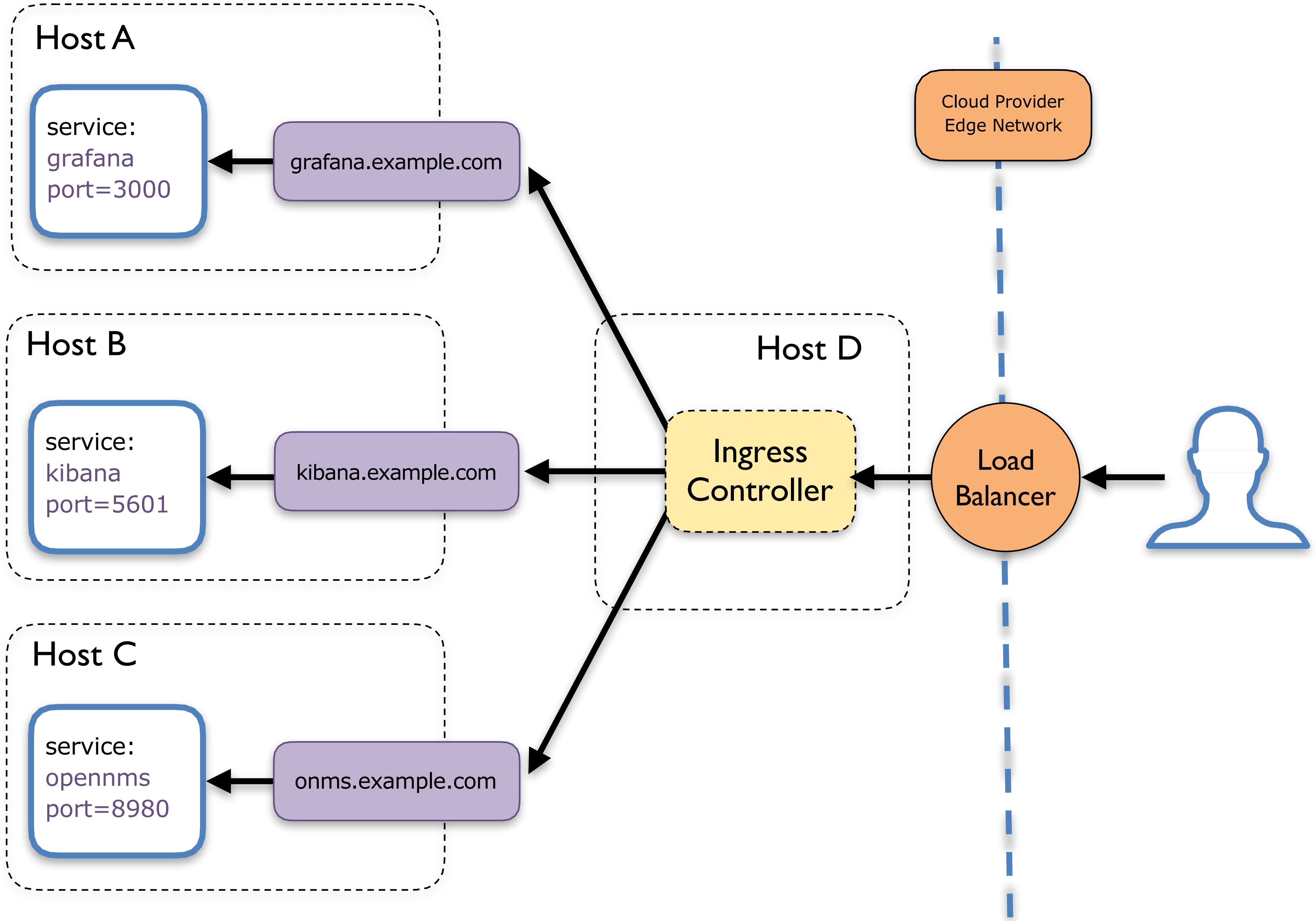


INGRESS

Known as Routes in OpenShift

- An API object that manages external access to the services in a cluster, through a single LoadBalancer.
- Provides load balancing, SSL termination and name/path-based virtual hosting
- Gives services externally-reachable URLs.
- Multiple implementations to choose from.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: ingress-rules
  annotations:
    kubernetes.io/ingress.class:  nginx
spec:
 rules:
     - host: onms.example.com
       http:
         paths:
           - path: /
             backend:
               serviceName: opennms
               servicePort: 8980
…
```



EXTERNAL NAME SERVICE

- **ExternalName** is used to reference endpoints **OUTSIDE** the cluster.
- Creates an internal **CNAME** DNS entry that aliases another.

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: ExternalName
spec:
  externalName: example.com
```

