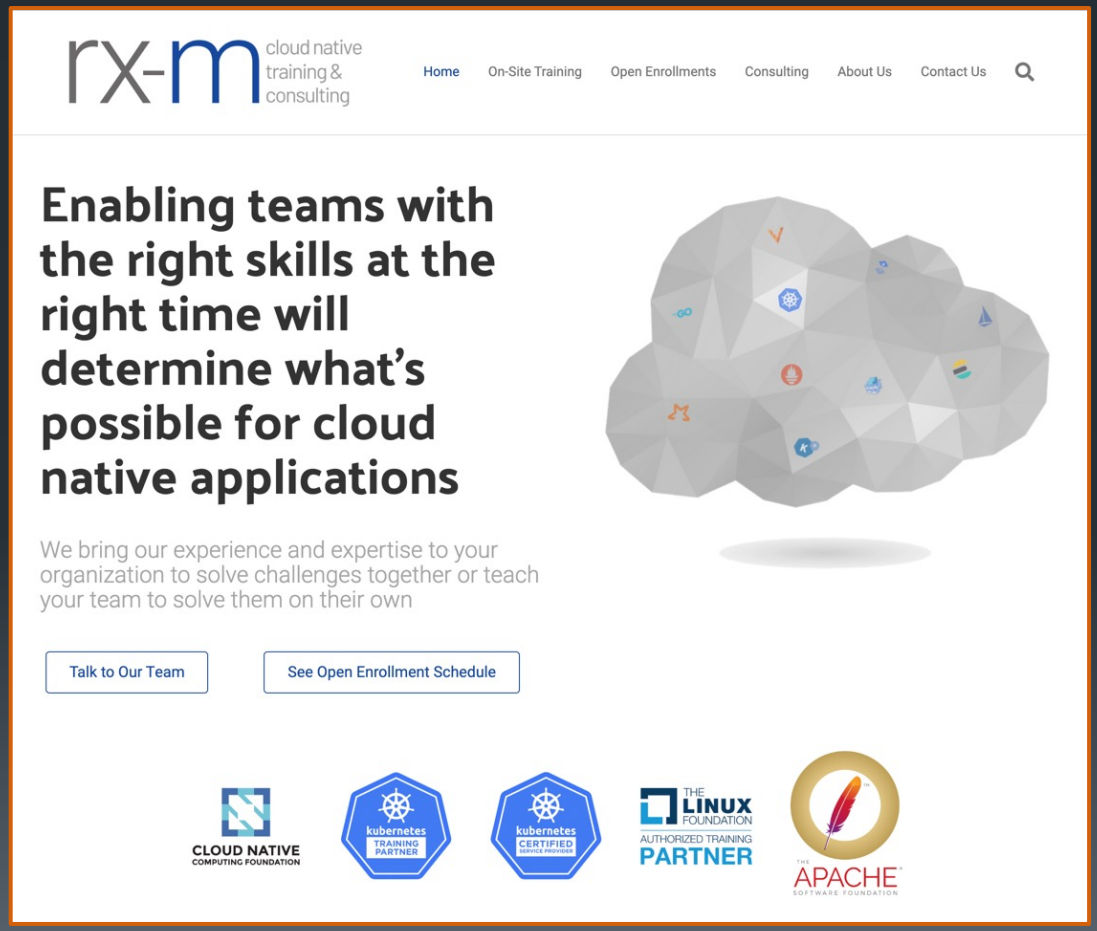
rx-m cloud native training & consulting

Kubernetes

Foundation

RX-M Cloud Native Advisory, Consulting and Training

- Microservice Oriented
  - Microservices Foundation [3 Day]
  - Building Microservices on Kubernetes [3 Day]
  - Building Microservices on AWS/GCP/Azure [3 Day]
  - Building Microservices with Go [3 Day]
  - Building Microservices with Thrift/gRPC [2 Day]
- Container Packaged
  - Docker Foundation [3 Day]
  - Docker Advanced [2 Day]
  - Container Technology [2 Day]
  - Containerd [2 Day]
  - Cri-O [2 Day]
- Dynamically Managed
  - Kubernetes Foundation [2 Day]
  - Kubernetes Advanced [3 Day]
  - Kubernetes CKA boot camp [5 Day]
  - Kubernetes CKAD boot camp [4 Day]
  - Kubernetes Day 2 operations [2 Day]
- Robustly Secured
  - Securing Kubernetes [2 Day]
  - Kubernetes CKS boot camp [5 Day]
  - Secure Development on Kubernetes [2 Day]
  - Zero Trust Architecture [2 Day]
  - SLSA - Securing the Software Supply Chain [3 Day]
  - Monitoring Kubernetes with Prometheus [2 Day]
- Intelligence Enabled
  - Machine Learning Foundation [3 Day]
  - Advanced Topics in Machine Learning [2 Day]
  - MLOps Foundation [2 Day]
  - Python for ML Engineers Foundation [3 Days]
  - ML/AI for Quants [3 Days]
  - LLM Foundation [2 Day]
  - Spark on K8s [2 Day]
  - Machine Learning on K8s [3 Day]
  - Ray Foundation [2 Day]
  - ML/AI on AWS/GCP/Azure [3 Day]



Overview

Day One

1. Kubernetes Orchestration
2. Pods: Basics
3. Pods: Architecture
4. Application Configuration

Day Two

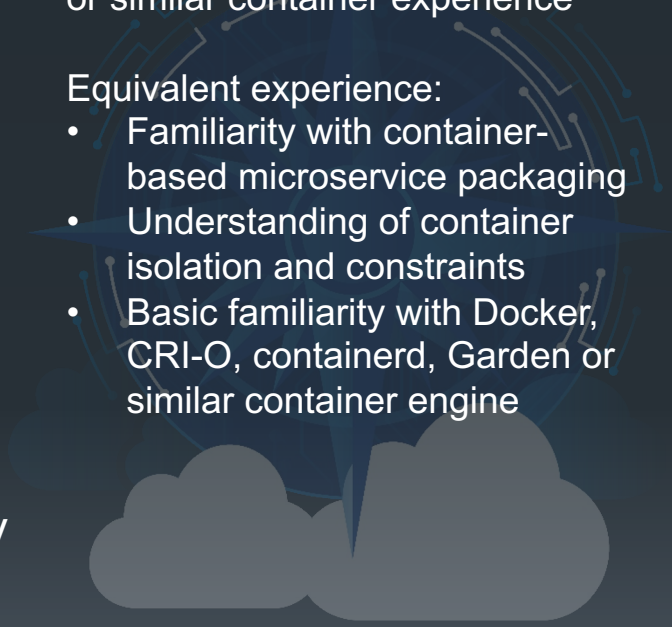
5. Application Controllers: Deployments
6. Services
7. Ingress Load Balancing & Network Policy
8. Stateful Workloads

Prerequisites:

RX-M Docker Foundation
or similar container experience

Equivalent experience:

- Familiarity with container-based microservice packaging
- Understanding of container isolation and constraints
- Basic familiarity with Docker, CRI-O, containerd, Garden or similar container engine

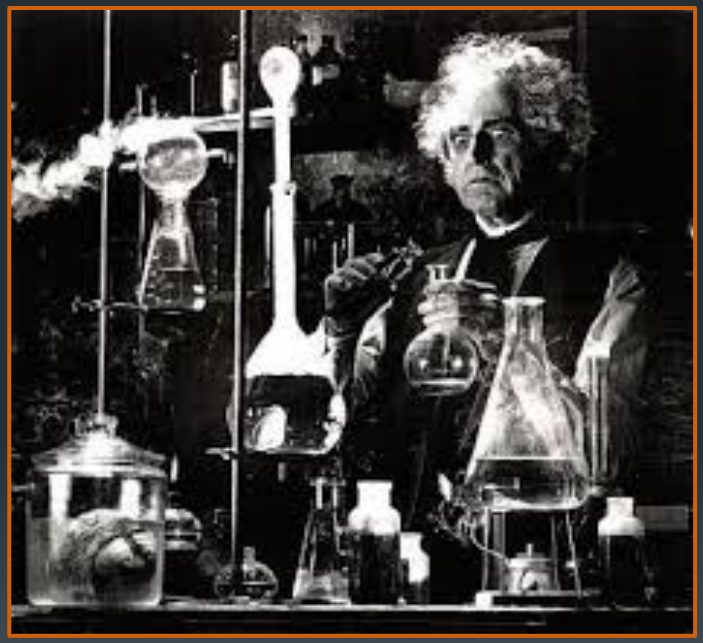


Administrative Info

- Length: 2 Days
- Format: Lecture/Lab/Discussion
- Schedule: 9:00AM – 5:00PM
  15 minute break, AM & PM
  1 hour lunch at noon
  Lab time at the end of each AM and PM session
- Location: Fire exits, Restrooms, Security, other matters
- Attendees: Name/Role/Experience/Goals for the Course
- Sign in: Sign in sheet
- Materials: Links in sign in sheet
- Lab Access: SSH instructions in sign in sheet

Lecture and Lab

- Our Goals in this class are two fold:
  1. Introduce concepts and ecosystems
     - Covering concepts and where things fit in the world is the primary purpose of the lecture/discussion sessions
     - The instructor will take you on a tour of the museum
       - Like a museum tour, you should interact with the instructor (tour guide), ask questions, discuss
       - Like a museum tour, you will not have time to read the slides during the tour, instead, the instructor will discuss and point out the highlights of the slides (exhibits) which will be waiting for you to read in depth later should you like to dig deeper
  2. Impart practical experience
     - This is the primary purpose of the labs
     - Classes rarely have time for complete real world projects so think of the labs as thought experiments
       - Like hands on exhibits at the museum



Day 1

1. Kubernetes Orchestration
2. Pods: Basics
3. Pods: Architecture
4. Application Configuration

1: Orchestration

Objectives

- Gain a high-level understanding of the Kubernetes container orchestration platform
- Describe the goals of the Kubernetes system
- Explore the Kubernetes architecture
- Discuss Kubernetes control plane components and node agents
- Examine Kubernetes-as-a-service solutions
- Discuss various techniques for installing kubernetes
- Understand how to configure and use the Kubernetes CLI: kubectl

After Containers, what's left?

- Orchestration!
  - Cloud native apps are delivered in 10s of images requiring 1000s of running containers
  - Registries manage image distribution
  - Orchestrators manage containers at scale
- Orchestration features:
  - Container Scheduling
    - Distributing containers to appropriate hosts
    - Host resource leveling
    - Availability zone diversity
    - Related container packaging and co-deployment
    - Affinity/Anti-affinity
  - Container Management
    - Monitoring and recovery
    - Image upgrade rollout
    - Scaling
    - Logging
  - Service Endpoints
    - Discovery/Name-Resolution
    - High Availability
    - Load Balancing
    - Traffic Routing
  - External Service Integration
    - Container Runtimes
      - CRI <--> OCI
    - Network configuration and management
      - CNI
    - Durable volume management
      - CSI

What is Kubernetes

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts
- Responds quickly to user demand:
  - Scaling applications on the fly
  - Seamlessly rollout new features
  - Optimized use of hardware using only the resources needed
- Kubernetes is:
  - lean: lightweight, simple, accessible
  - portable: public, private, hybrid, multi-cloud
  - extensible: modular, pluggable, hookable, composable
  - self-healing: auto-placement, auto-restart, auto-replication
- Commonly shortened to K8S (K at the beginning, S at the end, with 8 characters in between)
- Core components Kubernetes ships with:
  - A server that serves **the Kubernetes API**
  - A **base set of API resource types** allowing users to define various desired states
    - e.g. Run a container somewhere in a cluster of computers
  - **Software that consumes API resources** & changes a system per what is described

Kubernetes

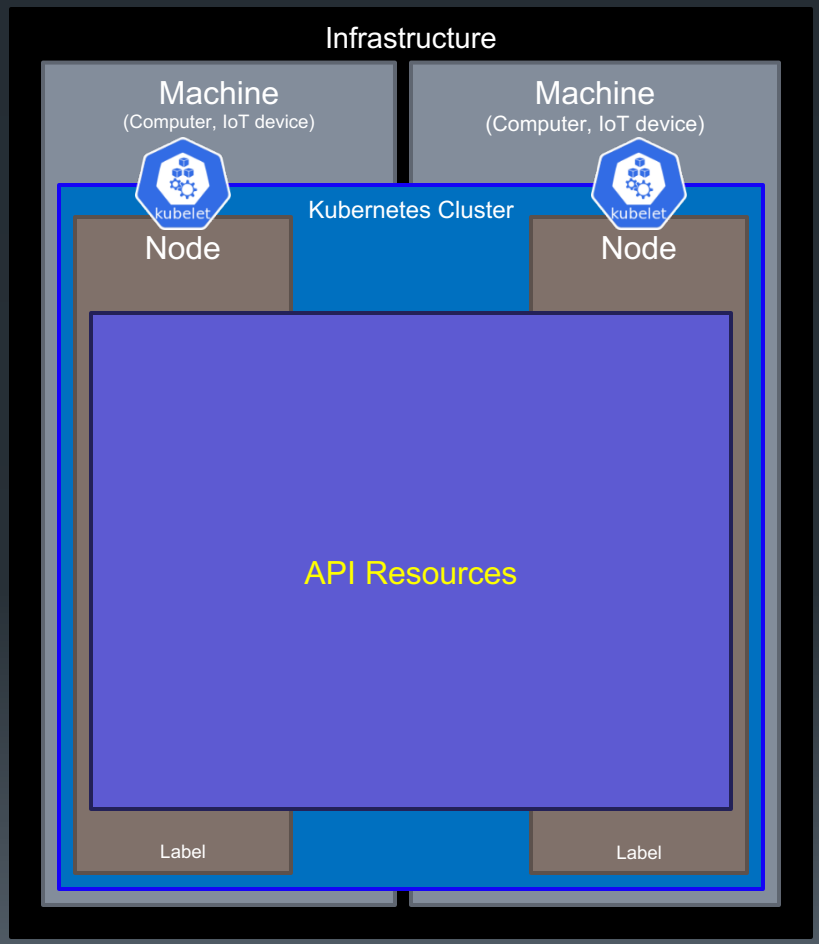
- Greek word (κυβερνήτης) meaning helmsman or captain
- Derived from the Greek word kubernan
- Derivations in other languages include the English cybernetics

- Kubernetes can be deployed directly on private systems in corporate data centers and on various cloud hosting providers, such as Google Compute Engine
- Under active development by many of the same engineers who built Borg

The Kubernetes project was started by Google in 2014
Kubernetes builds upon a decade and a half of experience that Google has with running production workloads at scale (Borg), combined with best-of-breed ideas and practices from the community

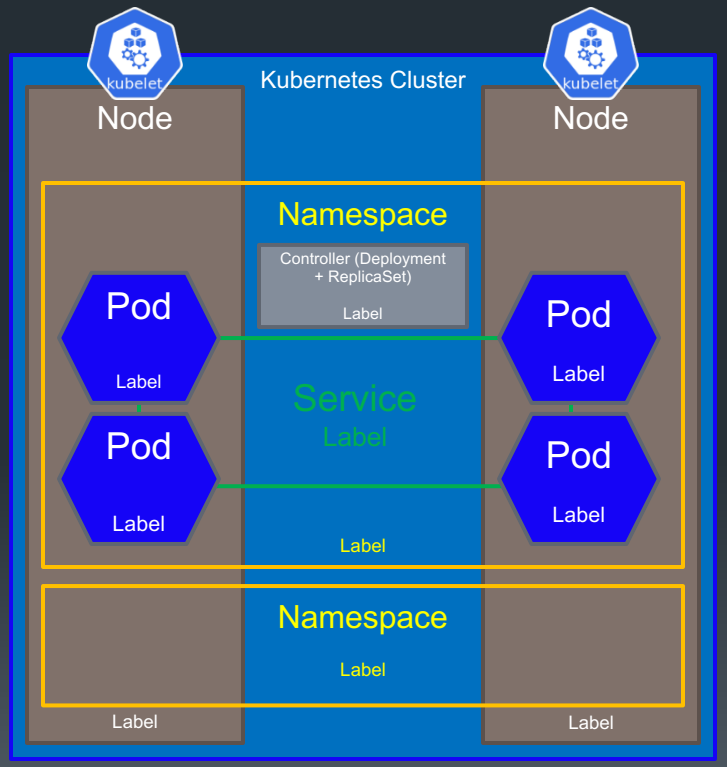
Kubernetes Concepts: Architectural

- Cluster
  - A cluster is a set of physical or virtual machines and other infrastructure resources used by Kubernetes to run your applications
  - Hosts a unique instance of the Kubernetes API
- Node
  - A node is a physical or virtual machine running the Kubernetes agent **`kubelet`**, onto which pods can be scheduled
- Namespace
  - A mechanism to partition resources created by users into a logically named group
  - Allows separate user communities to work in isolation
  - Manifests as separate endpoints in the Kubernetes API
  - Can be used as use, security, & network boundaries
    - Resource quotas & limit ranges to constraint resource use
    - Network Policies to enforce app-to-app communication
- API Resources
  - Specifications of Kubernetes constructs that will provide instructions on how to fulfill a desired state
- Label
  - Labels are optional key:value pairs stored in the metadata of Kubernetes resources used to organize and select groups of objects

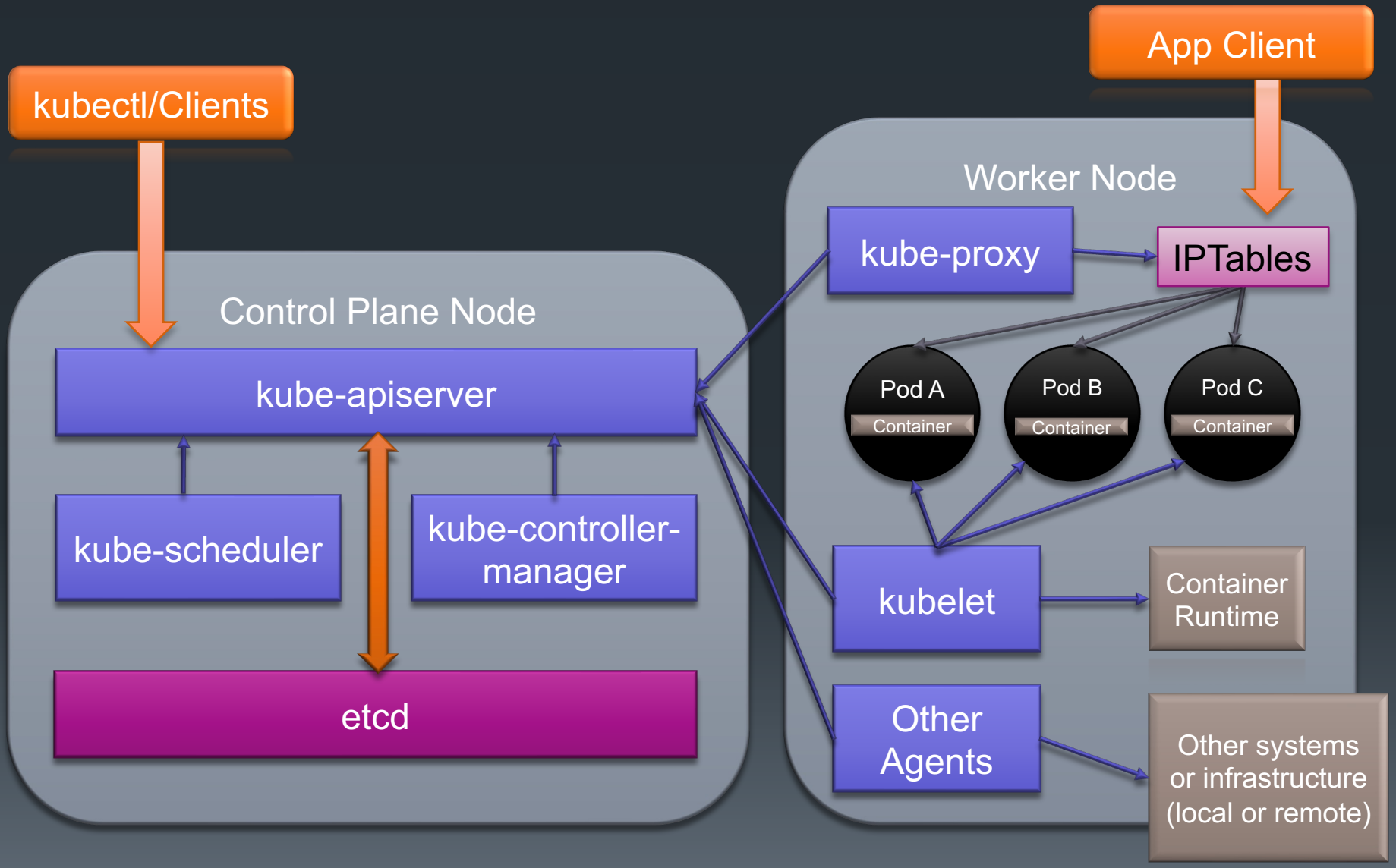


Kubernetes Concepts: Workload-related API Resources

- Pod
  - Pods are co-located groups of application containers
  - The smallest deployable units that can be created, scheduled, and managed with Kubernetes
  - Pods can be created individually, but it's recommended that you use a controller even if creating a single pod
- Controllers
  - Controllers ensure the running state of the cluster reflects the user's specified state
    - e.g. "Run n pods in the cluster"
  - Run control loops that watch for and reconcile differences between the cluster's current state and user's desired state
  - Separated into many types, including (but not limited to):
    - Replica Set
      - Replica Sets manage the lifecycle of pods
      - They ensure that a specified number of pods are running at any given time, by creating or deleting pods as required
    - Deployment
      - Allow initial deployment of pods through replica sets and rolling upgrades
- Service
  - Services provide a single, stable network identity (name and address) for a set of pods
  - Act as basic load balancers