WORKLOADS

WORKLOADS

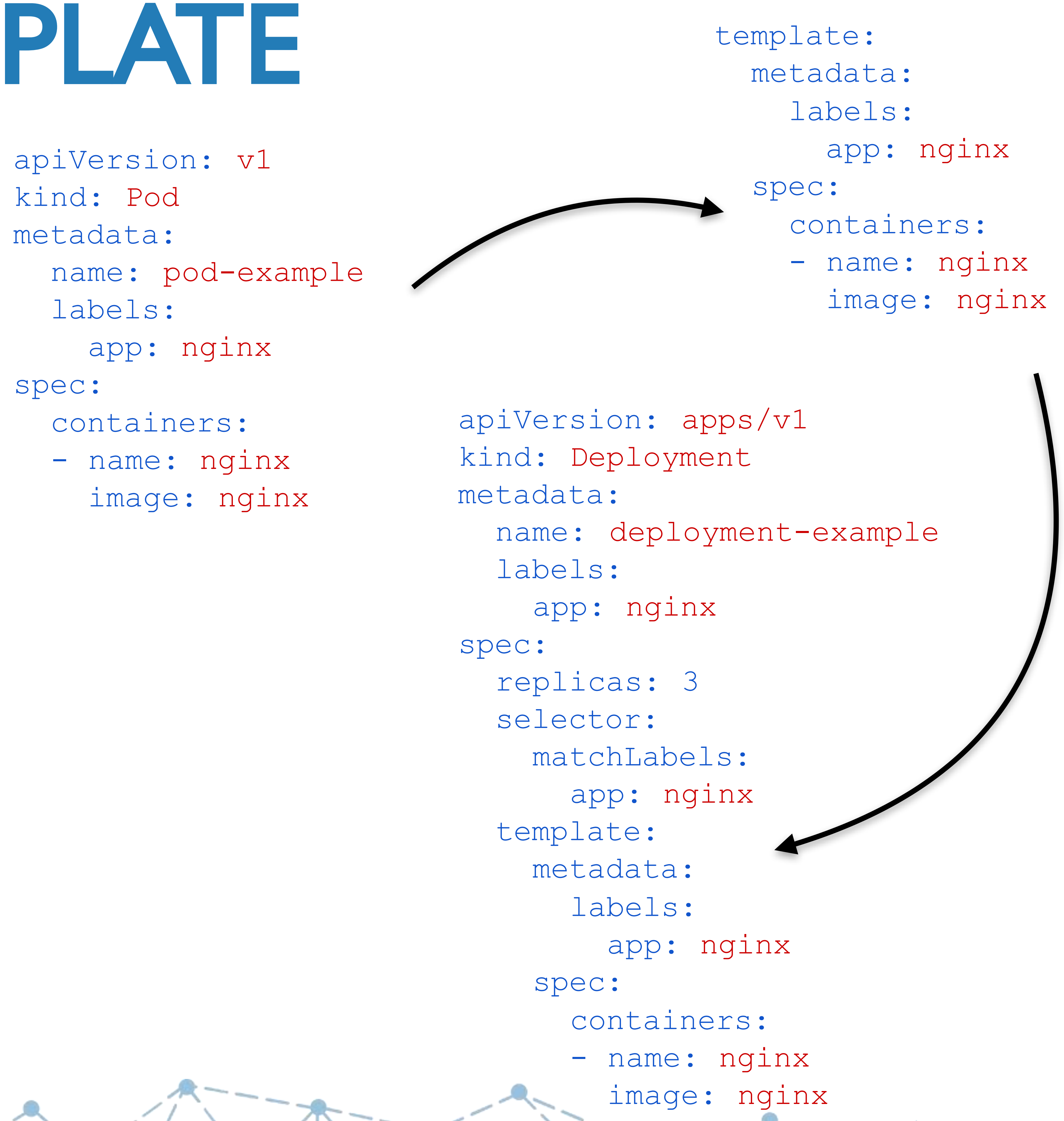
- Workloads within Kubernetes are higher level objects that manage Pods or other higher level objects.