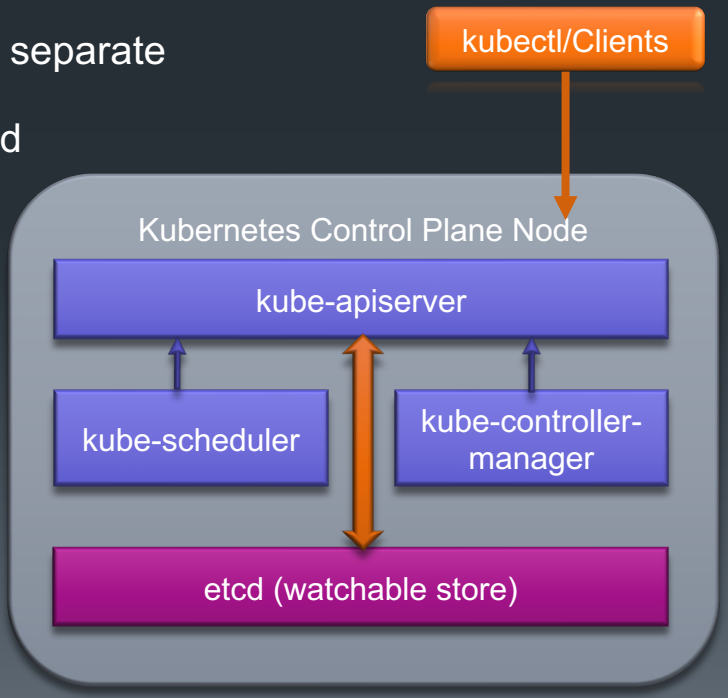


Kubernetes Architecture



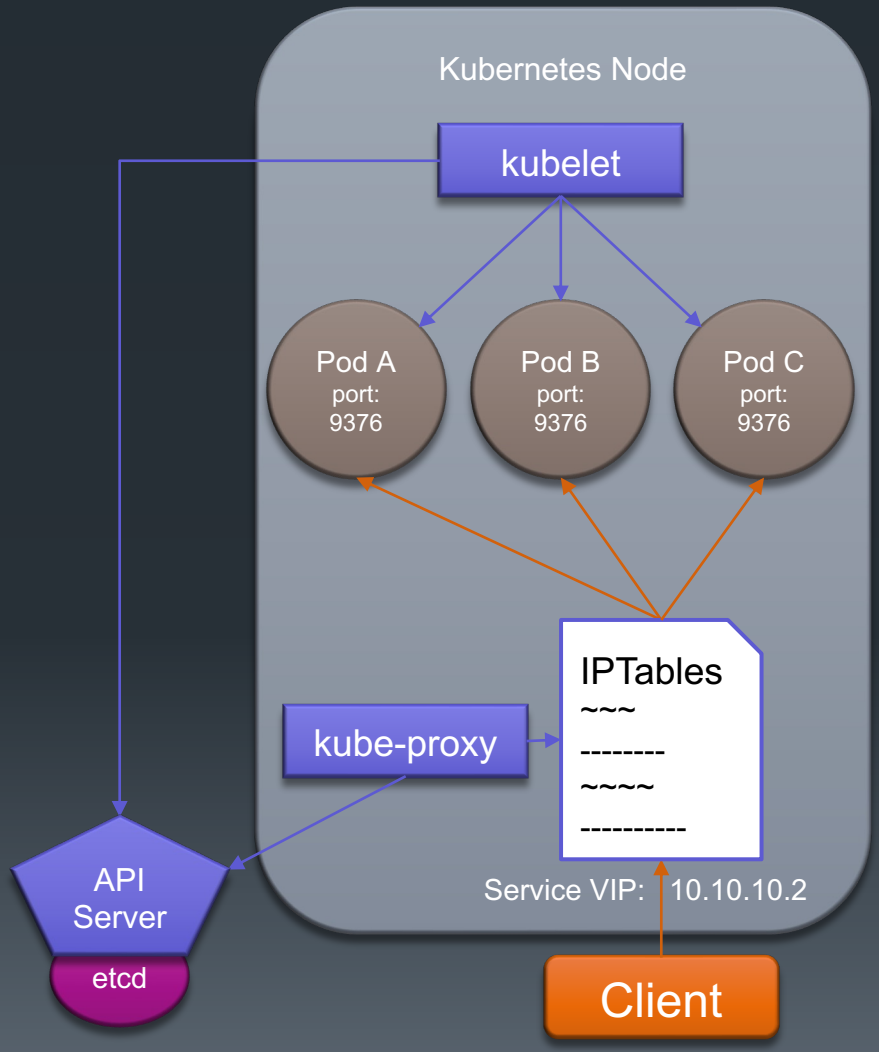
Control Plane

- The Kubernetes Control Plane
  - The Kubernetes control plane is split into a set of microservices
  - These components work together to provide a unified view of the cluster
- etcd
  - All persistent state is stored in an etcd cluster
    - Stores active copies of API resource specifications
    - If it exists in etcd, the cluster will maintain it
  - Watch support allows coordinating components to be notified of changes
- API Server
  - Serves the Kubernetes API
  - A CRUD-y server, most/all business logic implemented in separate components or plug-ins
  - Processes REST operations, validates them, updates etcd
- Scheduler
  - Binds unscheduled pods to nodes via the /binding API
  - The scheduler is customizable and pluggable
- Controller Manager
  - Other cluster-level functions are performed by the Controller Manager
    - Endpoints sync with services
    - Node discovery and management
    - Controller state management (ReplicaSets, Autoscalers, etc.)



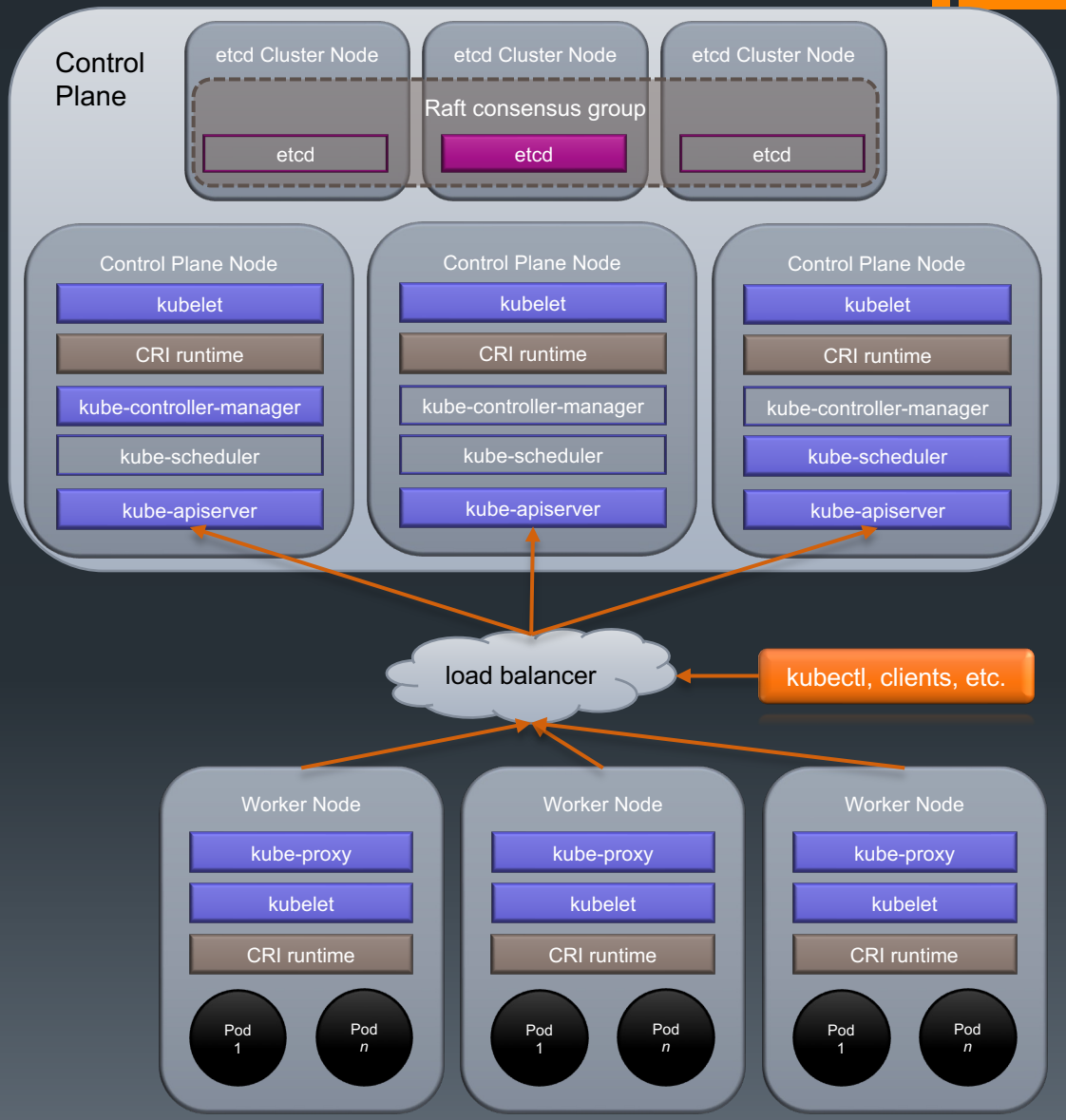
Node Agents

- kubelet
  - The kubelet is the primary “node agent” that runs on each node as a system service
  - The kubelet manages pods and their containers, their images, their volumes, etc.
    - Kubelet contacts the container runtime to create or delete Pod containers
    - The container runtime then assigns Pods IP addresses according to the CNI plugin in use
  - The kubelet consumes Pod specifications (PodSpec) from the API
    - A PodSpec is a YAML or JSON object that describes a pod
    - kubelet PodSpecs are provided through various mechanisms (primarily through the apiserver)
    - Kubelet ensures that the containers described in PodSpecs are running and healthy
  - Responsible for other node agents that run as pods
- kube-proxy
  - The Kubernetes network proxy runs on each node as a pod
  - Manages the iptables service mesh for services defined in the Kubernetes API
    - Ensures that pods can communicate with other pods in the cluster on any node
    - Also ensures that services correctly route to their endpoint pods
  - Can do simple TCP/UDP stream forwarding or round robin TCP/UDP forwarding across a set of backends in proxy mode
- More node agents may be deployed as pods to support additional features
  - e.g. Virtual Routers, Storage Attachers, Metrics/Log Agents



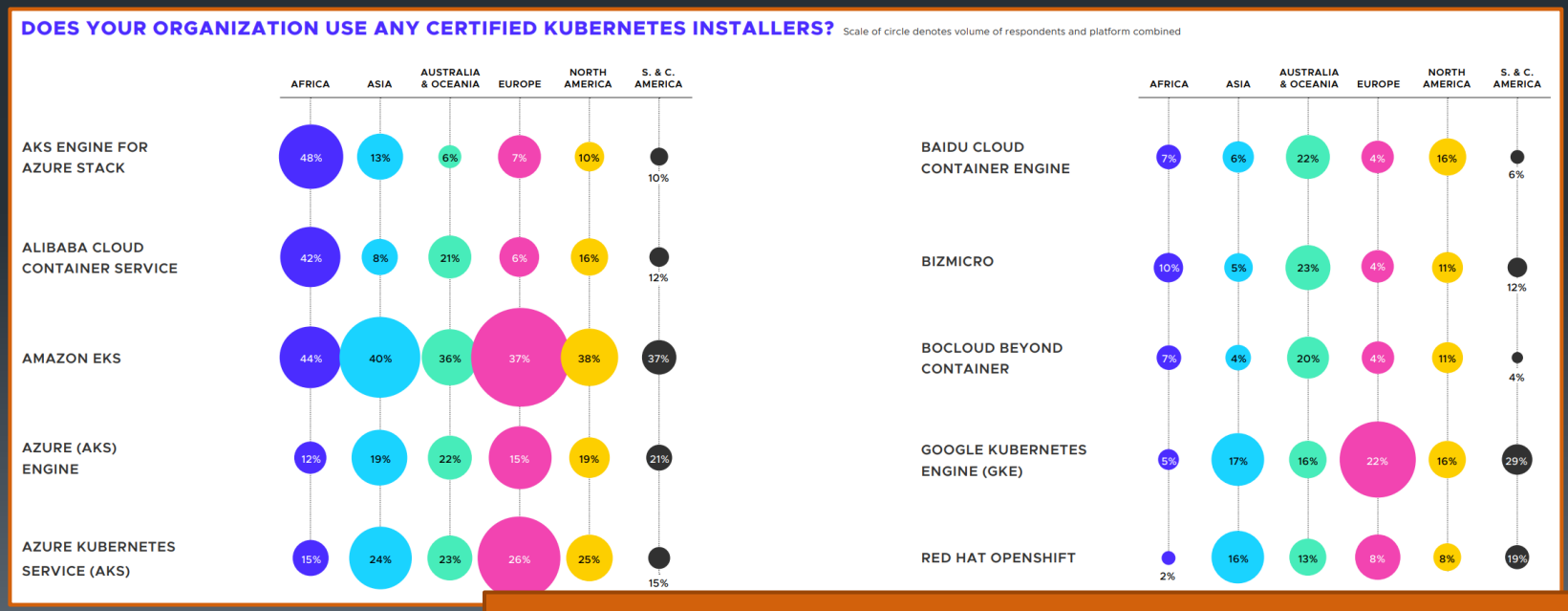
Scaled Architecture & HA Model

- Control Plane components manage the cluster and can be co-located or spread across machines
  - API Server is the central cluster state manager and event distributor
  - Scheduler and controller manager perform their own independent leader elections
    - These components change cluster state so only one instance of each should be active at any given time to prevent conflicting changes
    - Members elect new leader if current leader fails
    - The current leader holds a lease object stored in etcd in the kube-system namespace to indicate who to submit control requests to
- etcd is the distributed watchable key/value store used by the apiServer to store all cluster metadata
- Nodes run services that support pods
  - kubelet manages pods and their containers using plugins:
    - CRI: Container Runtime Interface implemented by a CRI/OCI container manager which downloads images and runs containers [required]
    - CNI: Container Network Interface implemented by a pod networking agent [required]
    - CSI: Container Storage Interface implemented by zero or more storage plugins [optional]
  - kube-proxy configures the pod service mesh



Container Orchestration Platform Adoption

- Kubernetes adoption has grown substantially over the past 3 years
- Some estimate 96% of enterprises use or are evaluating Kubernetes as of 2021
- Self-managed Kubernetes usage appears to be dropping on a percentage basis while aaS Kubernetes solutions appear to be growing
  - 79% of organizations are using some kind of certified installer



https://www.cncf.io/reports/cncf-annual-survey-2021/

K8s as a Service Solutions

- All major clouds presently offer a fairly mature Kubernetes as a Service solution
  - Open cloud console, click some buttons, use Kubernetes
- Certified Kubernetes
  - https://www.cncf.io/certification/software-conformance/
  - Software conformance ensures that every vendor's version of Kubernetes supports the required APIs, as do open source community versions
  - For organizations using Kubernetes, conformance enables interoperability from one Kubernetes installation to the next
  - CNCF runs the Certified Kubernetes Conformance Program
  - Most of the world's leading enterprise software vendors and cloud computing providers have Certified Kubernetes offerings
- KaaS certified offerings:
  - AKS – Microsoft Azure Kubernetes Service
  - EKS – Amazon EKS - Managed Kubernetes Service
  - GKE – Google Kubernetes Engine
  - IKE – IBM Cloud Kubernetes Service
  - OCE – Oracle Container Engine
  - PKS – VMware Cloud PKS
  - PMK – Platform9 Managed Kubernetes
  - RKS – Rackspace Kubernetes as a Service
  - RHOSD – RedHat OpenShift Dedicated
  - Many others

Installers

- Test/Dev Cluster
  - Minikube – the recommended method for creating a single node cluster for testing & development
  - Docker Desktop – single-node K8s cluster for development
  - Minishift – community version of OpenShift for Windows, macOS, Linux
  - MicroK8s – single-command fast install (~30 seconds) w/ plugin support
  - KinD – Kubernetes-in-Docker multi-node container-based cluster
  - Ubuntu on LXD – supports a 9-instance deployment on localhost via LXC
- Single Node Reference Installer
  - kubeadm – the reference installer for Kubernetes
- Multi Node Installers
  - kops – AWS GA, GCE/OpenStack Beta, vSphere Alpha
  - Kubespray – Ansible playbooks to install k8s on GCE, Azure, OpenStack, AWS, or baremetal (uses kubeadm)
  - Cloud Foundry Container Runtime (Kubo)
    - BOSH-based K8s installer

kubeadm

- Works with VMs, physical servers and cloud servers
- Designed to be part of a large provisioning system (used by kops for example)
- Also for easy manual provisioning
- `kubeadm init` bootstraps a K8s cluster:
  - Runs a series of pre-flight checks to validate the system state prior to making changes
  - Generates a token that additional nodes can use to register themselves with the Control Plane Node
  - Generates a self-signed CA using openssl to provision identities for each node in the cluster
    - Also used by the API server to secure communication with clients
  - Outputs a kubeconfig file for the kubelet to use to connect to the API server
    - Creates an additional kubeconfig file for administration
  - Generates Kubernetes resource manifests on the Control Plane Node in `/etc/kubernetes/manifests` for etcd, api-server, controller manager and scheduler
  - Installs any add-on components, such as DNS and kube-proxy
- Configuring kubeadm with a configuration file instead of command line flags is *recommended*
  - Some advanced features (such as using an external etcd cluster) only available as configuration file options

```
# kubeadm init --help
Run this command in order to set up the Kubernetes control plane

The "init" command executes the following phases:
```
preflight   Run pre-flight checks
certs   Certificate generation
...
kubeconfig   Generate all kubeconfig files necessary to e
...
kubelet-start   Write kubelet settings and (re)start the kub
control-plane   Generate all static Pod manifest files neces
...
etcd   Generate static Pod manifest file for local
...
upload-config   Upload the kubeadm and kubelet configuration
...
upload-certs   Upload certificates to kubeadm-certs
mark-control-plane   Mark a node as a control-plane
bootstrap-token   Generates bootstrap tokens used to join a node to a cluster
kubelet-finalize   Updates settings relevant to the kubelet after TLS bootstrap
...
addon   Install required addons for passing conformance tests
```

**`kubeadm join`**

- Uses the token to talk to the API server and securely get the root CA certificate
- Creates a local key pair; prepares a certificate signing request (CSR) and sends that off to the API server for signing
- Configures the local kubelet to connect to the API server
- Using the `--control-plane` flag enables joining the node as a Control Plane Node (Beta as of 1.15)
```

kubectl

- CLI client for Kubernetes
  - Submits requests to the Kubernetes API Server
  - Request contents can be formulated from the command line (imperatively) or using files (declaratively)
- Syntax
  - `kubectl [command] [TYPE] [NAME] [flags]`
    - `command` – operation to perform on the named resource(s)
    - `TYPE` – specifies the resource type
      - Specify singular, plural or abbreviated forms (`pod`, `pods`, `po`)
    - `NAME` – specifies the name of the resource
    - `flags` – optional flags: (`-o name` for just names, `-o yaml` for yaml output, etc.)
- Basic Resource Commands:
  - `get` – display one or many resources
  - `describe` – show details of a specific resource or group of resources
  - `create` – create a resource by filename or stdin
  - `apply` – apply a configuration change to a resource from a file or stdin
  - `delete` – delete resources by filenames, stdin, resources and names, or by resources and label selector
  - `logs` – print the logs for a container in a pod, (`-p`) for previous container
  - `attach` – attach to a running container
  - `exec` – execute a command in a container
  - `explain` – get documentation of resources
  - `api-resources` – list of all the supported resource types and their abbreviated aliases
- Autocompletion support
  - Add it to your current shell
    - `source <(kubectl completion bash)`
  - To add to your profile so it is automatically loaded in future shells :
    - `echo "source <(kubectl completion bash)" >> ~/.bashrc`

More commands: https://kubernetes.io/docs/reference/kubectl/overview/#operations

Summary

- Kubernetes is a system for managing containerized applications across multiple hosts
- Kubernetes is itself a microservice based system and some or all of it may be executed via containers
- Kubernetes-as-a-service offerings such as EKS, GKE, and others are gaining popularity
- Kubernetes can be installed easily for testing and development using the Kubernetes reference installer: kubeadm
- kubectl offers a number of commands to work with resources:
  - create
  - delete
  - describe
  - get
  - logs

Lab 1

- Kubernetes cluster setup with kubeadm on your lab VM

2: Pods: Basics

Objectives

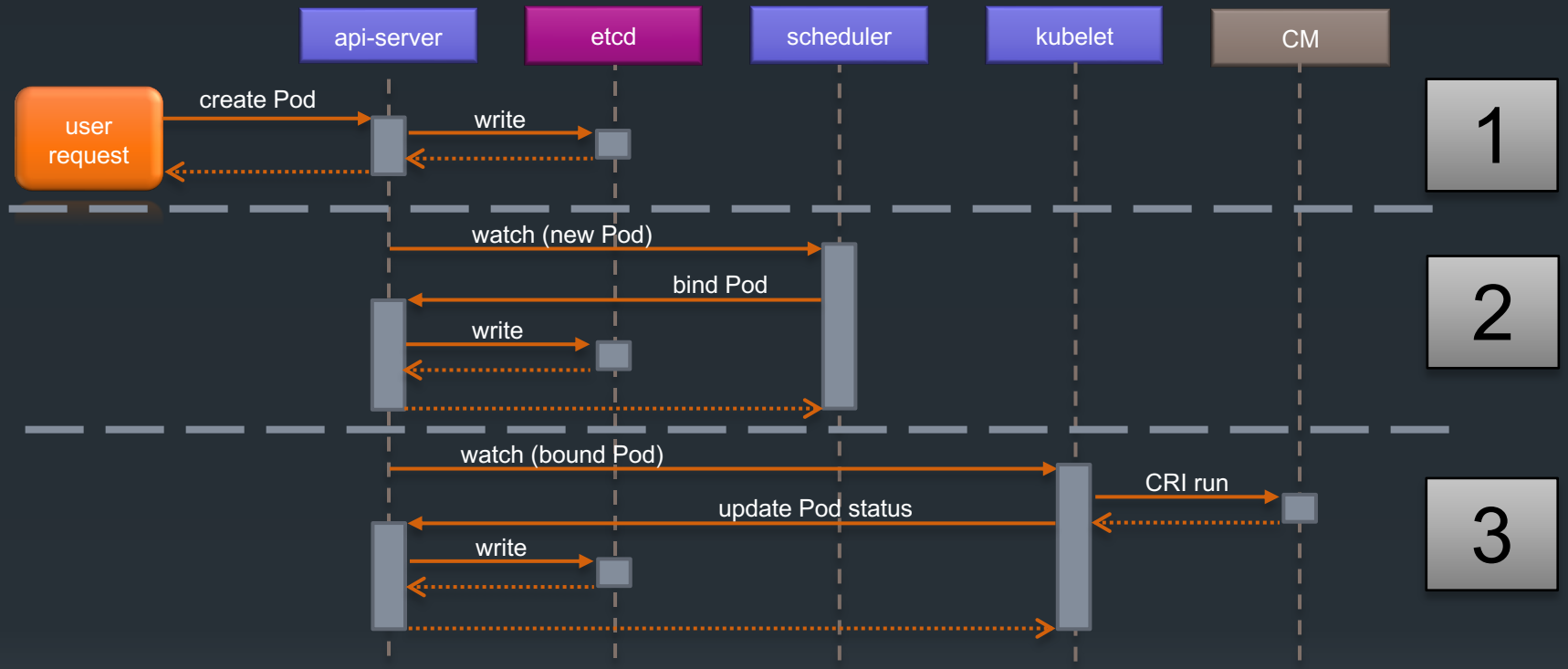
- Define Pods and the operation of Pods on Kubernetes
- Examine the interactions between Kubernetes agents during the creation and deployment of an application pod
- Learn how to define resources with YAML files
- Understand the relationship between images, containers and Pods