- 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



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

```yaml
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
    - name: nginx
      image: nginx
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-example
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

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

```
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"
```

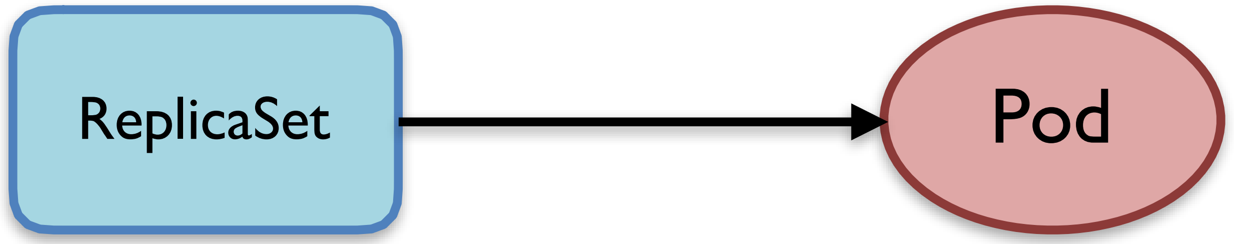
INIT CONTAINERS

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 that class.

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.**

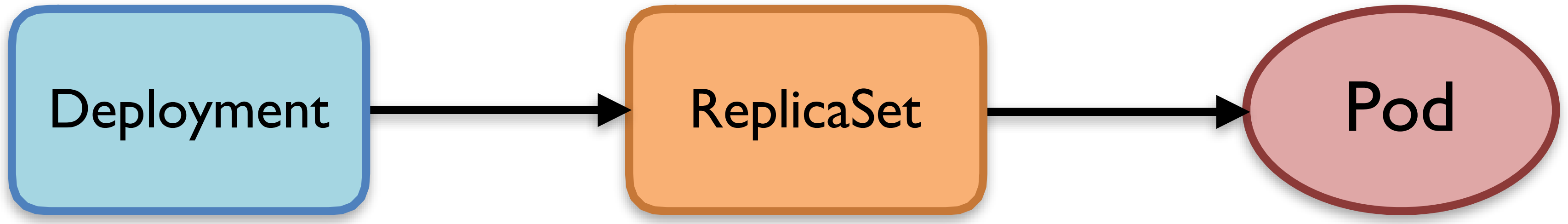


- **replicas**: The desired number of instances of the Pod.
- **selector**: The label selector for the **ReplicaSet** will manage **ALL** Pod instances that it targets; whether it's desired or not.

```yaml
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.



Deployment

ReplicaSet

Pod

- 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.

```yaml
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.

DaemonSet
Pod

- 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>
```

STATEFUL SET

- 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

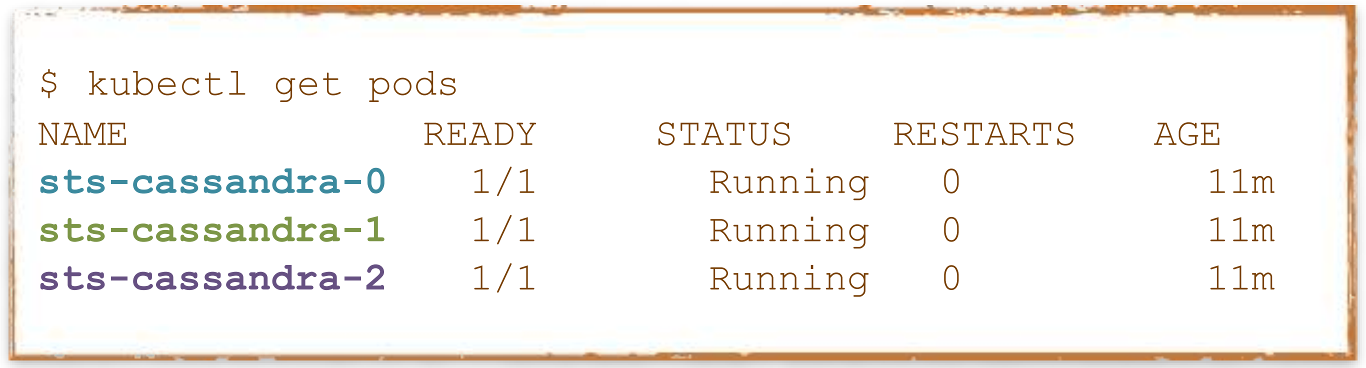
Template of the persistent volume(s) request to use for each instance of the StatefulSet.

StatefulSet
Pod

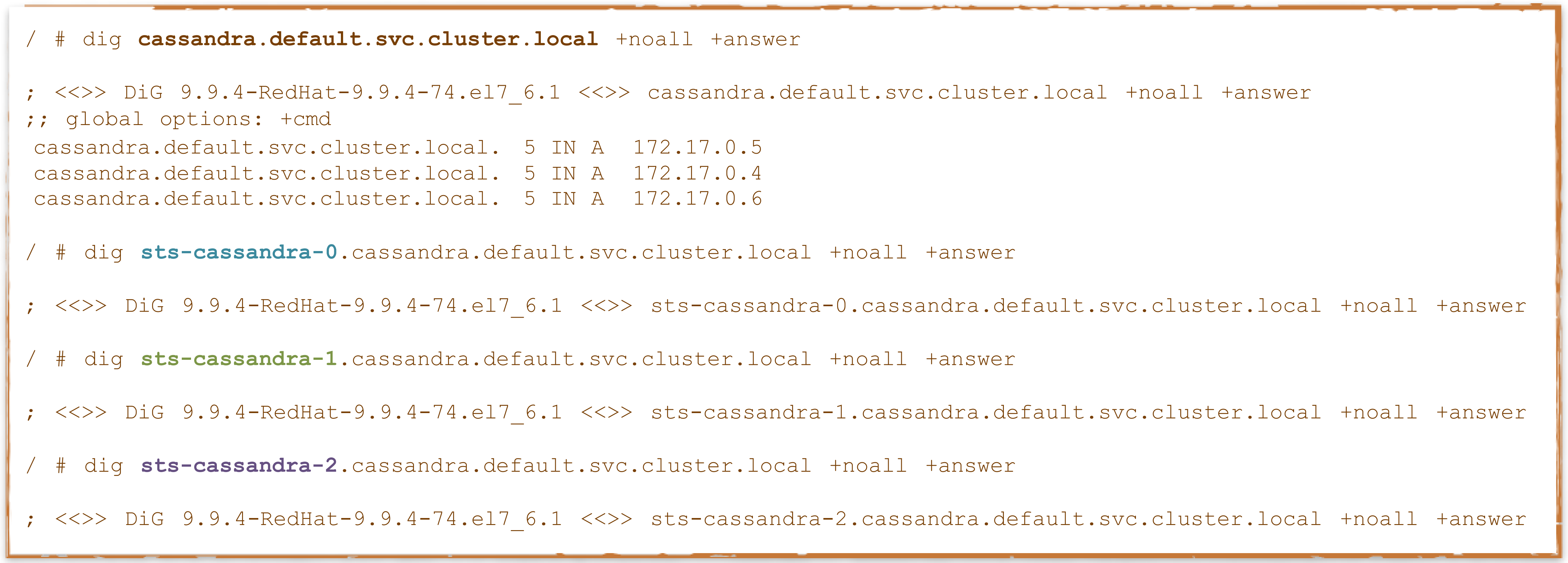
HEADLESS SERVICE

```
<StatefulSet Name>-<ordinal>.<service name>.<namespace>.svc.cluster.local
```

```
apiVersion: v1
kind: Service
metadata:
  name: cassandra
spec:
  clusterIP: None
  selector:
    app: cassandra
  ports:
  - name: intra-node
    port: 80
  - name: jmx
    port: 7199
  - name: cql
    port: 9042
```



```
$ kubectl get pods
NAME               READY     STATUS     RESTARTS    AGE
sts-cassandra-0     1/1        Running    0            11m
sts-cassandra-1     1/1        Running    0            11m
sts-cassandra-2     1/1        Running    0            11m
```