Running Workloads

- Running workloads in Kubernetes is a matter of defining a desired setup (state) using API Resources
  - Users specify API Resources declaratively using YAML or JSON specifications
    - a.k.a. manifests or "specs"
  - Specs are sent to the API Server for the cluster to fulfill
- e.g. To run containers, users must define pods
  - Pods are atomic resources that define & place containers in a Kubernetes cluster
- `kubectl` supports basic API Object management management:
  - Creating
    - Nonidempotent: `create -f manifest.yaml`
    - Idempotent: `apply -f manifest.yaml`
  - Listing – `get pod`
  - Deleting – `delete pod xxx`
- Many other kubectl commands and options are available
  - `describe` – show object details
  - `get -o yaml` – display the yaml spec for a pod(s)
  - `logs` – display the stdout/err output of the container in the pod
  - `exec` – execute a program inside the pod's container

```
~$ cat pod.yaml

apiVersion: v1
kind: Pod
metadata:
  name: hello
spec:
  restartPolicy: Never
  containers:
  - command:
    - echo
    - hello kubernetes
    image: busybox
    name: hi

~$ kubectl apply -f pod.yaml

pod/hello created

~$ kubectl get pod

NAME    READY   STATUS      RESTARTS   AGE
hello   0/1     Completed   0          7s

~$ kubectl logs hello

hello kubernetes

~$ kubectl delete pod hello

pod "hello" deleted
```

Pod Creation Flow



1. user creates a Pod via the API Server and the API server writes it to etcd
2. api-server sends an asynchronous event to the scheduler in response to the scheduler's "watch" for unbound pods
   - scheduler then binds the pod to an appropriate node and saves the binding to the api-server
3. api-server sends an asynchronous event to the Kubelet bound to the pod
   - kubelet runs the container via the container manager
   - kubelet updates the status of the pod to "Running" in the API Server

Specification Commonalities

- YAML formatted
- Main parts
  - # Comment
  - apiVersion
    - API Group where a resource type exists
  - kind
    - Kubernetes object type
    - Pod, Deployment, Service, Namespace, etc…
  - metadata
    - Name, labels and other data describing the object
  - spec, data, etc.
    - Instructions for building the object
    - The desired state
    - Field varies depending on what is important to the kind of object being created
- Construction
  - `kubectl apply -f filename`

```
# mypod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  containers:
  - name: example-pod
    image: httpd:latest
```

Imperative command equivalent example:

```
kubectl run example-pod \
--image=httpd:latest
```

apiVersions

- Resource groups that organize resource types
- Groups – 3 apiVersion group forms evolved over time
  - v1 – used in the original Kubernetes release without a group name, all GA resources existed in a single group, now referred to as "core"
  - apps/v1 – used in Kubernetes 1.1 and later, resources types are now created in groups, like "apps" or "batch", usually aligned with a SIG
  - storage.k8s.io/v1 – used in current Kubernetes releases, migrating group names to a dotted domain name form under k8s.io, such as: "storage.k8s.io" or "networking.k8s.io" or "rbac.authorization.k8s.io"
- Kubernetes clusters support multiple API versions
  - Kubernetes versions and API versions are indirectly related
- Maturity - API resources come in three levels of maturity
  - Alpha – typically disabled by default, used for features in early stage development
    - apiVersions like: v1alpha1, batch/v1alpha2 and snapshot.storage.k8s.io/v1alpha1
  - Beta – typically enabled by default, used with resources having fairly stable apis (specs) that are still being debugged, tuned, etc.
    - apiVersions like: v1beta1, batch/v1beta3, storage.k8s.io/v1beta1
  - Stable
    - apiVersions like: v1, batch/v1, storage.k8s.io/v1

```
~$ kubectl api-versions
admissionregistration.k8s.io/v1
apiextensions.k8s.io/v1
apiregistration.k8s.io/v1
apps/v1
authentication.k8s.io/v1
authorization.k8s.io/v1
autoscaling/v1
autoscaling/v2
autoscaling/v2beta1
autoscaling/v2beta2
batch/v1
batch/v1beta1
certificates.k8s.io/v1
coordination.k8s.io/v1
discovery.k8s.io/v1
discovery.k8s.io/v1beta1
events.k8s.io/v1
events.k8s.io/v1beta1
flowcontrol.apiserver.k8s.io/v1beta1
flowcontrol.apiserver.k8s.io/v1beta2
metrics.k8s.io/v1beta1
networking.k8s.io/v1
node.k8s.io/v1
node.k8s.io/v1beta1
policy/v1
policy/v1beta1
rbac.authorization.k8s.io/v1
scheduling.k8s.io/v1
storage.k8s.io/v1
storage.k8s.io/v1beta1
v1
~$
```

API Resources

- Prints out supported resources and associated shortnames, groups, and kinds
- Adding `-o wide` shows supported verbs
- Filter by namespaced / non-namespaced: `--namespaced=[true|false]`
- Filter by specific group: `--api-group=<group>ß`

```
~$ kubectl api-resources -o wide
NAME                              SHORTNAMES   APIVERSION                             NAMESPACED   KIND
bindings                                       v1                                     true         Binding
componentstatuses                 cs           v1                                     false        ComponentStatus
configmaps                        cm           v1                                     true         ConfigMap
endpoints                         ep           v1                                     true         Endpoints
events                            ev           v1                                     true         Event
limitranges                       limits       v1                                     true         LimitRange
namespaces                        ns           v1                                     false        Namespace
nodes                             no           v1                                     false        Node
persistentvolumeclaims            pvc          v1                                     true         PersistentVolumeClaim
persistentvolumes                 pv           v1                                     false        PersistentVolume
pods                              po           v1                                     true         Pod
podtemplates                                   v1                                     true         PodTemplate
replicationcontrollers            rc           v1                                     true         ReplicationController
resourcequotas                    quota        v1                                     true         ResourceQuota
secrets                                        v1                                     true         Secret
serviceaccounts                   sa           v1                                     true         ServiceAccount
services                          svc          v1                                     true         Service
mutatingwebhookconfigurations                  admissionregistration.k8s.io/v1        false        MutatingWebhookConfiguration
validatingwebhookconfigurations                admissionregistration.k8s.io/v1        false        ValidatingWebhookConfiguration
customresourcedefinitions         crd,crds     apiextensions.k8s.io/v1                false        CustomResourceDefinition
apiservices                                    apiregistration.k8s.io/v1              false        APIService
controllerrevisions                            apps/v1                                true         ControllerRevision
daemonsets                        ds           apps/v1                                true         DaemonSet
deployments                       deploy       apps/v1                                true         Deployment
replicasets                       rs           apps/v1                                true         ReplicaSet
statefulsets                      sts          apps/v1                                true         StatefulSet
```

What is YAML

- Human-readable data serialization format
- Designed to be easily mapped to data types common to most high-level languages:
  - List [array/set]
  - Associative array [hash/object/map]
  - Scalar
- Friendly to grep/Python/Perl/Ruby operations
- Used by Kubernetes and many other cloud native projects for configuration and manifest formatting
- Eschews enclosures (quotation marks, brackets, braces, open/close-tags, etc.)
- Data structure hierarchy is maintained by outline indentation
  - Spaces only, no tabs!
  - 2 spaces is convention
- All legal JSON is legal YAML (!)
  - Anywhere you can use YAML you can use JSON
  - Documents can be all YAML, all JSON or a mix
- Rhymes with camel

```
---
receipt: Invoice3344
date: 2012-08-06
total: 17.95
customer:
  given: Dorothy
  family: Gale
items:
- name: snickers
  price: 4.00
  qty: 4
- price: 1.95
  name: tax
  qty: 1
```

YAML Quick Reference

- Lists
  - Conventional block format uses a hyphen+space to begin a new item in list.

```
--- # Favorite movies
- Casablanca
- North by Northwest
- The Man Who Wasn't There
```

  - Or JSON

```
--- # Shopping list
[milk, pumpkin pie, eggs, juice]
```

- Associative arrays
  - Keys are separated from values by a colon+space

```
--- # Indented Block
  name: John Smith
  age: 33
```

  - Or JSON

```
--- # Inline Block
{name: John Smith, age: 33}
```

- Block literals
  - Strings do not require quotation
  - Newlines preserved, first line indent and trailing space is stripped

```
--- |
  There once was a man from Ealing
  Who got on a bus to Darjeeling
      It said on the door
      "Please don't spit on the floor“
  So he carefully spat on the ceiling
```

  - Newlines folded, converts newlines to spaces and removes leading whitespace

```
--- >
  Wrapped text
  will be folded
  into a single
  paragraph

  Blank lines  denote
  paragraph breaks
```

- Combinations (mixing yaml and json)
  - Lists of associative arrays

```
- {name: John Smith, age: 33}
- name: Mary Smith
  age: 27
```

  - Associative arrays of lists

```
men: [John Smith, Bill Jones]
women:
  - Mary Smith
  - Susan Williams
```

Using YAML and JSON


```
ubuntu@ip-172-31-46-20:~$ ls -l
total 12
-rw-rw-r-- 1 ubuntu ubuntu 317 Oct 23 04:30 pod.json
-rw-rw-r-- 1 ubuntu ubuntu 220 Oct 23 04:30 pod.yaml
-rw-rw-r-- 1 ubuntu ubuntu 219 Oct 23 04:31 pod.yamson
ubuntu@ip-172-31-46-20:~$ cat *
{
  "apiVersion": "v1",
  "kind": "Pod",
  "metadata": {
    "name":"myapp-json",
    "labels": { "app": "myapp" }
  },
  "spec": {
    "containers": [
      {
        "name":"myapp-container",
        "image":"busybox",
        "command": ["sh", "-c", "echo Hello Kubernetes! && sleep 3600"]
      }
    ]
  }
}
```

JSON

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-yaml
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox
    command:
    - sh
    - -c
    - echo Hello Kubernetes! && sleep 3600
```

YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-yamson
  labels: { "app": "myapp" }
spec:
  containers:
  - name: myapp-container
    image: busybox
    command: [ "sh", "-c", "echo Hello Kubernetes! && sleep 3600" ]

ubuntu@ip-172-31-46-20:~$ kubectl create -f .
pod/myapp-json created
pod/myapp-yaml created
ubuntu@ip-172-31-46-20:~$ kubectl create -f pod.yamson
pod/myapp-yamson created
ubuntu@ip-172-31-46-20:~$ kubectl get pod
NAME           READY   STATUS    RESTARTS   AGE
myapp-json     1/1     Running   0          20s
myapp-yaml     1/1     Running   0          20s
myapp-yamson   1/1     Running   0          8s
ubuntu@ip-172-31-46-20:~$ kubectl logs -l app=myapp
Hello Kubernetes!
Hello Kubernetes!
Hello Kubernetes!
ubuntu@ip-172-31-46-20:~$
```

YAML / JSON

- Kubectl accepts yaml and/or json
  - Then converts it all to json before POSTing the data to the api-server

```
ubuntu@ip-172-31-46-20:~$ kubectl get pod myapp-json -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2019-10-23T04:36:06Z"
  labels:
    app: myapp
  name: myapp-json
  namespace: default
  resourceVersion: "93071"
  selfLink: /api/v1/namespaces/default/pods/myapp-json
  uid: 8c30126b-4300-4d73-919f-be0e8bec903f
spec:
  containers:
  - command:
    - sh
    - -c
    - echo Hello Kubernetes! && sleep 3600
    image: busybox
    imagePullPolicy: Always
    name: myapp-container
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: default-token-gssng
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: ip-172-31-46-20
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: default
  serviceAccountName: default
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: default-token-gssng
    secret:
      defaultMode: 420
      secretName: default-token-gssng
```

- Kubectl can output yaml or json
  - The -o yaml switch outputs manifests in yaml
  - The -o json switch outputs manifests in json
  - The -o wide switch display normal extended output

Pod Configuration Files

- Configured using YAML or JSON files
- The more verbose the manifest, the more effective Kubernetes becomes
- Fields for Pod configuration include (but not limited to):
  - `spec`: the pod specification
    - `volumes`[]: List of volumes that can be mounted by containers in the pod
    - `restartPolicy`: applies to all containers in pod
    - `terminationGracePeriodSeconds`: number of seconds to shutdown before kill
    - `imagePullSecrets`: secrets that provide credentials for pulling images from secure registries
    - `containers`[]: containers to run in the pod
      - `name`: Name of the container
      - `image`: Docker image name
      - `command`[]: command line (like Docker ENTRYPOINT)
      - `args`[]: entrypoint arguments (like Docker CMD)
      - `env`[]: environment vars
      - `ports`[]: port exposures
        - `containerPort`: port to expose
        - `protocol`: port protocol
      - `volumeMounts`[]: paths to mount volumes
      - `securityContext`[]: security controls (users, privileges, etc.)
      - `livenessProbe`[]: user-defined condition to determine container health (and if it needs to be restarted)
      - `readinessProbe`[]: user-defined condition to determine container readiness (when it can start taking user traffic)
      - `lifecycle`[]: actions to run after the container starts and before the container stops

```
$ cat website.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: apache
  name: website
  namespace: default
spec:
  containers:
  - image: httpd:2.2
    imagePullPolicy: IfNotPresent
    command:
    - /usr/local/apache2/bin/apachectl
    args:
    - -f
    - /usr/local/apache2/conf/httpd.conf
    name: apache
    volumeMounts:
    - mountPath: /tmp
      name: scratch-disk-space
  restartPolicy: Always
  volumes:
  - name: scratch-disk-space
    emptyDir: {}

$ kubectl apply -f website.yaml
pod/website created

$ kubectl get pod
NAME       READY     STATUS    RESTARTS   AGE
website    0/1       Pending   0          15s

$ kubectl logs website
...

$ kubectl get pod
NAME       READY     STATUS    RESTARTS   AGE
website    1/1       Running   0          30s
```

Summary

- A workload is a microservice deployed as a Kubernetes Pod
- Pods are the unit of application deployment in Kubernetes
  - Pods are atomic and scheduled on a single node
- YAML files are used to specify resources and the metadata associated with them
  - Well-written manifests are key to deploying application pods
- All Kubernetes configs require a declaration of:
  - apiVersion
  - kind
  - metadata
  - spec

Lab 2

- Pods: Basics

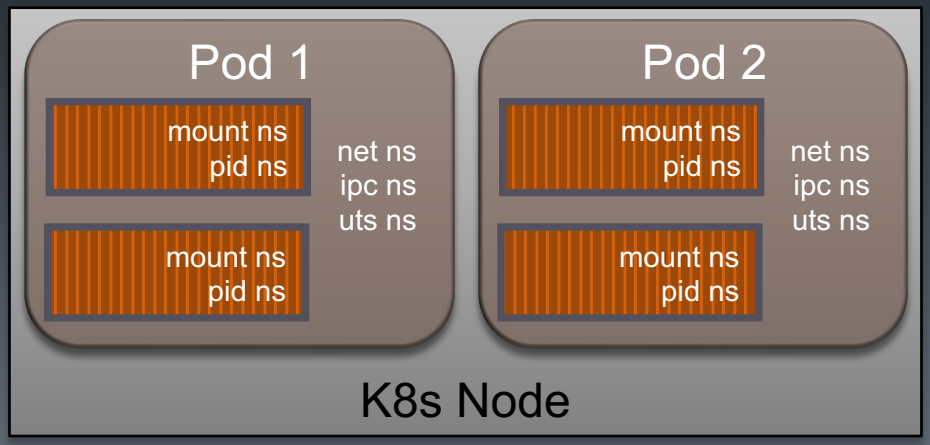
3: Pods: Architecture

Objectives

- Build an understanding of Pod architecture
- Examine common pod patterns and nomenclature
- Explore the other kinds of containers that a pod supports
- Discuss pod and container resource requests and limits
- Introduce other pod settings that influence container behavior and security

Multi-Container Pods

- Pods Model application-specific "logical hosts", enabling multiple containers (unrelated process trees) to run in a shared context
  - Therefore, a *pod* (as in a pod of whales) is a group of containers, the shared storage for those containers, and options about when and how to run the containers
  - The shared context is defined by Linux kernel namespaces and is known as the "pod sandbox"
    - Enables sharing and fast communication (ex. via loopback)
- Containers in a pod are placed and scheduled together as a unit on a Kubernetes Node, making a pod the atomic unit of scheduling
  - When the pod is replicated, all the containers are replicated by the same factor
    - Ex: replicating a 2-container pod 3 times results in 6 total containers



Infrastructure Container

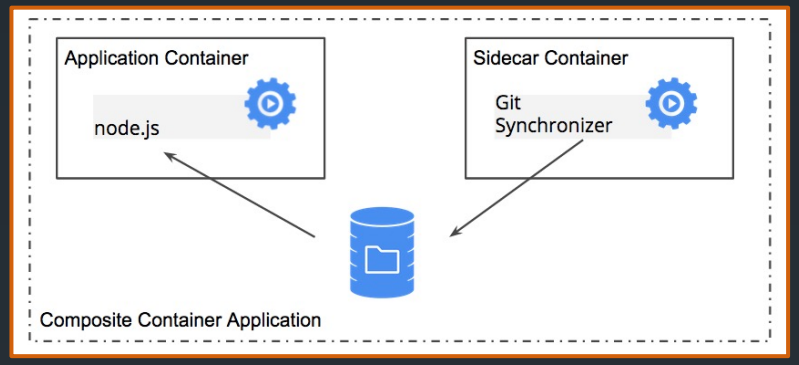
- The Infrastructure Container implements the pod sandbox and is used to set up and hold the Network and IPC Linux namespaces and resource limits for each pod
  - Containers isolate their own Mount, PID, and UTS namespaces but share the `/etc/hostname` and `/etc/resolv.conf` pod wide (UTS)
  - Often referred to as the "pause" container because it executes the pause function
    - A necessary live process so the pod receives an IP address from the CNI plugin

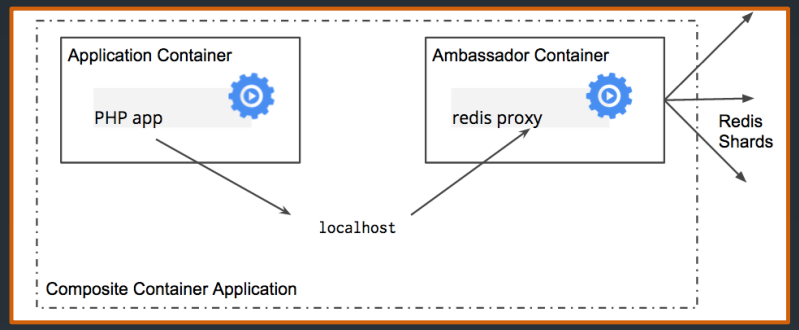
```
$ cat ns-ex.yaml

apiVersion: v1
kind: Pod
metadata:
  name: ns-sharing-example
spec:
  containers:
  - name: ns-sharing-container-1
    image: httpd
  - name: ns-sharing-container-2
    image: sidecar
```

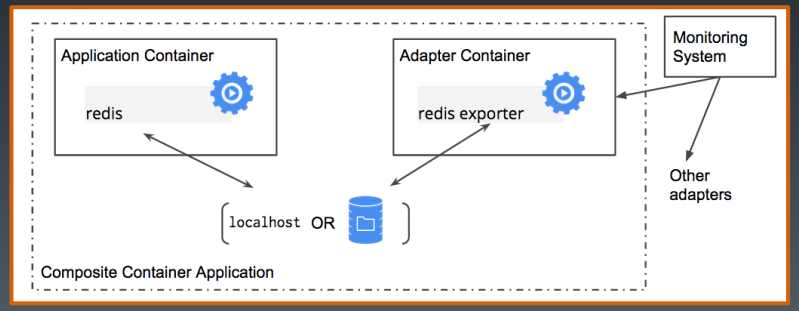
```
$ kubectl apply -f ns-ex.yaml
pod/ns-sharing-example created

$ kubectl get po
NAME                 READY   STATUS    RESTARTS   AGE
ns-sharing-example   2/2     Running   0          77s

$ sudo docker container ls -f "name=ns-sharing"
CONTAINER ID   IMAGE                  COMMAND                  CREATED         STATUS         PORTS   NAMES
9375f1680881   sidecar                "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes           k8s_ns-sharing-container-2...
eb2a788a0e8d   httpd                  "httpd-foreground"       2 minutes ago   Up 2 minutes           k8s_ns-sharing-container-1...
eb5ea13b9c2d   k8s.gcr.io/pause:3.2   "/pause"                 2 minutes ago   Up 2 minutes           k8s_POD_ns-sharing-example...

$ sudo docker container inspect $(sudo docker container ls -qf "ancestor=httpd") | grep eb5ea13b9c2d
 "ResolvConfPath": "/var/lib/docker/containers/eb5ea13b9c2d86e145dd47b82347433b47dd5b4f5ca20613e87025078d08b738/resolv.conf",
 "HostnamePath": "/var/lib/docker/containers/eb5ea13b9c2d86e145dd47b82347433b47dd5b4f5ca20613e87025078d08b738/hostname",
   "NetworkMode": "container:eb5ea13b9c2d86e145dd47b82347433b47dd5b4f5ca20613e87025078d08b738",
   "IpcMode": "container:eb5ea13b9c2d86e145dd47b82347433b47dd5b4f5ca20613e87025078d08b738",
      "io.kubernetes.sandbox.id": "eb5ea13b9c2d86e145dd47b82347433b47dd5b4f5ca20613e87025078d08b738"

$ sudo docker container inspect $(sudo docker container ls -qf "ancestor=sidecar") | grep eb5ea13b9c2d
 "ResolvConfPath": "/var/lib/docker/containers/eb5ea13b9c2d86e145dd47b82347433b47dd5b4f5ca20613e87025078d08b738/resolv.conf",
 ...
```

Pod Patterns

- Sidecar
  - Sidecars extend and enhance the "main" container in the Pod
    - Example: Nginx web server container; add a container that syncs the file system with a git repository
      - Share the file system between the containers and you have built Git push-to-deploy
      - A standard gitsync sidecar can be found here: https://github.com/kubernetes/git-sync
- Ambassador
  - Ambassadors proxy a Pod local connection to the world outside
    - Example: Redis cluster with read-replicas and a single write master
      - Create a Pod that groups the main application with a Redis ambassador container which splits reads and writes, sending them on to the appropriate servers
- Adapter
  - Adapter containers standardize and normalize output
    - Example: A task monitoring N different applications where each application has a different way of exporting monitoring data (e.g. JMX, StatsD, application specific statistics) but every monitoring system expects a consistent and uniform data model for the monitoring data it collects



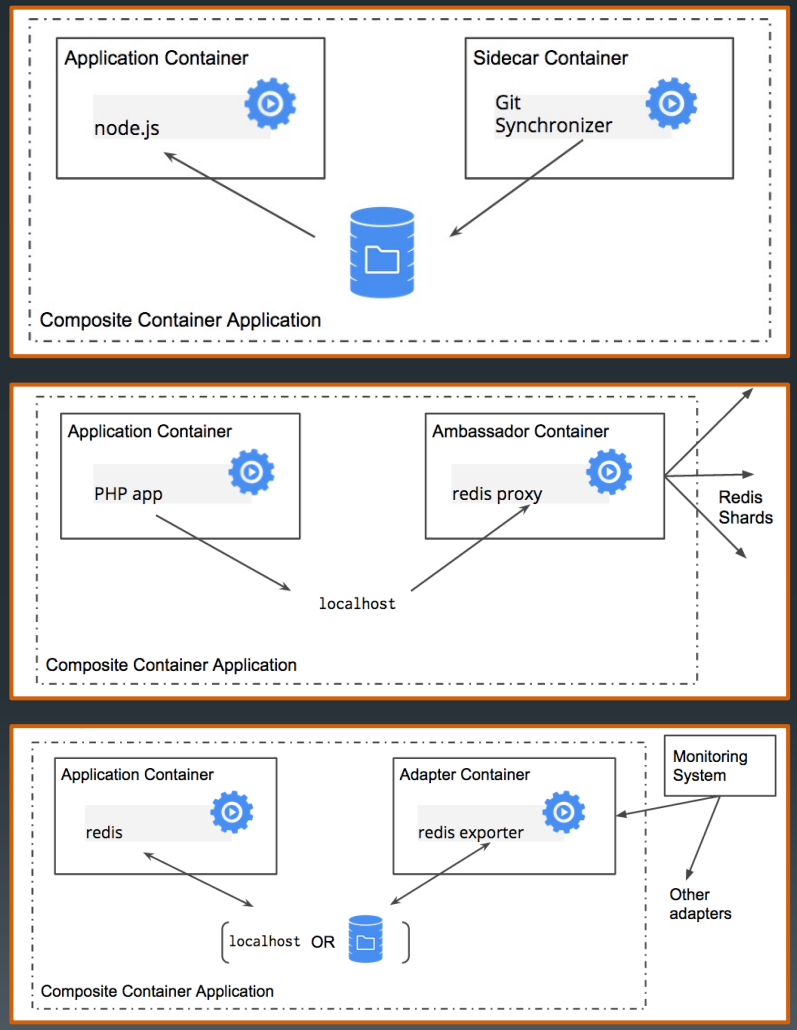




- Brendan Burns, Distinguished Engineer at Microsoft and former Software Engineer at Google
https://research.google.com/pubs/pub45406.html

Pod Patterns: Benefits

- Containers are units of:
  - Resource accounting and allocation – "main" container given priority over resources, "helper" container can be given strict restraints
  - Reuse – single version of "helper" used among many apps
  - Deployment – roll out/back independently
  - Failure boundaries – "main" container can continue even if "helper" fails
  - Security – one container can be given privilege without exposing the other(s) to that privilege
- Divide responsibility for development between two separate programming teams



- Brendan Burns, Distinguished Engineer at Microsoft and former Software Engineer at Google
https://research.google.com/pubs/pub45406.html

Init Containers

- Init Containers are used to bootstrap or initialize an application pod
- Started in order
- After network and volumes are initialized
- Each Init Container must exit successfully before the next is started
  - All Init Containers must exit successfully before *any* containers in the Containers array are allowed to run
    - Logic is that if Init Containers fail, the application is not properly bootstrapped and should not run
- Retried according to the Pod restartPolicy
  - If `Always` Init Containers use `OnFailure`
- A Pod is not Ready until all Init Containers have succeeded
- Ports on an Init Container are not aggregated under a service

```
spec:
  initContainers:
  - name: "init-chown-data"
    image: "busybox:latest"
    # 65534 is the nobody user that prometheus uses.
    command: ["chown", "-R", "65534:65534", "/data"]
    securityContext:
      capabilities:
        add: ["CAP_CHOWN"]
        drop: ["ALL"]
    volumeMounts:
    - name: storage-volume
      mountPath: /data

  containers:
  - name: prometheus-server
    image: prom/prometheus:latest
    args:
      - --config.file=/etc/config/prometheus.yml
      - --storage.tsdb.path=/data
    volumeMounts:
      - name: storage-volume
        mountPath: /data
    securityContext:
      runAsUser: 65534
  volumes:
    - name: storage-volume
      persistentVolumeClaim:
...
```

Example Uses for Init Containers

- Contain utilities that are not desirable to include in the app container for security reasons
- Execute custom code for setup that is not present in an app image
- Place values into a config file or run a template tool to dynamically generate a config file for the main app container
- Block or delay the startup of containers until some set of preconditions are met

```
apiVersion: v1
kind: Pod
metadata:
  name: git-init
spec:
  initContainers:
  - name: init-container
    image: rxmllc/alpine-git:0.1
    command: ["bin/sh", "-c"]
    args: ["git clone https://github.com/RX-M/trash-levels.git trash-levels "]
    volumeMounts:
    - name: sharedvol
      mountPath: /trash-levels
  containers:
  - name: busybox
    image: busybox
    command: ["bin/sh", "-c"]
    args: [echo "files from trash-levels github repo:" ; ls /trash-levels]
    volumeMounts:
    - name: sharedvol
      mountPath: /trash-leve
  volumes:
  - name: sharedvol
    emptyDir: {}
  restartPolicy: Never
```

```
$ kubectl apply -f git-init.yaml
pod/git-init created

$ kubectl logs git-init -c init-container
Cloning into 'trash-levels'...

$ kubectl logs git-init -c busybox
files from trash-levels github repo:
Dockerfile
LICENSE
README.md
docs
k8s
main.go
makefile
trash-levels
```

Files from init container

Ephemeral Containers

- Special type of container that runs temporarily in an *existing* Pod to accomplish user-initiated actions like troubleshooting
- Useful for interactively troubleshooting:
  - Crashed containers
  - Containers made from "distroless" images (container images with few or no tools or shells)
- Differences from other containers:
  - No guarantees for resources or execution
  - Never automatically restarted
- Can be injected into running pods using `kubectl debug`
  - Using `--share-processes` with `--copy-to` will enable process namespace sharing in a copy of the pod

```
spec:
  ephemeralContainers:
  - image: busybox
    imagePullPolicy: IfNotPresent
    name: debugger-r9nm6
    resources: {}
    stdin: true
    targetContainerName: bin-check
    terminationMessagePolicy: File
    tty: true
 containers:
  - image: rxmllc/trash-levels
    imagePullPolicy: Always
    name: bin-check
    ...
```

```
~$ kubectl run bin-check --image rxmllc/trash-levels

pod/bin-check created

~$ kubectl logs bin-check

2020/10/22 21:41:40 Trash Levels Server listening on port: 8080

~$ kubectl exec bin-check -- ls -l /
OCI runtime exec failed: exec failed: container_linux.go:349: starting
container process caused "exec: \"ls\": executable file not found in
$PATH": unknown
command terminated with exit code 126

~$ kubectl exec -it bin-check -- /bin/sh

OCI runtime exec failed: exec failed: container_linux.go:349: starting
container process caused "exec: \"/bin/sh\": stat /bin/sh: no such file
or directory": unknown
command terminated with exit code 126

~$ kubectl debug -it bin-check --image=busybox --target=bin-check

Defaulting debug container name to debugger-r9nm6.
If you don't see a command prompt, try pressing enter.

/ # ls -l /

total 36
drwxr-xr-x    2 root     root         12288 Oct 12 23:47 bin
drwxr-xr-x    5 root     root           360 Oct 22 21:45 dev
drwxr-xr-x    1 root     root          4096 Oct 22 21:45 etc
drwxr-xr-x    2 nobody   nobody        4096 Oct 12 23:47 home
dr-xr-xr-x  180 root     root             0 Oct 22 21:41 proc
drwx------    1 root     root          4096 Oct 22 21:45 root
dr-xr-xr-x   13 root     root             0 Oct 22 21:41 sys
drwxrwxrwt    2 root     root          4096 Oct 12 23:47 tmp
drwxr-xr-x    3 root     root          4096 Oct 12 23:47 usr
drwxr-xr-x    4 root     root          4096 Oct 12 23:47 var
```

Container (built from scratch) has no ls tool or a shell!

Debugging container using a lightweight Linux environment solves the problem!

Resource Limits

- You can specify the desired CPU, memory, and disk for containers in specs to ensure the scheduler chooses a node with the appropriate resources
  - CPU is specified in units of cores
    - `spec.container[].resources.requests.cpu`
  - Memory is specified in units of bytes
    - `spec.container[].resources.requests.memory`
  - Disk is specified in bytes
    - `spec.containers[].resources.requests.ephemeral-storage`
- Each container of a Pod can optionally specify constraints implemented by the container manager using CGroups:
  - `spec.container[].resources.limits.cpu`
  - `spec.container[].resources.limits.memory`
  - `spec.containers[].resources.limits.ephemeral-storage`
- If value of requests is not specified, they are set to be equal to limits by default (when limits are defined)
  - Limits must be greater than or equal to resource requests
- If the scheduler cannot find any node where a pod can fit, then the pod will remain unscheduled until a place can be found

```
apiVersion: v1
kind: Pod
metadata:
  name: resource-pod
spec:
  containers:
  - name: app
    image: app:v2.2
    resources:
      requests:
        memory: "128Mi"
        cpu: "0.25"
      limits:
        memory: "256Mi"
        cpu: "0.5"
  - name: sidecar
    image: sidecar:v3.4
    resources:
      requests:
        memory: "64Mi"
        cpu: "0.1"
      limits:
        memory: "128Mi"
        cpu: "0.25"
```

Pod Security Context

- Defines privilege and access control settings for a Pod or Container
  - Container settings override Pod settings
  - Most settings are valid for Pods and Containers though some are Pod only and some are Container only
- Security context settings include:
  - Discretionary Access Control
    - runAsNonRoot (bool) ensures containers run as a non root user (image metadata or pod spec must set user <> 0)
    - `runAsUser` field specifies that for any Containers in the Pod, the first process runs with a given user ID
    - `runAsGroup` field specifies that for any Containers in the Pod, the first process runs with a given group ID
    - `supplementalGroups` (P) lists groups applied to the first process run in each container, in addition to the container's primary GID
    - fsGroupChangePolicy (P) sets volume to fsGroup ownership if supported by volume (no affect on secret / configmap / emptydir types), values: "OnRootMismatch" / "Always"
      - `fsGroup` (P) GID to set when fsGroupChangePolicy is met
  - Linux Security Module Settings
    - `seLinuxOptions` (Security Enhanced Linux) sets SELinux options for the pod
      - SELinux must be loaded on the host OS
    - `seccompProfile` (Secure Computing mode) sets the seccomp profile for the pod
      - Seccomp must be enabled in the Linux kernel
  - Capabilities
    - `privileged` (C)(bool) enables all capabilities for root user
    - `capabilities` (C) allows Linux system call capabilities to be added (`add`) and dropped (`drop`)
    - `allowPrivilegeEscalation` (C)(bool) controls whether a process can gain more privileges than its parent process, controlling the no_new_privs flag
      - Always true when the container is run as privileged or has CAP_SYS_ADMIN
  - `readOnlyRootFilesystem` (C)(bool) sets container filesystem to read only
  - `windowsOptions` Windows specific settings

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-nginx
spec:
  securityContext:      # pod level security configs
    runAsUser: 1000
    fsGroup: 2000
  volumes:
  - name: sharedvol
    emptyDir: {}
  containers:
  - name: nginx
    image: nginx
    command: ["nginx", "-g", "daemon off;"]
    volumeMounts:
    - name: sharedvol
      mountPath: /data
    securityContext:   # container level security configs
      capabilities:
        add: ["NET_ADMIN", "SYS_TIME"]
      seLinuxOptions:
        level: "s0:c123,c456"
      readOnlyRootFilesystem: true
      runAsNonRoot: true
```

Health Checks

- Kubernetes provides two layers of health checking
  - HTTP or TCP checks
    - K8s can attempt to connect to a particular endpoint and give a status of healthy on a successful connection
  - Application-specific health checks
    - K8s can be configured to use a command line scripts
- livenessProbe
  - Specifies a health check – a container failing a health check is terminated and restarted according to its restart policy
- readinessProbe
  - Uses the same options but is used to assess container readiness to receive client/peer requests
- startupProbe
  - Intended to provide window to allow application to fully start up, potentially preventing livenessProbe from failing
  - Block other probes until successfully completes or if startupTime passes, applies restartPolicy after 300s
  - Test uses the same options with additional timing controls
    - spec.failureThreshold * spec.periodSeconds = startupTime allowed
- Available options (typically only one of):
  - httpGet - Takes a path and port
    - Status codes 200-399 are considered healthy
  - tcpSocket - Takes a port
    - Any successful connect passes
  - Exec - Takes a command
    - Exit code of 0 is success

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        livenessProbe:
          # httpGet Probe
          httpGet:
            path: /status/
            port: 80
          initialDelaySeconds: 30
          timeoutSeconds: 1
          periodSeconds: 10
        startupProbe:
          httpGet:
            path: /status/
            port: 80
          failureThreshold: 30
          periodSeconds: 10
```

```
# tcpSocket Probe
livenessProbe:
  tcpSocket:
    port: 80
```

```
# exec Probe
livenessProbe:
  exec:
    command: /scripts/ck.sh
```

Pre/Post Actions

- Containers can be assigned lifecycle actions to run after the container starts and before the container stops
  - Convenient for initializing the started container or cleaning up before the container shutdown
- Management of the container blocks until lifecycle actions are complete
- These hooks are called at least once
  - Be prepared for more than one call!
- Two types of hook handlers:
  - httpGet - request against a specific endpoint on the container
  - exec - Executes a command inside the cgroups and namespaces of the container
- `postStart` – called immediately after a container is created
  - If the handler fails, the container is terminated and restarted according to its restart policy
  - Other management of the container blocks until the action completes
- `preStop` – called immediately before a container is terminated
  - Blocking, so it must complete before the call to delete the container can be sent
    - If the hook hangs during execution, the Pod phase stays in a Terminating state and is killed after `terminationGracePeriodSeconds` of pod ends
    - Otherwise, the container is terminated after the handler completes

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-hook
spec:
  replicas: 3
  selector:
    matchLabels:
      app: apache-hook
template:
    metadata:
      labels:
        app: apache-hook
    spec:
      containers:
      - name: apache-hook
        image: bitnami/apache:latest
        ports:
        - containerPort: 80
        lifecycle:
          postStart:
            httpGet:
              path: http://my.regsvr.com/register/
              port: 80
          preStop:
            exec:
              command: ["/usr/bin/apachectl","-k", "graceful-stop"]
```

Summary

- Pods model a logical host that enables containers running within them to work together closely (e.g. a webserver with a proxy)
- The infrastructure container is used as a placeholder for Linux kernel namespaces and implements the pod sandbox
- InitContainers can be used in pod configs for bootstrapping activities
- EphemeralContainers can be used in pods for troubleshooting
- Resource requests and limits are used to:
  - Influence dynamic scheduling
  - Set constraints on processes during execution
- Container security can be adjusted in the pod spec
- Pods can be monitored for health and readiness
- Pods can be configured to perform post-start and pre-shutdown tasks

Lab 3

- Pods: Architecture

4: Application Configuration

Objectives

- Explore common configuration paradigms in modern systems
- Explore the use of ConfigMaps for configuration files & data
- Discuss creating and leveraging Secrets in configs
- Discuss creating immutable ConfigMaps and Secrets
- Examine using pod metadata in volumes and environment variables through the Downward API

Application Configuration Paradigms

- Possible configuration schemes:
  - Default Settings
    - Runs the program with the built-in options as they are coded in the binary
  - Command Line Options
    - Runs the program with modified options based on user input
  - Environment Variables
    - Runs the program with options set by the environment that is running them
  - Configuration Files
    - Runs the program using a pre-defined document containing various runtime options
    - Usually stored in some accessible location

```
ubuntu@ip-172-31-16-136:~$ docker run fluent/fluent-bit  /fluent-bit/bin/fluent-bit
Fluent Bit v1.6.10
* Copyright (C) 2019-2020 The Fluent Bit Authors
* Copyright (C) 2015-2018 Treasure Data
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

Error: No Input(s) have been defined. Aborting

ubuntu@ip-172-31-16-136:~$
```

```
ubuntu@ip-172-31-16-136:~$ docker run fluent/fluent-bit  /fluent-bit/bin/fluent-bit -i stdin -o stdout
Fluent Bit v1.6.10
* Copyright (C) 2019-2020 The Fluent Bit Authors
* Copyright (C) 2015-2018 Treasure Data
* Fluent Bit is a CNCF sub-project under the umbrella of Fluentd
* https://fluentbit.io

[2021/01/09 05:41:08] [ info] [engine] started (pid=1)
[2021/01/09 05:41:08] [ info] [storage] version=1.0.6, initializing...
[2021/01/09 05:41:08] [ info] [storage] in-memory
[2021/01/09 05:41:08] [ info] [storage] normal synchronization mode, checksum disabled, max_chunks_up=128
[2021/01/09 05:41:08] [error] [input collector] COLLECT_EVENT registration failed
[2021/01/09 05:41:08] [ info] [sp] stream processor started
```

```
1  [SERVICE]
2      Flush     5
3      Daemon    off
4      Log_Level debug
5
6  [INPUT]
7      Name  cpu
8      Tag   my_cpu
9
10 [OUTPUT]
11     Name  stdout
12     Match my*cpu
```

Application Configuration in Kubernetes

- Configurable Settings
  - Program to run
  - Program CL Options
- Environmental Settings
  - Environment Variables
  - Resource Settings
  - Volume Mounts

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: configurable
  name: configurable
spec:
  containers:
  - command:
    - ls
    - -l
    - -r
    env:
    - name: SHELL
      value: /bin/sh
    image: busybox
    name: configurable
    volumeMounts:
    - name: certs
      mountPath: /etc/ssl/certs
  volumes:
  - name: certs
    hostPath:
      path: /etc/ssl/certs
```

ConfigMaps

```
ubuntu@ip-172-31-0-249:~$ cat config.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mlparams
data:
  k: "9"
  q: "42"
ubuntu@ip-172-31-0-249:~$ kubectl apply -f config.yaml
configmap/mlparams created
ubuntu@ip-172-31-0-249:~$ kubectl get cm
NAME       DATA   AGE
mlparams   2      7s
ubuntu@ip-172-31-0-249:~$ kubectl describe cm mlparams
Name:         mlparams
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","data":{"k":"9","q":"42"},"kind":"ConfigM
ams","namespace":"default"}}

Data
====
q:
----
42
k:
----
9
Events:  <none>
ubuntu@ip-172-31-0-249:~$
```

```
ubuntu@ip-172-31-0-249:~$ cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: configme
spec:
  restartPolicy: Never
  containers:
  - name: test
    image: k8s.gcr.io/busybox
    command: [ "/bin/sh", "-c", "env" ]
    env:
    - name: ML_PARAM_K
      valueFrom:
        configMapKeyRef:
          name: mlparams
          key: k
ubuntu@ip-172-31-0-249:~$ kubectl apply -f pod.yaml
pod/configme created
ubuntu@ip-172-31-0-249:~$ kubectl get pod
NAME       READY   STATUS      RESTARTS   AGE
configme   0/1     Completed   0          5s
ubuntu@ip-172-31-0-249:~$ kubectl logs configme
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_SERVICE_PORT=443
HOSTNAME=configme
SHLVL=1
HOME=/root
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_PROTO=tcp
ML_PARAM_K=9
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
PWD=/
KUBERNETES_SERVICE_HOST=10.96.0.1
ubuntu@ip-172-31-0-249:~$
```

- Enables the Kubernetes API as a (configuration) data source!
- ConfigMap resources
  - meant to house configuration data in key/value pairs
- Keyed ConfigMap data can be mounted into a pod
  - Files
    - The key is the file name / the value is the contents of the file
  - Environment variables
    - The key can be the name of the environment variable (or you can set your own name) / the value is the value of the environment variable
    - Environment variables can be used as command line args