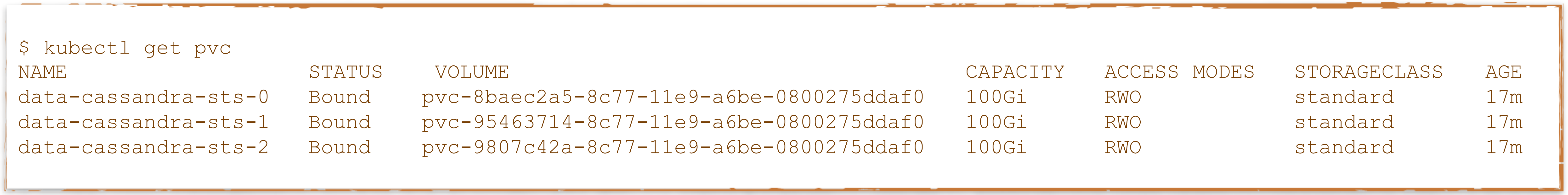
```
/ # dig cassandra.default.svc.cluster.local +noall +answer

; <<>> DiG 9.9.4-RedHat-9.9.4-74.el7_6.1 <<>> cassandra.default.svc.cluster.local +noall +answer
;; global options: +cmd
 cassandra.default.svc.cluster.local.  5 IN A  172.17.0.5
 cassandra.default.svc.cluster.local.  5 IN A  172.17.0.4
 cassandra.default.svc.cluster.local.  5 IN A  172.17.0.6

/ # dig sts-cassandra-0.cassandra.default.svc.cluster.local +noall +answer

; <<>> DiG 9.9.4-RedHat-9.9.4-74.el7_6.1 <<>> sts-cassandra-0.cassandra.default.svc.cluster.local +noall +answer

/ # dig sts-cassandra-1.cassandra.default.svc.cluster.local +noall +answer

; <<>> DiG 9.9.4-RedHat-9.9.4-74.el7_6.1 <<>> sts-cassandra-1.cassandra.default.svc.cluster.local +noall +answer

/ # dig sts-cassandra-2.cassandra.default.svc.cluster.local +noall +answer

; <<>> DiG 9.9.4-RedHat-9.9.4-74.el7_6.1 <<>> sts-cassandra-2.cassandra.default.svc.cluster.local +noall +answer
```

VOLUME CLAIM TEMPLATE

```
<Volume Name>.<StatefulSet Name>-<ordinal>
```

```
volumeClaimTemplates:
 - metadata:
     name: data
   spec:
     accessModes: [ "ReadWriteOnce" ]
     storageClassName: standard
     resources:
       requests:
         storage: 1Gi
```

Persistent Volumes associated with a StatefulSet will **NOT** be automatically garbage collected when its associated StatefulSet is deleted. They must manually be removed.



```
$ kubectl get pvc
NAME                   STATUS    VOLUME                                      CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-cassandra-sts-0   Bound    pvc-8baec2a5-8c77-11e9-a6be-0800275ddaf0   100Gi      RWO            standard       17m
data-cassandra-sts-1   Bound    pvc-95463714-8c77-11e9-a6be-0800275ddaf0   100Gi      RWO            standard       17m
data-cassandra-sts-2   Bound    pvc-9807c42a-8c77-11e9-a6be-0800275ddaf0   100Gi      RWO            standard       17m
```

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.