ConfigMaps with complex data

- These configuration artifacts should be decoupled from image content in order to keep containerized applications portable
- The ConfigMap resource provides mechanisms to inject containers with configuration data while keeping containers agnostic of Kubernetes
- ConfigMap can be used to store fine-grained information like individual properties or coarse-grained information like entire config files or JSON blobs

```
# ConfigMap of prometheus.yml config file
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    scrape_configs:
    - job_name: 'kubernetes-apiservers'
      kubernetes_sd_configs:
      - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name]
        action: keep
        regex: default;kubernetes;https
    - job_name: 'kubernetes-nodes'
...
```

```
scrape_configs:
- job_name: 'kubernetes-apiservers'
  kubernetes_sd_configs:
  - role: endpoints
  scheme: https
  tls_config:
    ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
    #    insecure_skip_verify: true
  bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
  relabel_configs:
  - source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name]
    action: keep
    regex: default;kubernetes;https
-  job_name: 'kubernetes-nodes'
...
```

Secrets

- Kubernetes objects used to hold sensitive information
  - Ex. passwords, OAuth tokens, SSH keys
- Only referenced from pods in same namespace
- Two ways to store data:
  - data: you base64 encode data
  - stringData: base64 encoded for you
- Typically encrypted in etcd through api-server settings
- Secrets type is used for programmatic secret processing (implies key names and contents expected):
  - Opaque - Normal arbitrary data
  - Docker-Registry (kubernetes.io/dockerconfigjson) – docker/oci registry token
  - TLS (kubernetes.io/tls) - public key certificate and private key pair
  - Other Secret types:
    - kubernetes.io/service-account-token - In-cluster credential for containers
    - kubernetes.io/dockercfg - Serialized ~/.dockercfg file
    - kubernetes.io/dockerconfigjson - Serialized ~/.docker/config.json file
    - kubernetes.io/basic-auth - Stores basic authentication information
    - kubernetes.io/ssh-auth - Stores SSH Keys
    - bootstrap.kubernetes.io/token - Authentication token used to allow new nodes to join the cluster)

```
$ echo -n 'admin' | base64
YWRtaW4=
$ echo -n 'password' | base64
cGFzc3dvcmQ=

$ vi dev-db-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: dev-db-secret-v1
type: Opaque
data:
  password: cGFzc3dvcmQ=
  username: YWRtaW4
```

```
$ kubectl create secret --help
Create a secret using specified subcommand.

Available Commands:
  docker-registry Create a secret for use with a Docker registry (kubernetes.io/dockerconfigjson)
  generic         Create a secret from a local file, directory or literal value (Opaque)
  tls             Create a TLS secret (kubernetes.io/tls)
```

Protecting Secrets

- Contents of secrets are only encoded in base64
  - Not encrypted by default in etcd
- Access to secrets should be restricted using RBAC
- Secrets can be further protected by encrypting them in etcd
  - With encryption at rest enabled, secrets are encrypted on write within etcd
  - `kube-apiserver` must be configured for encryption at rest with `--encryption-provider-config`
  - KMS-based systems that transmit secret data at startup and store secrets in memory are ideal
    - i.e. AWS Secrets Manager or Hashicorp Vault

```
~$ kubectl get secret secret1 -o yaml

apiVersion: v1
data:
  mykey: bXlkYXRh
kind: Secret
metadata:
  name: secret1
  namespace: default
type: Opaque
```

Secret stored in etcd without encryption enabled

```
~$ etcdctl get /registry/secrets/default/secret1 --cacert /etc/kubernetes/pki/etcd/ca.crt
--cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key

/registry/secrets/default/secret1
k8s


v1Secret
secret1default"*$02833b1c-fc3d-413a-a9b0-8c9ff4b9b4ef2ňz
kubectl-createUpdatevňFieldsV1:.
,{"f:data":{".":{},"f:mykey":{}},"f:type":{}}B
mykeymydataOpaque"
```

```
~$ etcdctl get /registry/secrets/default/secret1 --cacert /etc/kubernetes/pki/etcd/ca.crt
--cert /etc/kubernetes/pki/etcd/server.crt --key /etc/kubernetes/pki/etcd/server.key

/registry/secrets/default/secret1
k8s:enc:aescbc:v1:key1:
!*j.BjO^3QoTT:AvB[Uo1J36(K y J $L¤|Ku.N
_                    Fd U p W I q  T ?`S $%>D
._rx ?$n          ,[%eL.Y r .O#I ^YW D P
```

Secret stored in etcd with encryption enabled

Creating ConfigMaps/Secrets

- Can be created from directories, files, or key-value pairs supplied from the command line
  - `--from-file`
    - Creates a data key from the supplied file
      - File name becomes the key, contents become the value
      - Key can be overridden with: `--from-file=<my-key-name>=<path-to-file>`
    - If a directory is supplied, each file creates a new key in the resulting ConfigMap/Secret
  - `--from-env-file`
    - Parses every line in a file as separate data keys in the resulting ConfigMap/Secret
  - `--from-literal=`
    - Creates a data key for each key-value supplied by the user in the command line
  - Each `--from-file` or `--from-literal` arguments creates a new key
- As a yaml file using `kubectl apply -f`
  - Secrets must have the values base64 encoded

```
$ cat app.conf
key=value1
property=config
another.property=configValue

$ cat ui.conf
color.one=blue
color.two=white
font.headings=sans-serif
font.body=serif

$ kubectl create configmap config-v1 \
--from-file=/home/user/configmap/

configmap/config created

$ kubectl describe configmap config

Name:         config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
app.conf:
----
key=value1
property=config
another.property=configValue

ui.conf:
----
color.one=blue
color.two=white
font.headings=sans-serif
font.body=serif

Events:  <none>
```

ConfigMap/Secret Volumes

- When a pod is created, Secret/ConfigMap existence is not checked!
  - If the kubelet can not fetch the Secret/ConfigMap:
    - It will report a pod event
    - The kubelet will periodically retry
    - Once the secret/configMap is created and fetched by the kubelet, it will create the volume and start the associated pod
- When a Secret/ConfigMap being consumed in a volume is updated, projected keys are eventually updated as well
- `imagePullSecret` pass a secret that contains an image registry credential to Kubelet so it can pull a private image on behalf of your pod
  - Never mounted inside pod/container filesystem
- You can specify the permission mode bits of all Secret/ConfigMap files (or part of the files a Secret/ConfigMap will have)
  - Otherwise 0644 is used by default

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: secret-file-pod
  labels:
    secretpod: db-client
spec:
  volumes:
  - name: dev-db-secret
    secret:
      secretName: dev-db-secret-v1
      defaultMode: 400
  - name: app-config
    configMap:
      name: config-v1
  containers:
  - name: secret-file-container
    image: redis
    imagePullSecrets:
    - name: myregistrykey
    volumeMounts:
    - name: dev-db-secret
      readOnly: true
      mountPath: "/etc/secret-volume"
    - name: app-config
      mountPath: "/etc/app/config"
```

Secrets/ConfigMaps as Environment Variables

- Secrets appear as normal environment variables containing the base-64 decoded values of the secret data
  - ConfigMaps are always plain text
- Add an environment variable to each container for each key you wish to consume
- When a Secret/ConfigMap being consumed in ENVs is updated, projected values are NOT updated!
  - Requires a pod restart or rolling update
- Use `envFrom` to define all of a Secret/ConfigMap's data as container environment variables
  - *Keys that are considered invalid environment variable names will be skipped!*
    - The pod will be allowed to start
    - An `InvalidVariableNames` event will contain the list of invalid keys that were skipped

```
apiVersion: v1
kind: Pod
metadata:
  name: env-pod
spec:
  containers:
  - name: env-container
    image: redis
    env:
    - name: SECRET_USERNAME
      valueFrom:
        secretKeyRef:
          name: dev-db-secret-v1
          key: username
    - name: SECRET_PASSWORD
      valueFrom:
        secretKeyRef:
          name: dev-db-secret-v1
          key: password
    - name: CONFIG_VAR
      valueFrom:
        configMapKeyRef:
          name: config-v1
          key: property
    envFrom:
    - configMapRef:
        name: multi-config
```

Immutable ConfigMaps/Secrets

- Set individual Secrets and ConfigMaps as immutable
  - Prevents updates to ConfigMaps and Secrets that may break applications running in pods
  - Improves cluster performance
    - Reduces load on kube-apiserver by closing watches for Secrets and ConfigMaps
  - Requires `immutable: true` in manifest to set a ConfigMap or Secret to be immutable
- Update strategy for Immutable ConfigMaps and Secrets:
  - Create a new ConfigMap with updated contents
  - Update any pods or their controllers that are using the old ConfigMap with the new version
    - This approach retains the old version (for a time) of the ConfigMap/Secret so it is available in case a rollback is necessary

```
$ kubectl get cm config-v1
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-v1
data:
  app.conf: |
    key: value1
    property: config
    another.property: configValue
  ui.conf: |
    color.one: blue
    color.two: white
    font.headings: sans-serif
    font.body: serif
immutable: true

$ kubectl get cm config-v2
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-v2
data:
  app.conf: |
    key: value1
    property: config
    another.property: configValue
  ui.conf: |
    color.one: red
    color.two: white
    font.headings: serif
    font.body: serif
immutable: true
```

Downward API

- Allows containers to consume information about themselves or the cluster without using the kubectl client or API server
- Ways of exposing Pod and Container fields through:
  - Environment variables
  - DownwardAPIVolumeFiles
- Several basic data fields like names, labels, and annotations are available via the `fieldRef` key:
  - `spec.nodeName`
  - `status.hostIP`
  - `metadata.name`
  - `metadata.namespace`
  - `status.podIP`
  - `spec.serviceAccountName`
  - `metadata.uid`
  - `metadata.labels`
    - When appended with `['<key>']` a single value of a pod's label defined by <key>
  - `metadata.annotations`
    - When appended with `['<key>']` a single value of a pod's annotation defined by <key>
- Container resource Information available via `resourceFieldRef`:
  - Container CPU limits and request
  - Memory limits and memory request

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: downward-api-pod
  labels:
    cluster: test-cluster1
    rack: rack-22
spec:
  containers:
  - name: downward-api-container
    image: busybox
    env:
    - name: MY_POD_IP
      valueFrom:
        fieldRef:
          fieldPath: status.podIP
    volumeMounts:
    - name: dapi-info
      mountPath: /etc/dapinfo
  volumes:
  - name: dapi-info
    downwardAPI:
      items:
      - path: "labels"
        fieldRef:
          fieldPath: metadata.labels
      - path: "mem_limit"
        resourceFieldRef:
          containerName: downward-api-container
          resource: limits.memory
          divisor: 1Mi
```

Summary

- Applications in modern systems are configurable using command-line arguments, environment variables, and configuration files
- ConfigMaps are used for the following:
  - Populate the value of environment variables
  - Set command-line arguments in a container
  - Populate config files in a volume
- Secrets can be used to pass sensitive data to Pods
  - kubelet can use secrets for logging into registries
- Downward API can be used to deliver dynamic metadata to applications running in Pods as files or environment variables

Lab 4

- ConfigMaps, Secrets, and Downward API

Day 2

5. Application Controllers: Deployments
6. Services
7. Ingress Load Balancing & Network Policy
8. Stateful Workloads

5: Application Controllers: Deployments

Objectives

- Explore the types, roles, and workings of Controllers
  - Deployments, Replica Sets
- Understand labels and selectors and how they are used to identify resources in a Kubernetes cluster
- Understand how to perform application rollouts

Controllers

- Controllers are control loops that watch the cluster's state
- A controller watches and changes a resources' current state to match the desired state
  - e.g. A Pod is deleted, a new pod is created to satisfy current state == desired state
  - e.g. A resource is scaled up or down, pods will be created or terminated to match the desired state

```
$ cat redis-deploy.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 2
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - image: redis
        name: redis
```

```
$ kubectl apply -f redis-deploy.yaml
deployment.apps/redis created

$ kubectl get deploy,rs,pod
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/redis   2/2     2            2           9s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/redis-85d47694f4    2         2         2       9s

NAME                          READY   STATUS    RESTARTS   AGE
pod/redis-85d47694f4-h8l4w    1/1     Running   0          9s
pod/redis-85d47694f4-n7lgv    1/1     Running   0          9s

$ kubectl delete pod redis-85d47694f4-h8l4w
pod "redis-85d47694f4-h8l4w" deleted

$ kubectl get deploy,rs,pod
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/redis   2/2     2            2           35s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/redis-85d47694f4    2         2         2       35s

NAME                          READY   STATUS    RESTARTS   AGE
pod/redis-85d47694f4-hmx78    1/1     Running   0          10s
pod/redis-85d47694f4-n7lgv    1/1     Running   0          35s

$ kubectl scale deploy redis --replicas=3
deployment.apps/redis scaled

$ kubectl get deploy,rs,pod
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/redis   3/3     3            3           54s

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/redis-85d47694f4    3         3         3       54s

NAME                          READY   STATUS    RESTARTS   AGE
pod/redis-85d47694f4-hmx78    1/1     Running   0          29s
pod/redis-85d47694f4-lkh65    1/1     Running   0          4s
pod/redis-85d47694f4-n7lgv    1/1     Running   0          54s
```

Deployment created

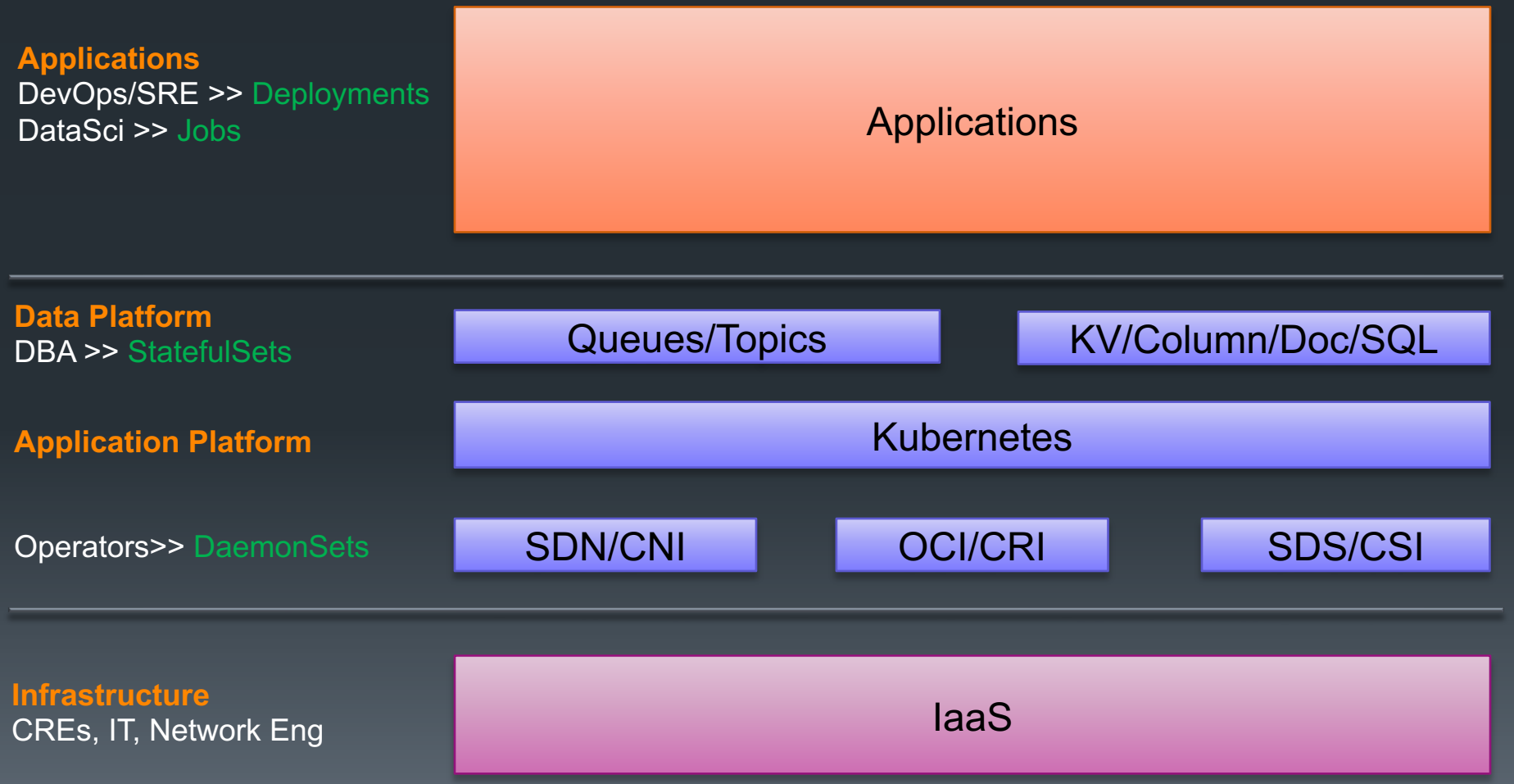
Pod deleted
Pod created to satisfy number of replicas desired

Deployment scaled

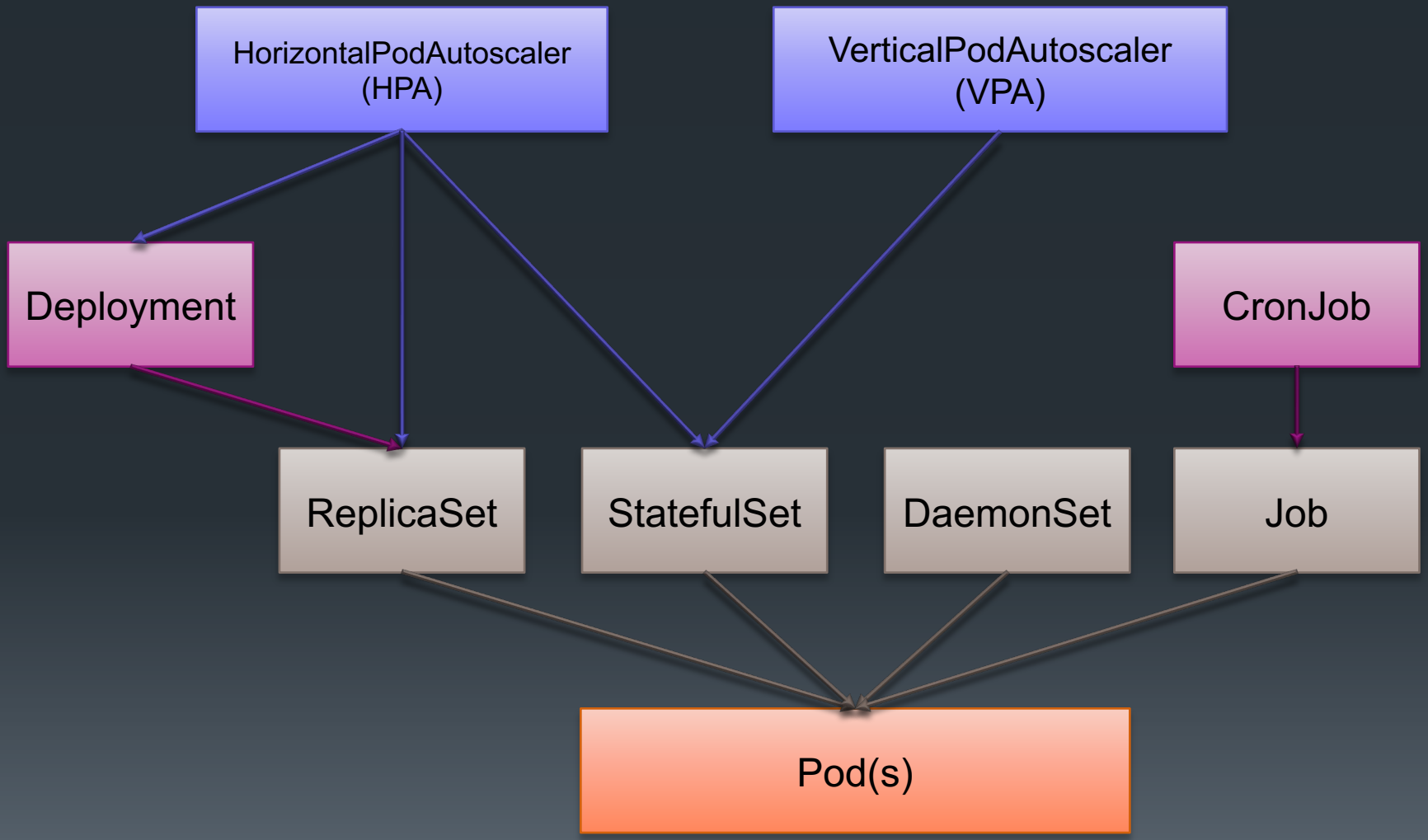
List of Controllers

- For running stateless applications...
  - Deployment - Defines a set and number of pods that need to run
  - ReplicaSet - Runs a specified number of pods using a given pod template
    - Used by deployment to maintain multiple versions of a pod template
  - Job - Runs a number of pods to some completion condition
  - CronJob - Creates Jobs according to a schedule
- For running stateful applications...
  - StatefulSet - Persistence-optimized controller for running pods
- For running Kubernetes-controlled system components...
  - DaemonSet - Runs a single copy of a pod on every node in the cluster
    - Mostly used for cluster agents (e.g. kube-proxy or CNI plugins)
- Non-application related controllers
  - HorizontalPodAutoscaler - Adjust the number of pods run by Deployments or StatefulSets according to metrics
  - Persistent Volume Controller - non user-facing controller that ensures storage requests are fulfilled
  - Certificate Signer - non user-facing controller that signs incoming certificate requests made to the cluster (usually for node additions)
- Kubernetes can be extended to support custom controllers that manage custom resources

Division of Labor in a Cloud Native World



Workload Resource Types



Labels and Annotations

- Labels are key value pairs associated with a resource
  - Any resource can have label metadata
    - pods, nodes, services, namespaces, etc.
  - Resources can be “selected” using labels
  - Label keys and values must consist of letters, numbers, dashes and underbars
- Annotations can contain arbitrary key value data
  - Like labels, annotations are metadata and can be associated with any resource
  - Unlike labels annotations can store values containing any character sequence
  - Not used by Kubernetes label selectors
  - Third party tools often read resource annotations to parameterize their behavior
    - e.g. a CNI plugin may read a pod annotation to decide which network to place the pod on

```
$ kubectl get pod --show-labels
NAME    READY   STATUS    RESTARTS   AGE     LABELS
web     1/1     Running   0          8m35s   team=marketing
$ kubectl label pod web app=configurator
pod/web labeled
$ kubectl get pod --show-labels
NAME    READY   STATUS    RESTARTS   AGE     LABELS
web     1/1     Running   0          9m20s   app=configurator,team=marketing
$ kubectl label pod web team=sales --overwrite=true
pod/web labeled
$ kubectl get pod --show-labels
NAME    READY   STATUS    RESTARTS   AGE     LABELS
web     1/1     Running   0          9m46s   app=configurator,team=sales
$ kubectl label pod web app-
pod/web labeled
$ kubectl get pod --show-labels
NAME    READY   STATUS    RESTARTS   AGE   LABELS
web     1/1     Running   0          10m   team=sales
$ kubectl get pod -l team=sales
NAME    READY   STATUS    RESTARTS   AGE
web     1/1     Running   0          10m
$ kubectl get pod -l team=marketing
No resources found in default namespace.
$

$ kubectl get pods -o custom-columns=NAME:metadata.name,ANNO:.metadata.annotations
NAME    ANNO
web     <none>
$ kubectl annotate pod web web-fwd="using:ingress1##@delivery==priv:repeat^ save"
pod/web annotated
$ kubectl get pods -o custom-columns=NAME:metadata.name,ANNO:.metadata.annotations
NAME    ANNO
web     map[web-fwd:using:ingress1##@delivery==priv:repeat^ save]
$ kubectl annotate pod web web-fwd-
pod/web annotated
$ kubectl get pods -o custom-columns=NAME:metadata.name,ANNO:.metadata.annotations
NAME    ANNO
web     <none>
$
```

Selectors

- There are several options you can use to select on labels:
  - `=` or `==` –select keys with values equal to the string on the right: `name=apache`
  - `!=` – select keys with values that do not equal the string on the right: `environment!=test`
  - `in` – select resources whose labels have keys with values in this set: `'tier in (web,app)'`
  - `notin` – select resources whose labels have keys with values not in this set: `'tier notin (lb,app)'`
  - <Key name> – use a key name only to select resources whose labels contain this key: `tier`
- Multiple selectors can be separated by commas and are logically “AND”ed
  - There is no logical OR operator
- Many kubectl subcommands offer the `-l` switch to specify label selectors (`get`, `delete`, `logs`, etc.)

```
$ kubectl get deploy -l 'team in (scarif,bespin)' --show-labels
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE    LABELS
configurator-web         3/3     3            3           50s    app=configurator,release=stable,team=scarif,tier=frontend
configurator-web-canary  3/3     3            3           84s    app=configurator,release=canary,team=scarif,tier=frontend
rideshare-web            3/3     3            3           119s   app=rideshare,release=stable,team=bespin,tier=frontend
$ kubectl get deploy -l 'team in (scarif,bespin),app=rideshare' --show-labels
rideshare-web   3/3     3            3           2m39s   app=rideshare,release=stable,team=bespin,tier=frontend
$ kubectl logs -l app=rideshare,tier=frontend
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.36.0.3. Set the 'ServerName'
directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.40.0.2. Set the 'ServerName'
directive globally to suppress this message
AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using 10.40.0.3. Set the 'ServerName'
directive globally to suppress this message
$ kubectl delete deploy -l 'team in (scarif,bespin),release=canary'
deployment.apps "configurator-web-canary" deleted
```

Deployment Config

- A Deployment controller provides declarative updates for Pods and ReplicaSets
- You can define Deployments to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments
- Keys related to the Deployment kind
  - **`replicas`** - defines the desired number of pods
  - **`selector`** - specifies a label selector for the Pods targeted by this deployment
    - Tells the Replica Set which pods to watch
  - **`strategy`**
    - `type`: Recreate or RollingUpdate (default)
    - `maxSurge`: max number of pods that can be scheduled above the desired number of pods
    - `maxUnavailable`: max number of pods that can be unavailable during the update
  - **`minReadySeconds`** - optional field that 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
  - **`revisionHistoryLimit`** - optional field that specifies the number of old ReplicaSets to retain to allow rollback
  - **`template`** - defines a template to launch a new pod
    - Contains the same elements defined for pod configs
- Selector values need to match the labels values specified in the Pod template
  - A Deployment's label selector is **immutable** after it gets created

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: k8sapp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: k8sapp
      deployment: development
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  template:
    metadata:
      labels:
        app: k8sapp
        deployment: development
    spec:
      containers:
      - name: k8sapp-container
        image: k8sapp:v3.3
        ports:
        - containerPort: 80
```

Simple Deployment / RS Walk Through


```
$ kubectl apply -f mydep.yaml
deployment.apps/myapp created

$ $ kubectl get deploy,rs,po
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/myapp   3/3     3            3           76s

NAME                               DESIRED   CURRENT   READY   AGE
replicaset.apps/myapp-1072152842   3         3         3       76s

NAME                         READY   STATUS    RESTARTS   AGE
pod/myapp-1072152842-fn5cg   1/1     Running   0          72s
pod/myapp-1072152842-k4pwl   1/1     Running   0          72s
pod/myapp-1072152842-v5wm2   1/1     Running   0          72s

$ kubectl describe deployment myapp
Name:                   myapp
Namespace:              default
CreationTimestamp:      Wed, 01 Aug 2019 12:48:13 -0800
Labels:                 team=kamino,app=myapp,deployment=dev
Annotations:            deployment.kubernetes.io/revision=1
Selector:               app=myapp,deployment=dev
Replicas:               3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:               app=myapp
                        deployment=dev
  Containers:
   myapp-container :
    Image:              logagg
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type                  Status        Reason
  ----                  ------        ------
  Available             True          MinimumReplicasAvailable
 Progressing            True          NewReplicaSetAvailable
OldReplicaSets:         <none>
NewReplicaSet: myapp-1072152842 (3/3 replicas created)
Events:
  FirstSeen     LastSeen   Count  From                    SubObjectPath  Type     Reason            Message
  ---------     --------   -----  ----                    -------------  -------- ------            -------
  3m            3m         1      {deployment-controller}                Normal   ScalingReplicaSet  Scaled up replica set myapp-1072152842 to 3
```

```
$ cat mydep.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp
  labels:
    team: kamino
    app: myapp
    deployment: dev
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
      deployment: dev
  template:
    metadata:
      labels:
        app: myapp
        deployment: dev
    spec:
      containers:
      - name: myapp-container
        image: myapp:v2.8
        ports:
        - containerPort: 80
```

Rollouts

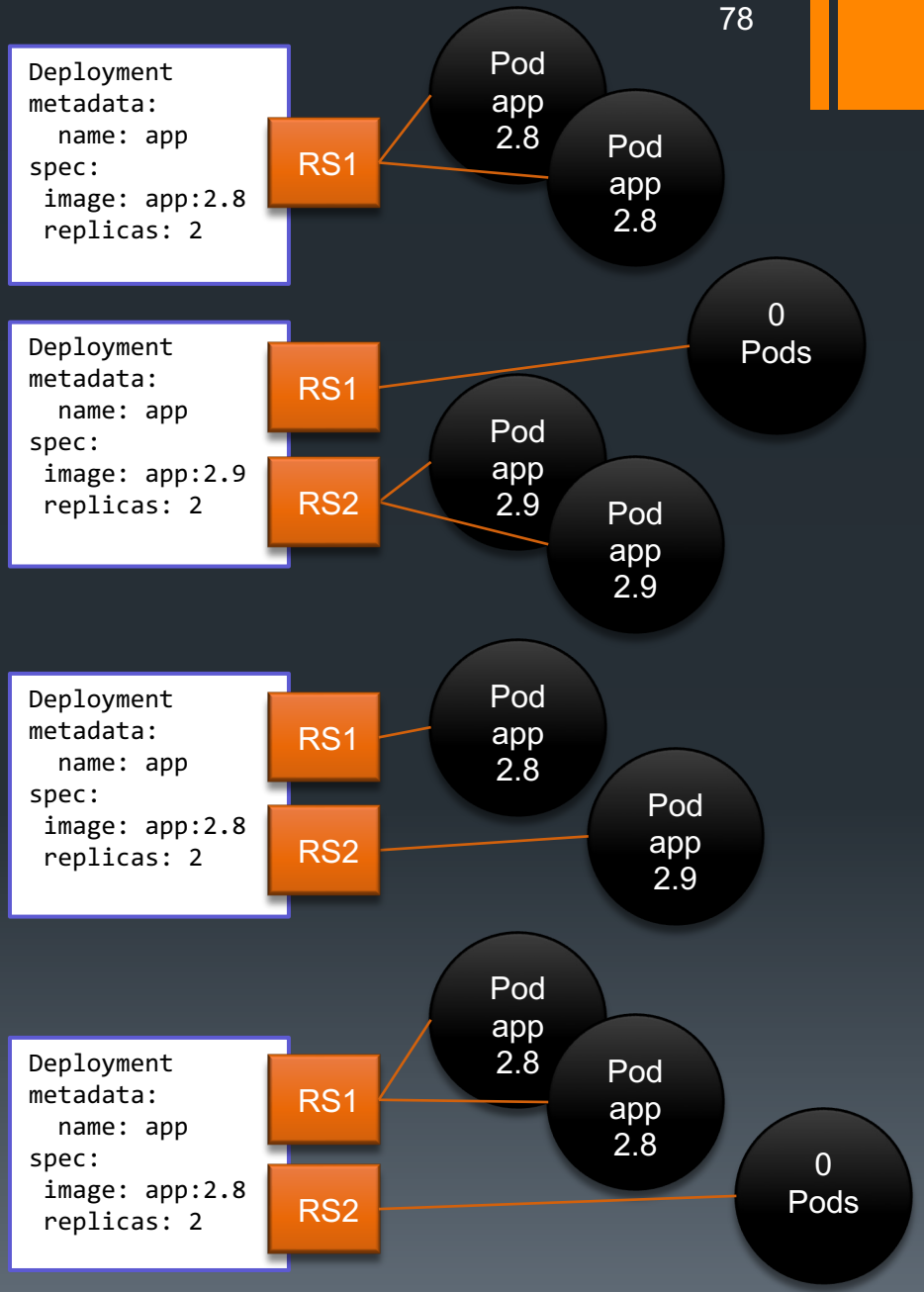
- `kubectl apply -f appdeploy.yaml`

- `kubectl set image deploy/app \ mypod=app:2.9 --record`
  (can also use `edit`, `patch`, or `apply`)

- `kubectl rollout undo deploy/app`
- `kubectl rollout pause deploy/app`

- `kubectl rollout resume deploy/app`

- `kubectl rollout history deploy/app`
- `kubectl rollout status deploy/app`



Summary

- Kubernetes uses sets of key/value pairs called labels to identify resources
  - Some Kubernetes resources use labels to target other resources
- Deployments/ReplicaSets are key components in the orchestration of horizontally-scaled microservices
- RSs use kubelets to perform on-node monitoring and restart tasks

Lab 5

- Deployments, ReplicaSets

6: Services

Objectives

- Explore the nature of Kubernetes services
- Discuss the various types of Services:
  - ClusterIPs, NodePorts, LoadBalancers
- Understand the role of kube-proxy in relation to Services
- Review the IPTables manipulations made by the proxy
- Explain the use of external load balancers with K8s
- Examine Kubernetes DNS

Services

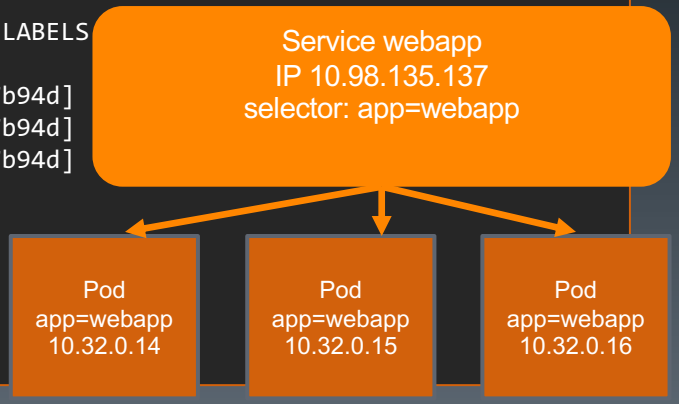
- A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them
  - Pods targeted by a service are identified by a label that matches service's Selector
- Provides a single, stable network endpoint clients use to access groups of pods
  - Pods IPs are unreliable (changing depending on which node they are assigned in a cluster)
  - Services provide a virtual IP and/or cluster-internal DNS name to route traffic to pods

```
$ kubectl get svc -o wide
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   11h   <none>
webapp       ClusterIP   10.98.135.137   <none>        80/TCP    22s   app=webapp

$ kubectl get deploy,rs -o wide
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
deployment.apps/webapp   3/3     3            3           12s   nginx        nginx:latest   app=webapp

NAME                                DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES         SEL
replicaset.apps/webapp-fdcf7b94d    3         3         3       12s   nginx        nginx:latest   app
template-hash=fdcf7b94d

$ kubectl get pod -o=custom-columns='NAME:.metadata.name, IP:.status.podIP, LABELS
NAME                     IP           LABELS
webapp-fdcf7b94d-lnqw5   10.32.0.16   map[app:webapp pod-template-hash:fdcf7b94d]
webapp-fdcf7b94d-lws5m   10.32.0.14   map[app:webapp pod-template-hash:fdcf7b94d]
webapp-fdcf7b94d-xfgpv   10.32.0.15   map[app:webapp pod-template-hash:fdcf7b94d]

$ curl -L http://10.98.135.137
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

webapp service abstracts pods with label app=webapp

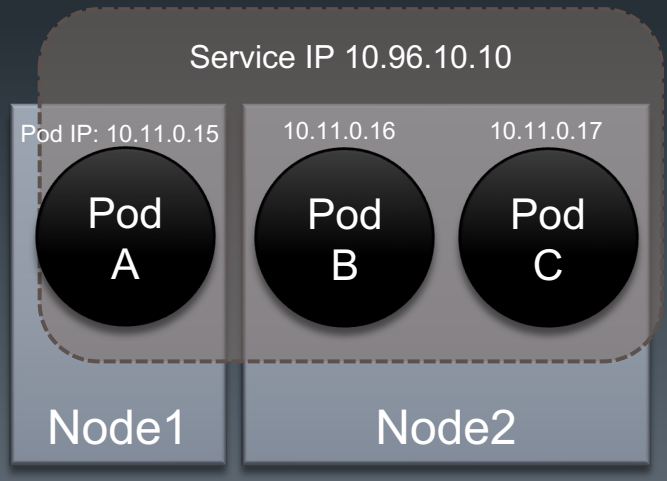
traffic

curl the service IP reaches a webapp pod



Service Types

- Services can be exposed in different ways by specifying a type in the ServiceSpec
  - Each method offers a different level of exposure
- `ClusterIP` – exposes the Service on an internal IP in the cluster
  - Only accessible by containers in the cluster and cluster nodes
- `NodePort` – exposes the Service on the same port of each selected Node in the cluster using NAT
  - Accessible by anybody who has the node address and node port
- `LoadBalancer` – assigns an external address to the Service and LB
  - External address provided by a cloud backend (e.g. AWS ELB) or advertising agent (e.g. MetalLB)
  - Accessible by clients who have the external address
- `ExternalName` – maps a Service to an external DNS name
  - Backs a service IP with an external DNS name rather than Pod IPs
  - Enables apps to use internal Kubernetes DNS names while taking advantage of non-containerized backends and services



Service Config spec

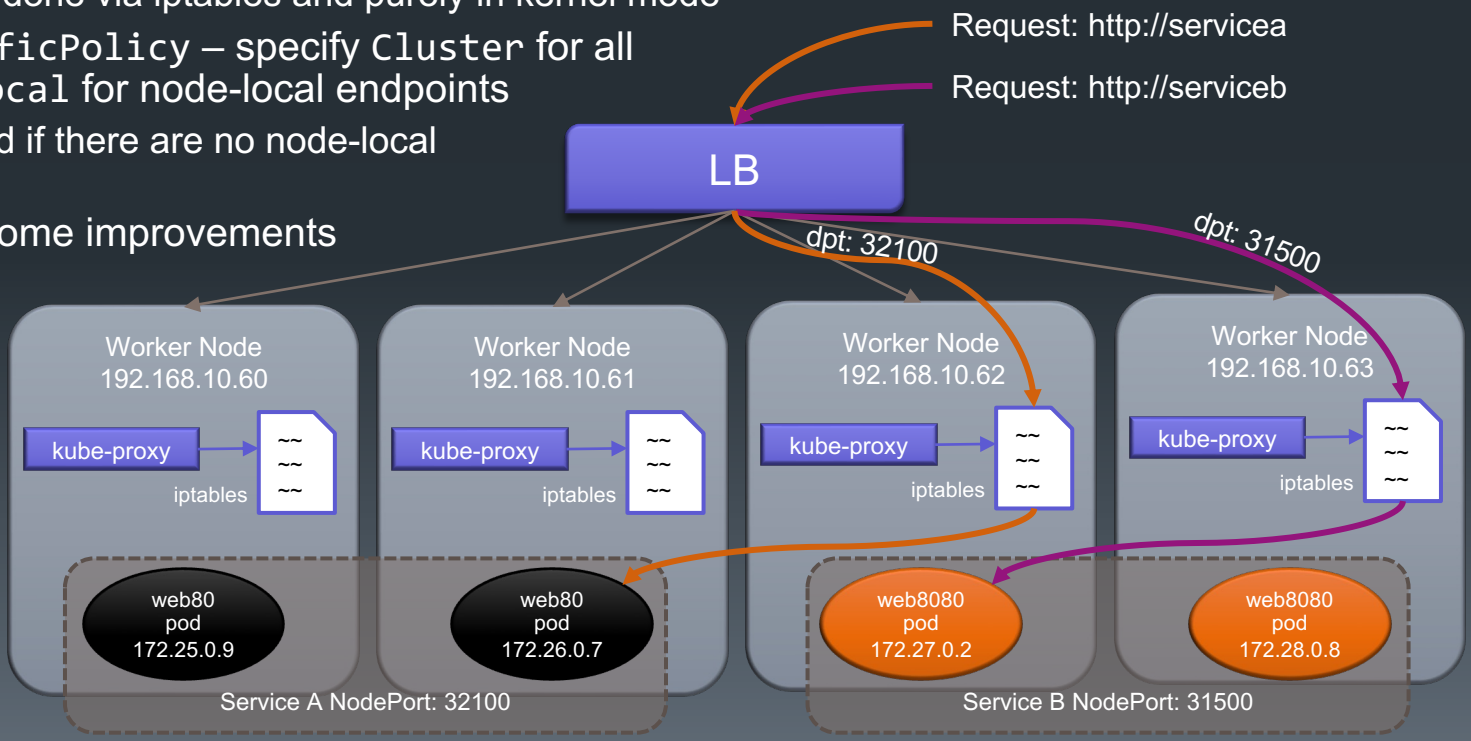
- `type` - type of exposed service
  - `ClusterIP` – [Default] uses kube-proxy and cluster based virtual IP (available only inside the cluster)
  - `NodePort` – exposes the service on the same port on each node of the cluster (for external access)
  - `LoadBalancer` – uses an external load balancer, typically with a public IP and port (AWS ELB, etc.)
- `port` - The list of ports that are exposed by this service
  - Need not be the same as the port of the actual pods
- `targetPort` – The port exposed by the backend Pods
  - By default the targetPort will be set to the same value as the port field
  - Can be a string, referring to the name of a port in the backend Pods
- `selector` - This service will route traffic to pods having labels matching this selector
  - If empty, all pods are selected (!)
  - Pods that qualify for a service's selector are called endpoints
    - Stored inside an `endpoint` resource that shares the name of the service
    - Pods must be in a `Ready` state in order to be included as endpoints for a service
- `clusterIP` - IP is usually assigned by the master and is the IP address of the service
  - If specified, it will be allocated to the service if it is unused or else creation of the service will fail
  - Valid values are `None` (headless service), empty string (""), or a valid IP address
- `sessionAffinity` - Supports "ClientIP" and "None"
  - Used to maintain session affinity
  - Defaults to None

```
apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    app: websvc
spec:
  type: ClusterIP
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
  selector:
    run: testweb
```

```
$ kubectl get endpoints testweb

NAME     ENDPOINTS                                      AGE
testweb  10.32.0.2:80,10.46.0.3:80,10.44.0.6:80  1h
```