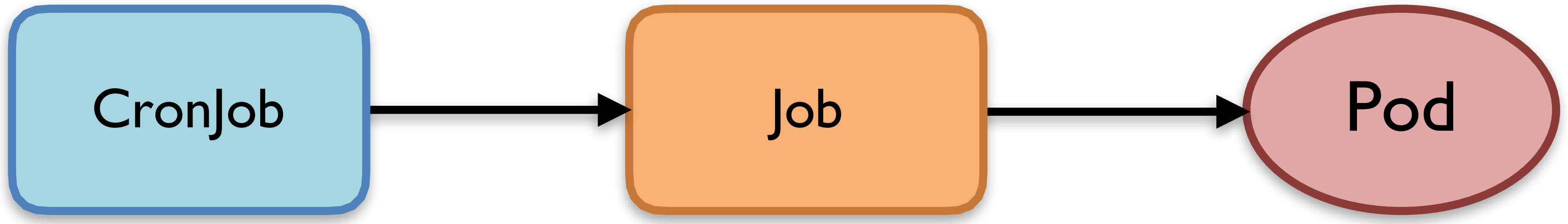
- **backoffLimit**: The number of failures before the job itself is considered failed.
- **completions**: The total number of successful completions desired.
- **parallelism**: How many instances of the pod can be run concurrently.
- **spec.template.spec.restartPolicy**:Jobs only support a restartPolicy 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.**



```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: cronjob-example
spec:
  schedule: "*/1 * * * *"
  successfulJobsHistoryLimit: 3
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      completions: 4
      parallelism: 2
      template:
        <pod template>
```

The number of successful jobs to retain

The cron schedule for the job

The number of failed jobs to retain

OPERATORS

- Might be the most powerful feature of K8s.
- When none of the workloads fits the needs of your app, you can create your own controllers with your own specifications (CRD).
- Should be implemented in Go, but there are alternatives.
- A given controller usually requires a ServiceAccount with proper RBAC Role to be able to manage workloads (e.x. create and maintain Pods, Services, etc.)

```
apiVersion: acid.zalan.do/v1
kind: postgresql
metadata:
  name: opennms-database
spec:
  teamId: OpenNMS
  volume:
    size: 100Gi
  numberOfInstances: 3
  enableMasterLoadBalancer: false
  enableReplicaLoadBalancer: false
  users:
    opennms:
    - superuser
  -   createdb
  databases:
    opennms: opennms
  postgresql:
    version: "10"
```

STORAGE

VOLUMES

- Storage that is tied to the **Pod’s Lifecycle**.
- A pod can have one or more types of volumes attached to it.
- Can be consumed by any of the containers within the pod.
- Survive Pod restarts; however their durability beyond that is dependent on the Volume Type.

PERSISTENT VOLUMES

- A **PersistentVolume** (PV) represents a storage resource.
- PVs are a **cluster wide resource** linked to a backing storage provider: NFS, EBS, GCEPersistentDisk, etc.
- Generally provisioned by an administrator.
- Their lifecycle is handled independently from a pod
- **CANNOT** be attached to a Pod directly. Relies on a **PersistentVolumeClaim** (PVC).
- A PVC is a **namespaced** request for storage.

STORAGE CLASS

- Satisfies a set of requirements instead of mapping to a storage resource directly.

- Ensures that an application’s claim for storage is portable across numerous backends or providers.

CONFIGURATION

CONFIG MAPS

- Externalized data stored within kubernetes.
- Can be referenced through several different means:
  - environment variable
  - a command line argument (via environment variable)
  - injected as a file into a volume mount
- Can be created from a manifest, literals, directories, or files directly.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: manifest-example
data:
  state: Minnesota
  city: Minneapolis
  content: |
    Look at this,
    its multiline!
```

SECRETS

- Functionally identical to a ConfigMap.
- Stored as **base64 encoded content**.
- Encrypted at rest within etcd (**if configured!**).
- Ideal for username/passwords, certificates or other sensitive information that should not be stored in a container.
- Can be created from a manifest, literals, directories, or from files directly.

- **type**: There are three different types of secrets within Kubernetes:
  - **docker-registry** - credentials used to authenticate to a container registry
  - **generic/Opaque** - literal values from different sources
  - **tls** - a certificate based secret
- **data**: Contains key-value pairs of base64 encoded content.