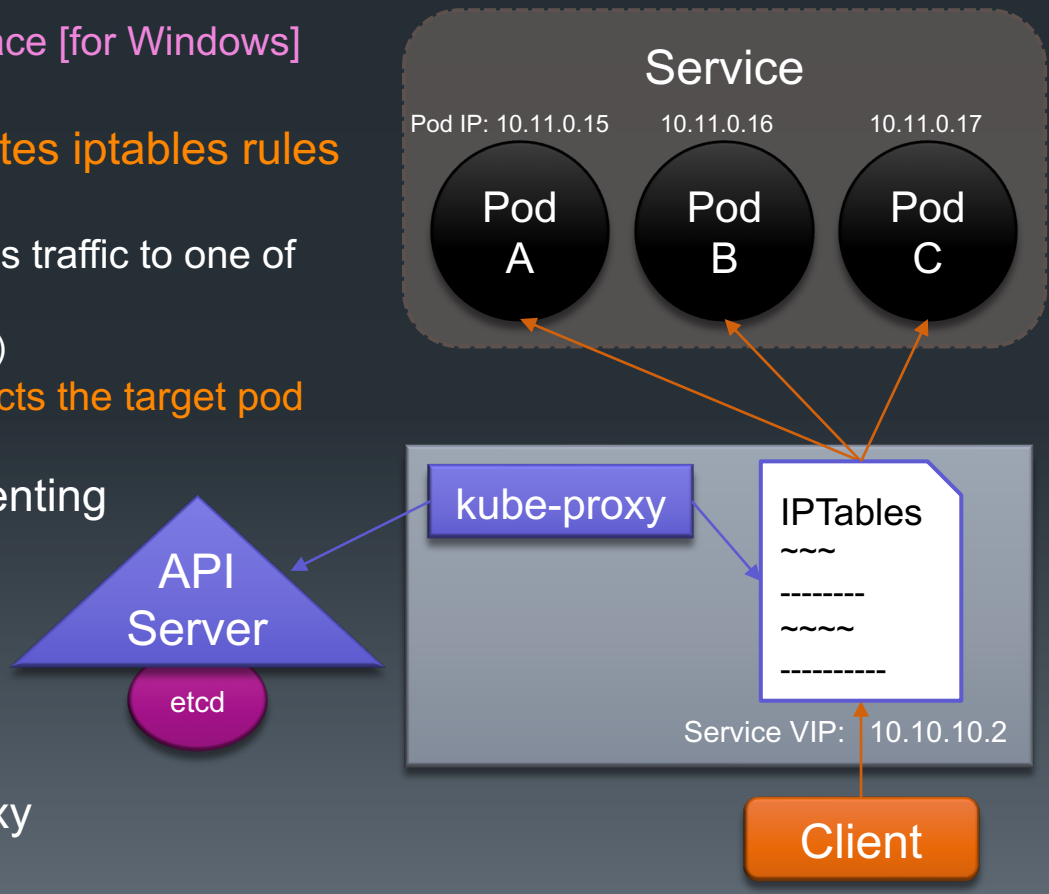
Load Balancers

- LoadBalancer Services wire up cloud load balancers:
  - GCE's ForwardingRules, AWS's ELB (NLB beta as of v1.15), Azure Load Balancer
- LoadBalancer Services outside the cloud can use alpha `loadBalancerClass`
- NodePort Services in combination with an **external load balancer**
- Pod IPs are determined by the SDN (and are thus not real), so the load balancer must rely on Node IP addresses (which are real) and their IP tables for DNAT to route traffic to the pods
  - If IPs are real, `spec.allocateLoadBalancerNodePorts` can be set to `false`
- Most traffic will get double-bounced on the network
  - Nodes essentially act as routers
  - All forwarding is done via iptables and purely in kernel mode
  - `internalTrafficPolicy` – specify `Cluster` for all endpoints or `Local` for node-local endpoints
    - Traffic dropped if there are no node-local endpoints!
- Ingress provides some improvements



Kube-Proxy

- Routing Mesh
  - The kube-proxy usually runs on each node as a pod in the cluster providing a routing mesh to map services to implementation pods
    - Consumes Service and Endpoint API objects
  - The proxy operates in one of 2 modes (`--proxy-mode`):
    - IPTables [default in Linux] / Kernelspace [for Windows]
    - IPVS
  - In IPTables mode Kube-Proxy creates iptables rules for every service
    - Produces a virtual endpoint that routes traffic to one of the service's target pods
      - Typically a ClusterIP, aka Virtual IP (VIP)
    - Iptables loadbalancing randomly selects the target pod to forward to
  - kube-proxy identifies pods implementing the service via a selector
  - All Kube-Proxies watch the API Server for service changes, updating their iptables as needed
  - Some network plugins have components that replace kube-proxy



NodePort & ClusterIP Service


```
$ kubectl create deploy testweb --image=nginx --replicas=3
deployment.apps/testweb created

$ kubectl apply -f svc.yaml
service/testweb created

$ kubectl get deploy,po,rs
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/testweb   3/3     3            3           48s

NAME                               READY   STATUS    RESTARTS   AGE
pod/testweb-58d5cbcdcf-9f7qz   1/1     Running   0          40s
pod/testweb-58d5cbcdcf-9xctw   1/1     Running   0          40s
pod/testweb-58d5cbcdcf-vkq2w   1/1     Running   0          48s

NAME                                     DESIRED   CURRENT   READY   AGE
replicaset.apps/testweb-58d5cbcdcf   3         3         3       48s

$ kubectl get service
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes           ClusterIP   10.96.0.1       <none>        443/TCP        7h6m
testweb              NodePort    10.97.67.39     <none>        80:30320/TCP   49s

$ kubectl get endpoints testweb
NAME      ENDPOINTS                                  AGE
testweb   10.32.0.4:80,10.44.0.5:80,10.39.0.9:80   53s

$ curl -s http://10.97.67.39 | head -4
...
<title>Welcome to nginx!</title>

$ curl -s http://10.32.0.4 | head -4
...
<title>Welcome to nginx!</title>
```

```
$ vim svc.yaml

apiVersion: v1
kind: Service
metadata:
  name: testweb
  labels:
    svc: testweb
spec:
  type: NodePort
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 600
  ports:
  - port: 80
  selector:
    app: testweb
```

NodePort & ClusterIP Service Nat

```
user@ubuntu:~/svc$ sudo iptables -nL -t nat

Chain PREROUTING (policy ACCEPT)
target          prot opt source              destination
KUBE-SERVICES   all  --  0.0.0.0/0           0.0.0.0/0       /* kubernetes service portals */
DOCKER          all  --  0.0.0.0/0           0.0.0.0/0       ADDRTYPE match dst-type LOCAL


Chain KUBE-NODEPORTS (1 references)
target                    prot opt source              destination
KUBE-MARK-MASQ            tcp  --  0.0.0.0/0           0.0.0.0/0     /* default/testweb: */ tcp dpt:30320
KUBE-SVC-EG66NKXDU7X2QZTA tcp  --  0.0.0.0/0           0.0.0.0/0     /* default/testweb: */ tcp dpt:30320


Chain KUBE-SERVICES (2 references)
target                         prot opt source         destination
KUBE-SVC-XGLOHA7QRQ3V22RZ      tcp  --  0.0.0.0/0      10.106.125.172  /* kube-system/kubernetes-dashboard: cluster IP */ tcp dpt:80
KUBE-SVC-NPX46M4PTMTKRN6Y      tcp  --  0.0.0.0/0      10.96.0.1       /* default/kubernetes:https cluster IP */ tcp dpt:443
KUBE-SVC-TCOU7JCQXEZGVUNU      udp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns cluster IP */ udp dpt:53
KUBE-SVC-ERIFXISQEP7F7OF4      tcp  --  0.0.0.0/0      10.96.0.10      /* kube-system/kube-dns:dns-tcp cluster IP */ tcp dpt:53
KUBE-SVC-EG66NKXDU7X2QZTA      tcp  --  0.0.0.0/0      10.97.67.39     /* default/testweb: cluster IP */ tcp dpt:80
KUBE-NODEPORTS                 all  --  0.0.0.0/0      0.0.0.0/0       /* kubernetes service nodeports; NOTE: this must be the last rule in this chain */
ADDRTYPE match dst-type LOCAL


Chain KUBE-SVC-EG66NKXDU7X2QZTA (1 references)
target                    prot opt source      destination
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-7KX2LIG5LIB5UVVO side: source mask: 255.255.255.255
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-TSJSFJG2GUXROSTI side: source mask: 255.255.255.255
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: CHECK seconds: 600 reap name: KUBE-SEP-RY3VYSR3AYX2BBO2 side: source mask: 255.255.255.255
KUBE-SEP-7KX2LIG5LIB5UVVO  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.33332999982
KUBE-SEP-TSJSFJG2GUXROSTI  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */ statistic mode random probability 0.50000000000
KUBE-SEP-RY3VYSR3AYX2BBO2  all  --  0.0.0.0/0   0.0.0.0/0       /* default/testweb: */


Chain KUBE-SEP-7KX2LIG5LIB5UVVO (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.4   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-7KX2LIG5LIB5UVVO side: source mask: 255.255.255.255 tcp to:10.32.0.4:80


Chain KUBE-SEP-TSJSFJG2GUXROSTI (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.5   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-TSJSFJG2GUXROSTI side: source mask: 255.255.255.255 tcp to:10.44.0.5:80


Chain KUBE-SEP-RY3VYSR3AYX2BBO2 (1 references)
target          prot opt source      destination
KUBE-MARK-MASQ  all  --  10.32.0.6   0.0.0.0/0   /* default/testweb: */
DNAT            tcp  --  0.0.0.0/0   0.0.0.0/0   /* default/testweb: */ recent: SET name: KUBE-SEP-RY3VYSR3AYX2BBO2 side: source mask: 255.255.255.255 tcp to:10.39.0.9:80
```

Statistic IPTables module: apply a rule with a probability

IPTables module "recent" enables session affinity

Reverse NAT for hairpin traffic

Services & DNS

- Services defined in a cluster (including the DNS server itself) are resolvable in DNS by name
  - DNS name format: **`serviceName.serviceNamespace.svc.clusterDomain`**
  - A service named "web" in namespace "area51" in a cluster with the suffix "groomlake.local" would produce the FQDN: `web.area51.svc.groomlake.local`
- Pod receive a tailored /etc/resolv.conf configured to use the Cluster DNS with a search list including the Pod's namespace and cluster's domain suffix
- Services within the pod's name space can be resolved directly:
  - In the area51 ns, web would resolve to web.area51.svc.groomlake.local
  - web.area77 (in any ns) would resolve to web.area77.svc.groomlake.local

```
ubuntu@ip-172-31-29-148:~$ kubectl attach -n default -it client
Defaulting container name to client.
Use 'kubectl describe pod/client -n default' to see all of the containers in this pod.
If you don't see a command prompt, try pressing enter.
/ # cat /etc/resolv.conf
nameserver 10.96.0.10
search default.svc.cluster.local svc.cluster.local cluster.local us-west-1.compute.internal
options ndots:5
/ # cat /etc/hostname
client
/ # cat /etc/hosts
# Kubernetes-managed hosts file.
127.0.0.1       localhost
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
fe00::0 ip6-mcastprefix
fe00::1 ip6-allnodes
fe00::2 ip6-allrouters
10.44.0.6       client
/ # ping -c 1 kube-dns.kube-system
PING kube-dns.kube-system (10.96.0.10): 56 data bytes
```

```
ubuntu@ip-172-31-29-148:~$ kubectl get service -n kube-system
NAME       TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
kube-dns   ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   46h
ubuntu@ip-172-31-29-148:~$
```

```
~$ kubectl get pods,svc,deploy -o wide ### The Testing Environment

NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE              NOMINATED NODE   READINESS GATES
pod/dns-test                  1/1     Running   0          47m   10.32.0.7    ip-172-31-3-67    <none>           <none>
pod/website                   1/1     Running   0          48m   10.32.0.6    ip-172-31-3-67    <none>           <none>
pod/website-6c475f8d5c-5pcnp  1/1     Running   0          79s   10.32.0.9    ip-172-31-3-67    <none>           <none>

NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   SELECTOR
service/kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        4d    <none>
service/mysvc        NodePort    10.102.43.253   <none>        80:31741/TCP   44m   run=website

NAME                      READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES   SELECTOR
deployment.apps/website   1/1     1            1           100s  nginx        nginx    run=website
```

DNS availibility

Type	curl	200 OK from a Pod In the same Namespace	200 OK from a Pod in a different Namespace	200 OK from a cluster Node	200 OK From external nodes that can reach the cluster node network
Pod IP	10.32.0.6	Yes	Yes	Yes	No
Standalone Pod DNS (Type A)	10-32-0-6.default.pod.cluster.local	Yes	Yes	No	No
Standalone Pod DNS (Type A)	10-32-0-6.default.pod	Yes	Yes	No	No
Service DNS (Type A)	mysvc.default.svc.cluster.local	Yes	Yes	No	No
Service DNS (Type A)	mysvc.default.svc	Yes	Yes	No	No
Service DNS (Type A)	mysvc.default	Yes	Yes	No	No
Service DNS (Type A)	mysvc	Yes	No	No	No
Service ClusterIP	10.102.43.253	Yes	Yes	Yes	No
Pod created by Deployment exposed by service DNS (Type A)	10-32-0-9.mysvc.default.svc.cluster.local	Yes	Yes	No	No
Pod created by Deployment exposed by service DNS (Type A)	10-32-0-9.mysvc.default.svc	Yes	Yes	No	No
NodePort Service via Node IP	172.31.3.67:31741	Yes	Yes	Yes	Yes
NodePort Service via localhost IP	127.0.0.1:31741	No	No	Yes	No
NodePort Service via Node DNS (Type A)	ip-172-31-3-67.ec2.internal:31741	Yes	Yes	Yes	Yes

Summary

- Services provide an abstraction for delivering traffic to Pods
  - ClusterIP services expose Pods to internal clients only
  - NodePort services expose Pods to external clients and internal client Pods
  - LoadBalancer services expose Pods to Internet, external and internal clients
    - External load balancers, whether LoadBalancer services or manually-configured, typically double-bounce application traffic
- Kubernetes implements a simple Linux native networking model using standards-driven technologies
  - Virtual IPs
  - iptables
  - DNS
  - Etc.
- Kubernetes provides DNS for services and pods

Lab 6

- Services and Service Selectors

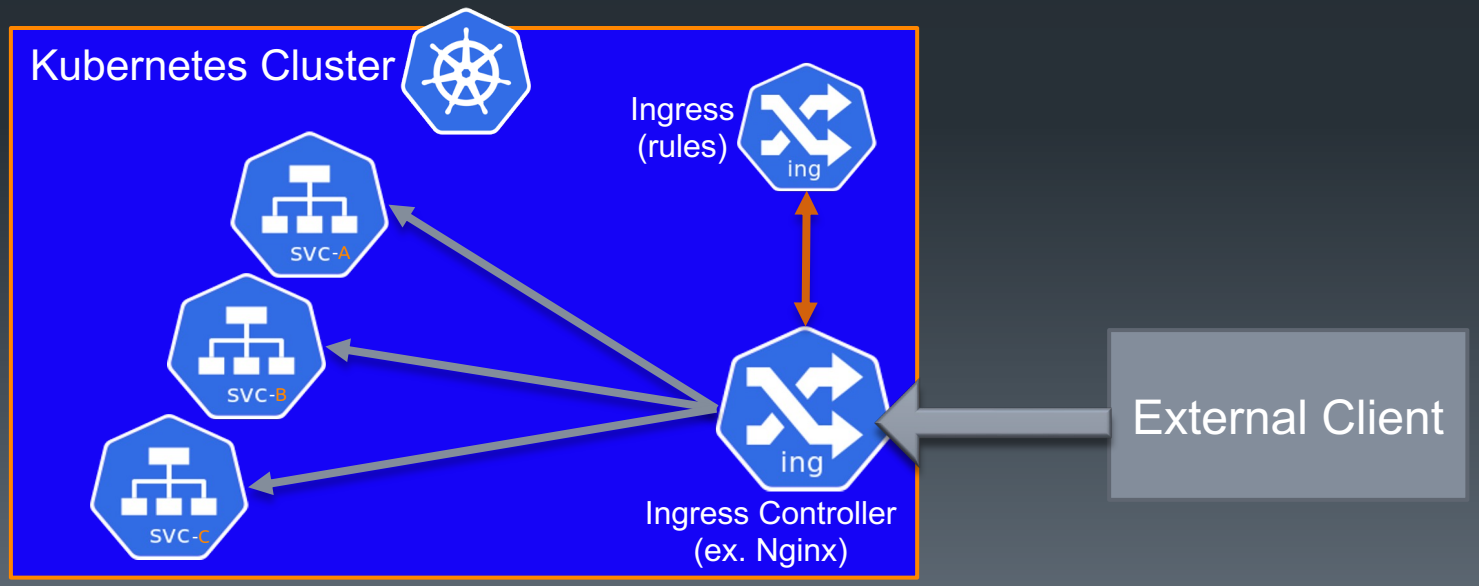
7: Ingress Load Balancing & Network Policy

Objectives

- Describe the architecture of Ingress in Kubernetes
- Understand common Ingress rule patterns
- Learn about policy-based networking
- Discuss various ways to configure network policies

Kubernetes Ingress

- Ingress is a k8s framework used to configure external cluster access via Ingress resources and Ingress Controllers
  - Ingress resources define HTTP and HTTPS routes to clients
  - Rules within ingress resources map routes/headers to services within the cluster
  - Ingress Controllers forward traffic and terminate SSL based on Ingress resource definitions
- Gateway API creates a more general-purpose framework for external cluster access
  - In Beta as of K8s 1.25
  - Gateway resources define listeners with added support for gRPC, UDP and TCP
  - Route resources define routing of traffic from Gateways to services within the K8s cluster
    - e.g HTTPRoute, GRPCRoute, UDPRoute, TCPRoute, TLSRoute (list likely to grow over time)



Ingress Controllers

- Monitor Ingress resources via the Kubernetes API and update the configuration of a load balancer in case of any changes
  - Acts as reverse proxy for the Ingress rules found at api-server's /ingresses endpoint
- Types:
  - GLBC – is a GCE L7 load balancer controller that manages external loadbalancers configured through the Kubernetes Ingress API
  - nginx – deployed as a Deployment/Pod & uses a ConfigMap to store its configuration
  - Non-official: Træfik, others…
- Each Ingress Controller has a different set of features which dictate security options

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-ingress-dep
  namespace: kube-system
  labels:
    app: nginx-ingress
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx-ingress
    spec:
      containers:
      - image: gcr.io/google_containers/nginx-ingress-controller
        name: nginx-ingress
        ports:
        - containerPort: 80
          hostPort: 80
        - containerPort: 443
          hostPort: 443
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress-svc
  namespace: kube-system
  labels:
    app: nginx-ingress
spec:
  ports:
  - name: http
    port: 80
    targetPort: 80
  - name: https
    port: 443
    targetPort: 443
  selector:
    app: nginx-ingress
```

https://docs.nginx.com/nginx-ingress-controller/installation/installation-with-manifests/
contains an example on how to run, configure Nginx Ingress Controller

Ingress (rules)

- Collections of rules that allow ingress connections to reach cluster services
- Ingress resource supports:
  - Exposing services via:
    - URLs
    - Virtual (multiple) host names
  - SSL termination for each exposed host name
- Ingress spec contains a list of rules matched against all incoming requests
  - HTTP rules contains:
    - rules[#]
      - host
        - path[s]
          - pathType – determines how the URL matches
            - backend (service or CRD resource) associated with the paths
- An Ingress controller is required to use an Ingress
  - Simply creating the Ingress resource will have no effect
  - Ingress controllers read Ingress (rules) and update loadbalancers

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
spec:
  rules:
  - host: host.example.com
    http:
      paths:
      - pathType: Prefix
        path: "/somepath"
        backend:
          service:
            name: some-svc
            port:
              number: 80
```

Ingress Rule Types pt. 1

- Single Service Ingress
  - Specify a default backend with no rules

```
# Simple
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: single-ingress
spec:
  defaultBackend:
    service:
      name: singlesvc
      port:
        number: 80
```

```
# Fanout
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: fanout-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: dev.bar.com
    http:
      paths:
      - path: /dev
          pathType: Prefix
          backend:
            service:
              name: service-one
              port:
                number: 80
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service-two
            port:
              number: 8080
```

- Fanout
  - Routes traffic from a single IP address to more than one Service, based on the HTTP URI being requested
  - Keeps the number of load balancers to a minimum

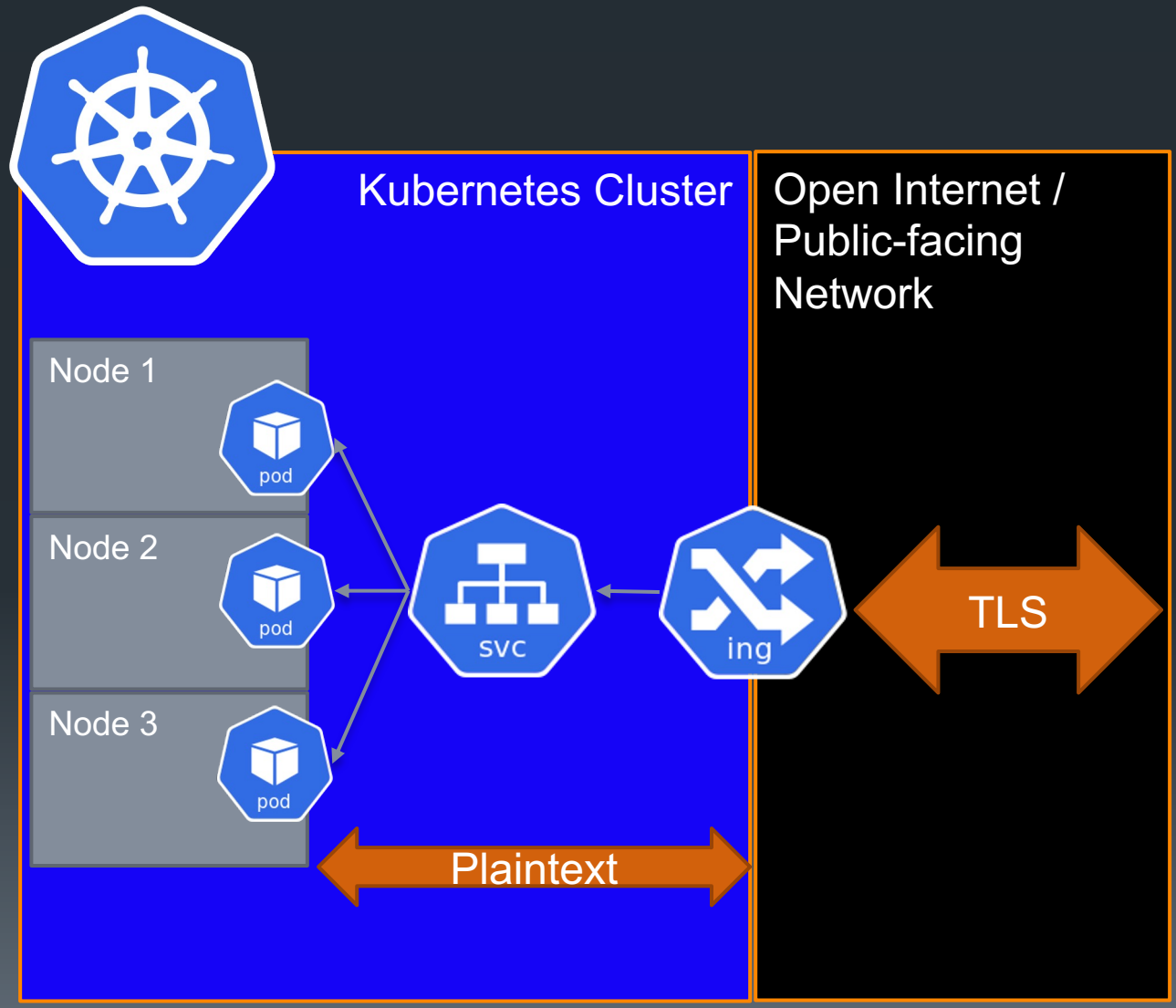
Ingress Rule Types pt. 2

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: rideshare.example.com
    http:
      paths:
      - pathType: Prefix
        path: /driver
        backend:
          service:
            name: driver-svc
            port:
              number: 8080
      - pathType: Prefix
        path: /rider
        backend:
          service:
            name: rider-svc
            port:
              number: 80
  - host: ecommerce.foo.com
    http:
      paths:
      - pathType: Prefix
        path: /cart
        backend:
          service:
            name: cart-svc
            port:
              number: 80
      - pathType: Prefix
        path: /recommender
        backend:
          service:
            name: recommender-svc
            port:
              number: 4444
```

- Name based virtual hosting
  - Supports routing HTTP traffic to multiple host names at the same IP address
  - The load balancer routes requests based on the host header
  - Each hostname can have multiple paths that lead to backend services/pods

TLS Termination

- Allows applications without TLS capabilities to benefit from TLS
- Ingress controllers configure LB TLS by specifying a secret that contains a private key and certificate
- Ingress supports a single TLS port
- Multiple hosts are multiplexed on the same port according to the hostname specified through the SNI TLS extension



Enabling TLS Termination