```
apiVersion: v1
kind: Secret
metadata:
  name: manifest-secret
type: Opaque
data:
  username: ZXhhbXBsZQ==
  password: bXlwYXNzd29yZA==
```

INSTALLATION

Install
OpenShift
is hard ;)

BARE-METAL, CLOUD, OR LOCAL INSTALLATION

Local PC Installation
For development and testing

- minikube
- minishift (for OKD)
- oc cluster up (for OKD)
- Docker for Mac or Windows
- microk8s (for Linux via Snap)

Cloud Installation
Single Command Experience

- gcloud container clusters create
- kops create cluster
- eksctl create cluster
- az aks create
- az openshift create

OpenShift 4 promises this for the first time since its conception.

Bare-Metal / On-Premise

- kubeadm (single command to spin up a master or a worker)
- OpenShift/OKD Ansible Tools

On-Premise vs Cloud Managed

- Masters are managed for you, and the only additional resources are the instances for worker nodes
- Complete integration with cloud services (volumes, network, load balancers, etc.)

Here is where OpenShift is Excellent!!!

SECURITY

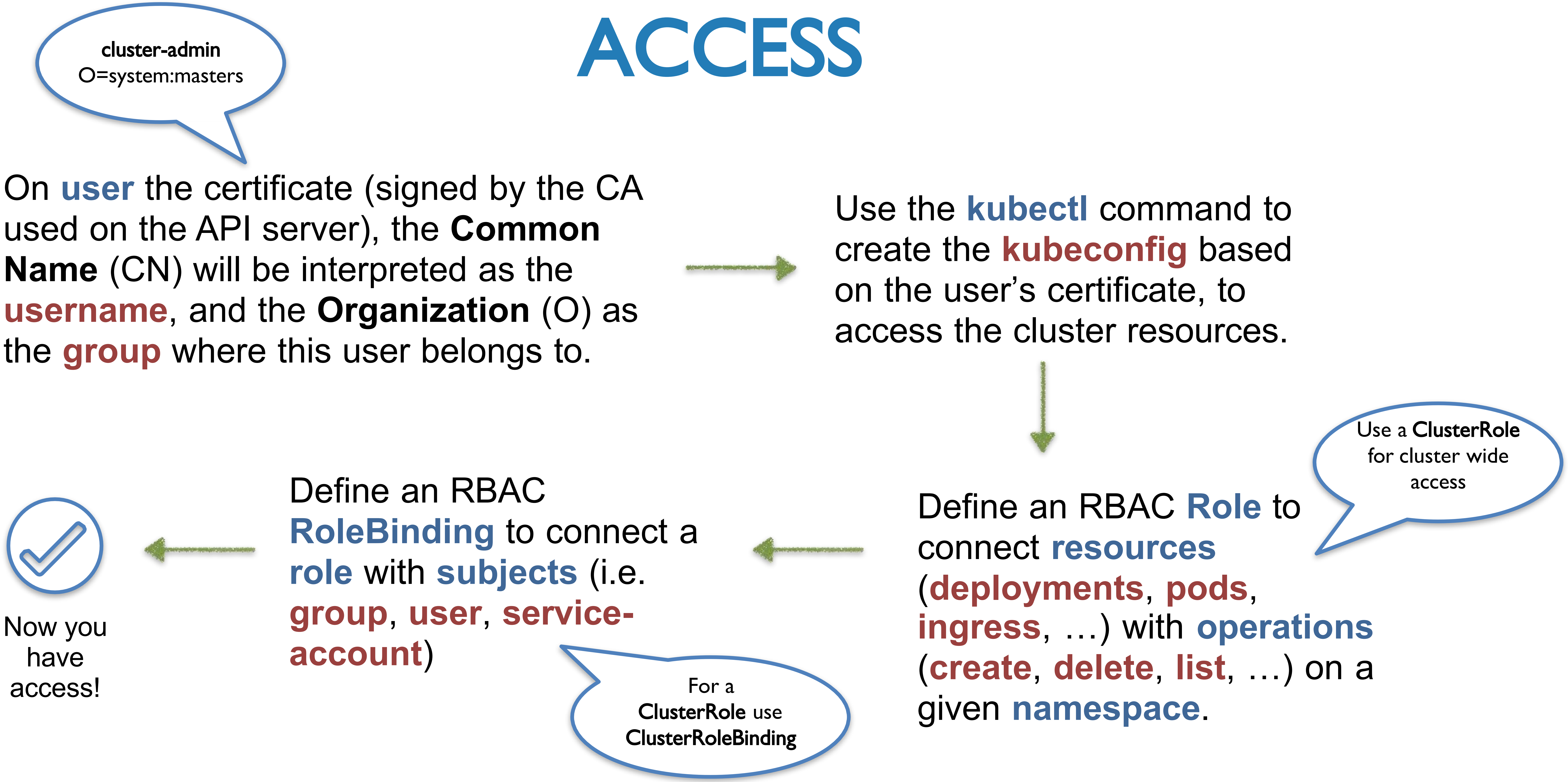
https://kubernetes.io/blog/2018/07/18/11-ways-not-to-get-hacked/

Some inspired by OpenShift

THINGS TO KEEP IN MIND

- Users (to access cluster through the API). Usually implemented through certificates.
- Service Accounts (for controllers and operators, to make changes from within K8s).
- RBAC: Roles and RoleBinding, ClusterRole and ClusterRoleBinding (for users, groups and service accounts).
- Never run containers as root, and block privilege escalations (OpenShift enforce this by default using SecurityContextConstraint; known I k8s as PodSecurityPolicy).
- Limit resources per namespace (total CPUs and memory, number of Pods, …).
- Create your own images and keep a private registry (OpenShift offers this by default).
- Research about NetworkPolicy (which requires a CNI that supports this feature).
- Research about applications like vault (to enhance protection for your secrets).
- Research about clair (a tool for vulnerability scans on containers).
- Research about kube-bench (a tool that analyzes your cluster for security best practices).
- Research about gvisor.dev (a container sandbox focused on security and efficiency).

ACCESS



cluster-admin
O=system:masters

On **user** the certificate (signed by the CA used on the API server), the **Common Name** (CN) will be interpreted as the **username**, and the **Organization** (O) as the **group** where this user belongs to.

Use the **kubectl** command to create the **kubeconfig** based on the user’s certificate, to access the cluster resources.

Define an RBAC **Role** to connect **resources** (**deployments**, **pods**, **ingress**, …) with **operations** (**create**, **delete**, **list**, …) on a given **namespace**.

Use a **ClusterRole** for cluster wide access

Define an RBAC **RoleBinding** to connect a **role** with **subjects** (i.e. **group**, **user**, **service-account**)

For a **ClusterRole** use **ClusterRoleBinding**

Now you have access!

MONITORING

HOW TO MONITOR K8S?

- Based on Google's Four Golden Signals: latency, errors, traffic, saturation, use Brendan Gregg's USE method (Utilization, Saturation, Errors) for Resources, and Tom Wilkie's RED method (Rate, Errors, Duration) for Services.

- Metric Server collects metrics such as CPU and Memory by each pod and node from the Summary API, exposed by Kubelet (via cAdvisor) on each node. This allows the usage of K8s features like the HorizontalPodAutoscaler.

- The usage of Prometheus is widely adopted. This solution is based on their own node_exporter (deployed as DaemonSet) which exposes lots of Linux metrics in Prometheus format.

DEMO

OVERWHELMED?

Do not forget we haven’t talked about …

- Several k8s features not discussed here ;)
- Helm (the k8s package manager; the yum or apt-get for k8s)
- Service Meshes (Envoy, Istio, Linkerd, …)
- Serverless (Kubeless, Fission, Knative, …)
- Machine Learning (kubeflow, …)
- VMs and Kubernetes (KubeVirt)
- Operators In Depth:
  https://operatorhub.io
  https://github.com/operator-framework

And more…

Not a big enough list ? Check https://landscape.cncf.io

QUESTIONS?

THANK YOU!

This work has been inspired by a few presentations from:
https://github.com/cncf/presentations/tree/master/kubernetes
and an OPENNMS presentation done by Alejandro Galue