```
apiVersion: v1
kind: Secret
metadata:
  name: rideshare-tls-secret
  namespace: rideshare-app
type: kubernetes.io/tls
data:
  tls.crt: <base64 encoded cert>
  tls.key: <base64 encoded key>
```

- TLS secrets must contain keys named `tls.crt` and `tls.key` that contain the certificate and private key
- In the example, TLS secret is from a certificate embedded in a K8s Secret that contains a `CN` for `rideshare.example.com`

```
apiVersion: name-virtual-host-ingress
kind: Ingress
metadata:
  name: rideshare-ingress
spec:
  tls:
  - hosts:
    - rideshare.example.com
    secretName: rideshare-tls-secret
  rules:
    - host: rideshare.example.com
      http:
        paths:
        - path: /driver
          pathType: Prefix
          backend:
            service:
              name: driver-svc
              port:
                number: 8080
        - path: /route
          ...
```

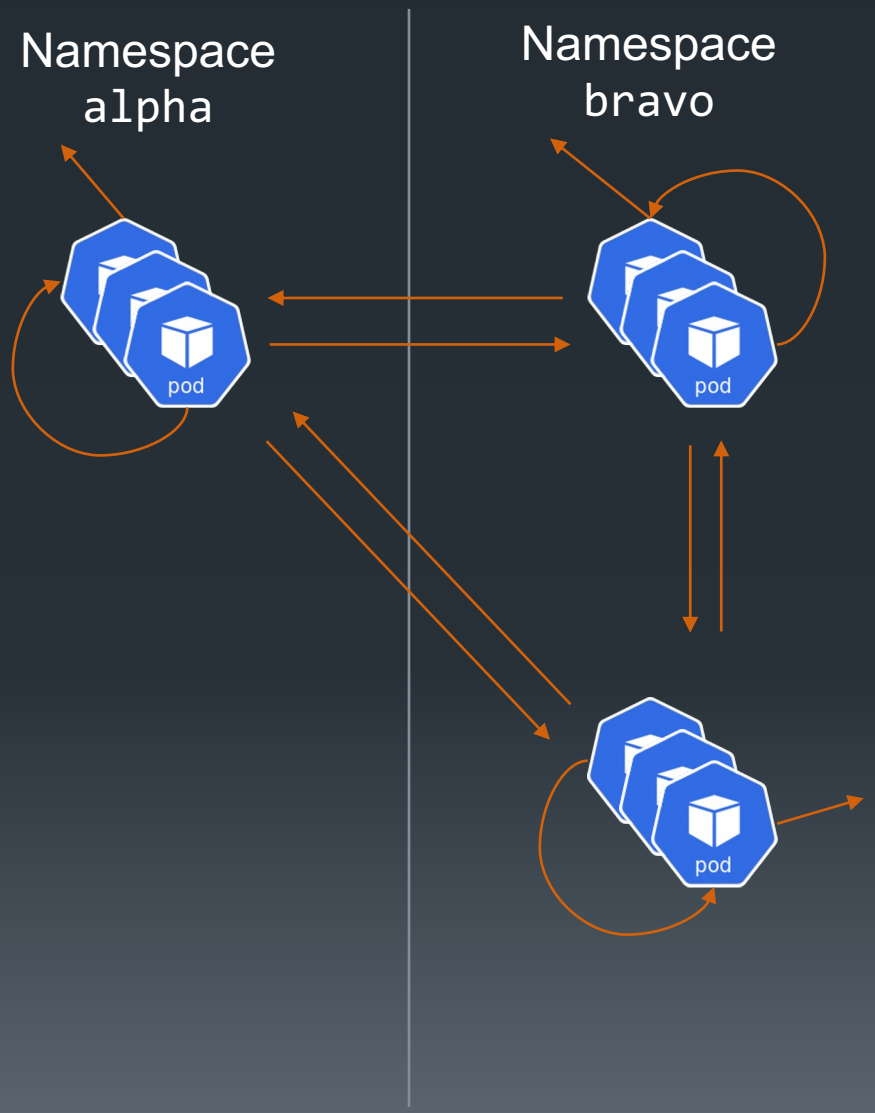
Network Policy

- Specification of how groups of pods are allowed to communicate with each other and other network endpoints
- Creating a policy for a set of labeled pods only restricts traffic for those pods
  - All other pods can still communicate unrestricted
- Use labels to select pods and define rules which specify what traffic is allowed to the selected pods (spec.podSelector)
  - Supports both matchLabels and matchExpression
- A pod will reject any connections that are not allowed by a `NetworkPolicy`
- Policies are additive
  - Eliminates potential for conflicting policies
- Implemented by a network plugin
  - Must use a CNI provider which supports the `NetworkPolicy` object
  - Simply creating the resource without a CNI plugin to implement it will have no effect!

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: example-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: example-app
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          team: kamino
    - podSelector:
        matchLabels:
          app: example-client
    ports:
    - protocol: TCP
      port: 1138
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          team: kamino
    - namespaceSelector:
        matchExpression:
          key: team
          operator: NotIn
          value:
          - daiyu
          - moraband
    ports:
    - protocol: TCP
      port: 1516
```

Without Network Policies

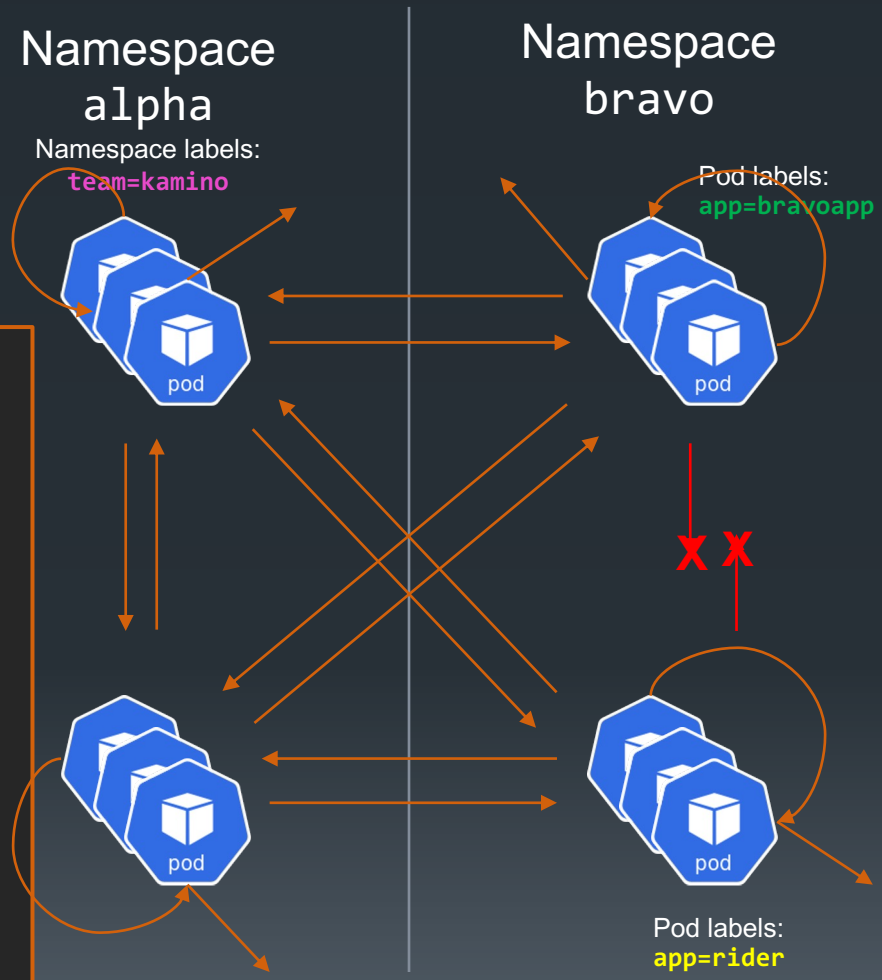
- All pods participate in the same network
  - No way to enforce network partitioning
  - Exceptions are Multus and CNI Genie
    - i.e. multi-nic on separate networks
- By default, a pod is not isolated from other pods on the network
  - Pods accept traffic from any source in the cluster
    - Pods in the any namespace
    - Nodes (control plane and worker)
- By default, a pod is isolated from the outside (things not in k8s cluster)



Example – Ingress and Egress Permissive Networks

- The added egress policy allows traffic to be sent to the example namespace with the given label
- spec.policyTypes.[Ingress, Egress]
  - Sets default behavior if no rules are also set

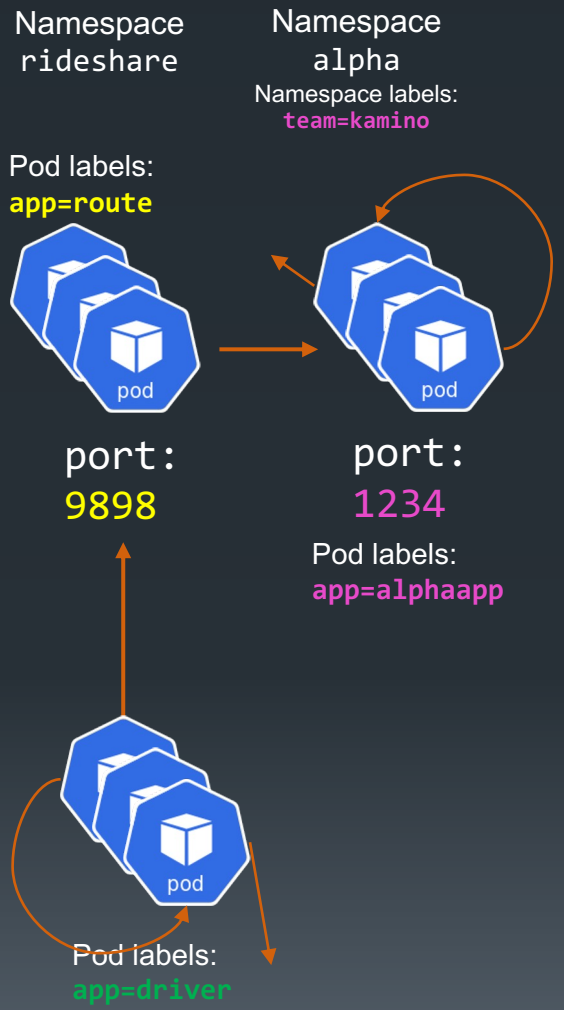
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: namespace-policy
  namespace: bravo
spec:
  podSelector:
    matchLabels:
      app: bravoapp
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          team: kamino
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          team: kamino
```



podSelector and ports

- `podSelector`
  - enables pod-to-pod traffic from pods in the *same namespace*
    - In the example, requests from pods with label `app=driver` in the `rideshare` namespace is allowed
  - Traffic from pods with the same label(s) in a different namespace is dropped
    - Traffic from pods with `app=driver` in any other namespace is dropped
- `ports`
  - ports field requires that the request uses the specified port(s) or traffic is dropped
  - ports can be specified with any type of ingress or egress policy
  - Port ranges supported
    - `networkPolicy.spec.[ingress|egress].ports.endPort` indicates the range of ports from `port` to `endPort`

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: pod-port-policy
  namespace: rideshare
spec:
  podSelector:
    matchLabels:
      app: route
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: driver
    ports:
    - protocol: TCP
      port: 9898
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          team: kamino
      podSelector:
        matchLabels:
          app: alphaapp
    ports:
    - protocol: TCP
      port: 1234
```



Default Deny Everywhere

For every pod in a namespace

`metadata.namespace`
`&&`
`Spec.podSelector`

deny ingress and egress

`policyTypes[Ingress, Egress]`

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-policy
  namespace: alpha
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-policy
  namespace: bravo
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Namespace alpha

Namespace bravo

remember to set all namespaces you want to block

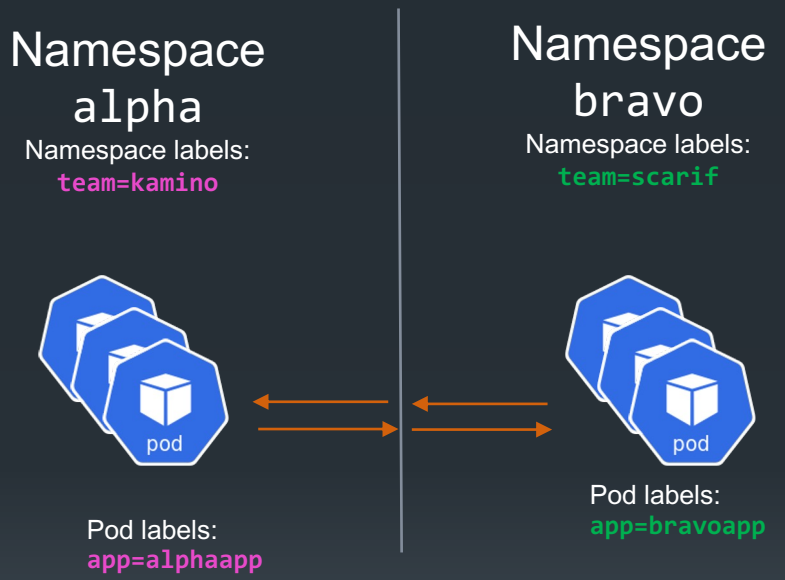
Bidirectional Allow

- Ingress/Egress policies for ”alphaapp" pods *and* "bravoapp" pods are required for bidirectional traffic when default deny policies are in place *in both namespaces*

```
apiVersion: network
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: alpha
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
---
apiVersion: network
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: bravo
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

```
...
  namespace: alpha
spec:
  podSelector:
    matchLabels:
      app: alphaapp
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          team: scarif
      podSelector:
        matchLabels:
          app: bravoapp
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          team: scarif
      podSelector:
        matchLabels:
          app: bravoapp
```

```
...
  namespace: bravo
spec:
  podSelector:
    matchLabels:
      app: bravoapp
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          team: kamino
      podSelector:
        matchLabels:
          app: alphaapp
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          team: kamino
      podSelector:
        matchLabels:
          app: alphaapp
```



Summary

- Ingress controllers implement inbound load balancing tasks for a Kubernetes cluster
- Ingress resources define discrete mappings from ingress routes to Kubernetes cluster IP services
- Network policies enable defining access policies for pods:
  - From other namespaces
  - From other pods in the same namespace
  - From pods in other namespaces

Lab 7

- Network Policy

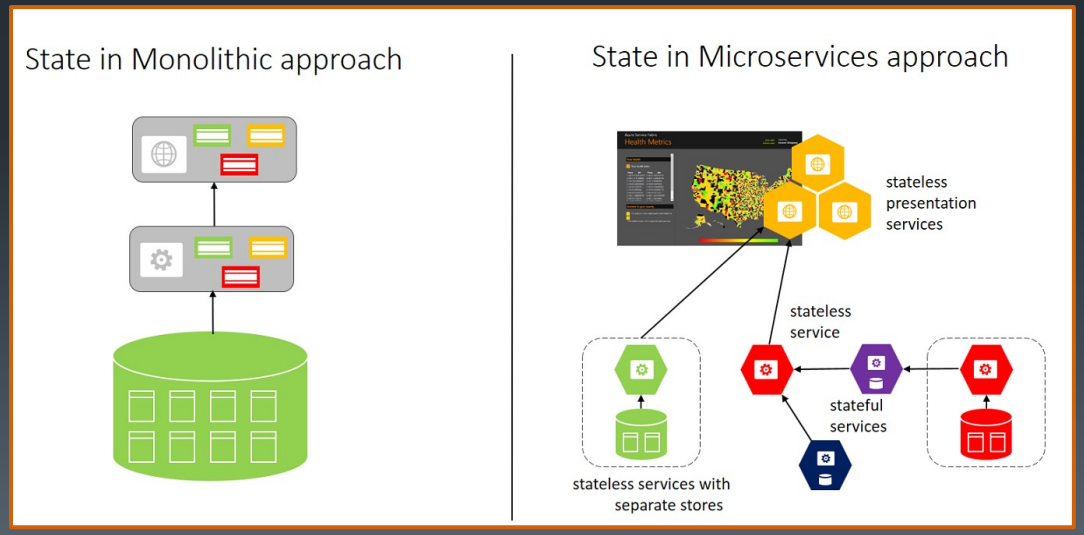
8: Stateful Workloads

Objectives

- Explore the lifecycle of volumes in Kubernetes
- Discover different types of volumes available to Pods and containers in a Kubernetes cluster
- Learn about ways to abstract storage providers
  - Persistent Volumes
  - Persistent Volume Claims
  - Storage Classes
- Examine the StatefulSet controller and its behavior
- Discuss the use of headless services with StatefulSets

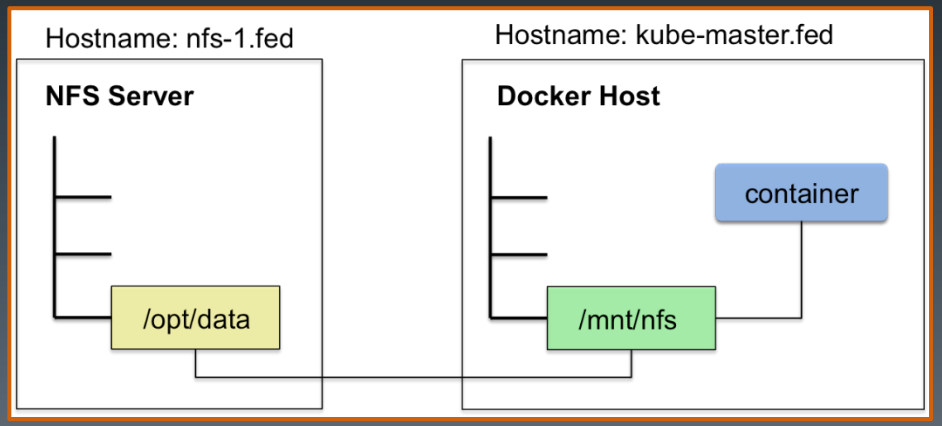
Cloud native state management, patterns and practices

- Cloud native applications divided microservices into 2 camps
  - Stateless
  - Stateful
- Stateful services use volumes to store durable data
- State managers are typically cloud native cluster aware systems
  - Redis, MongoDB, Cassandra, etc.
- State managers are owned by a single service and live inside the bounded context of that service



Kubernetes Volumes

- Cases for volumes:
  - Persist data across container failures
    - kubelet replaces containers by rerunning the original image
      - The files from the dead container are lost
  - Share files between containers running together in a pod
    - Storage outside of the container mount namespace can be shared
  - Map parts of the host's filesystem into the pod's containers
    - For monitoring, storage daemons, etc.
  - Provide configuration and metadata dynamically to a pod
- In all cases, a volume is a feature with storage and persistence rules outside of the scope of the container
- A standard Kubernetes volume has a lifetime the same as the pod that encloses it
  - Different from the Docker volume model, wherein volumes remain until explicitly deleted, regardless of whether there are any running containers using it
- Pods are anchored by the infrastructure container which cannot crash
  - Thus a pod volume outlives any containers that run within the Pod except the pause container preserving volume data across container restarts
  - Only when a Pod is deleted or the node the pod runs on crashes does the volume cease to exist



emptyDir

- One of the easiest ways to achieve improved persistence amid container crashes and data sharing within a pod
- Example uses:
  - Scratch space, such as for a disk-based merge sort
  - Checkpointing a long computation for recovery from crashes
  - Holding files that a sidecar container fetches while a webserver container serves the data
- Created when a Pod is assigned to a Node and is initially empty
- Erased when a Pod is deleted/evicted/removed
- Stored on the medium backing the Node (SSD, network storage, etc.)
  - Specifying `emptyDir.medium` to `Memory` creates a RAM-backed filesystem
    - Cleared on machine reboot
    - Counts against container memory limits for each container that mounts it
  - When `SizeMemoryBackedVolumes` feature gate is enabled:
    - Specify a size for memory backed volumes
    - Otherwise memory backed volumes are sized to 50% of the memory on the node

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
        - name: grafana
          image: grafana/grafana
          ports:
          - name: grafana-port
            containerPort: 3000
          volumeMounts:
            - name: grafana-dashboard-shared-volume
              mountPath: /var/lib/grafana/dashboards/
        ## sidecar - watches all configMaps and places ones
        ##   with a defined label into a shared emptyDir
        - name: grafana-dashboard-sidecar
          image: kiwigrid/k8s-sidecar:latest
          volumeMounts:
            - name: grafana-dashboard-shared-volume
              mountPath: /tmp/dashboards
      volumes:
        - name: grafana-dashboard-shared-volume
          emptyDir: {}
```

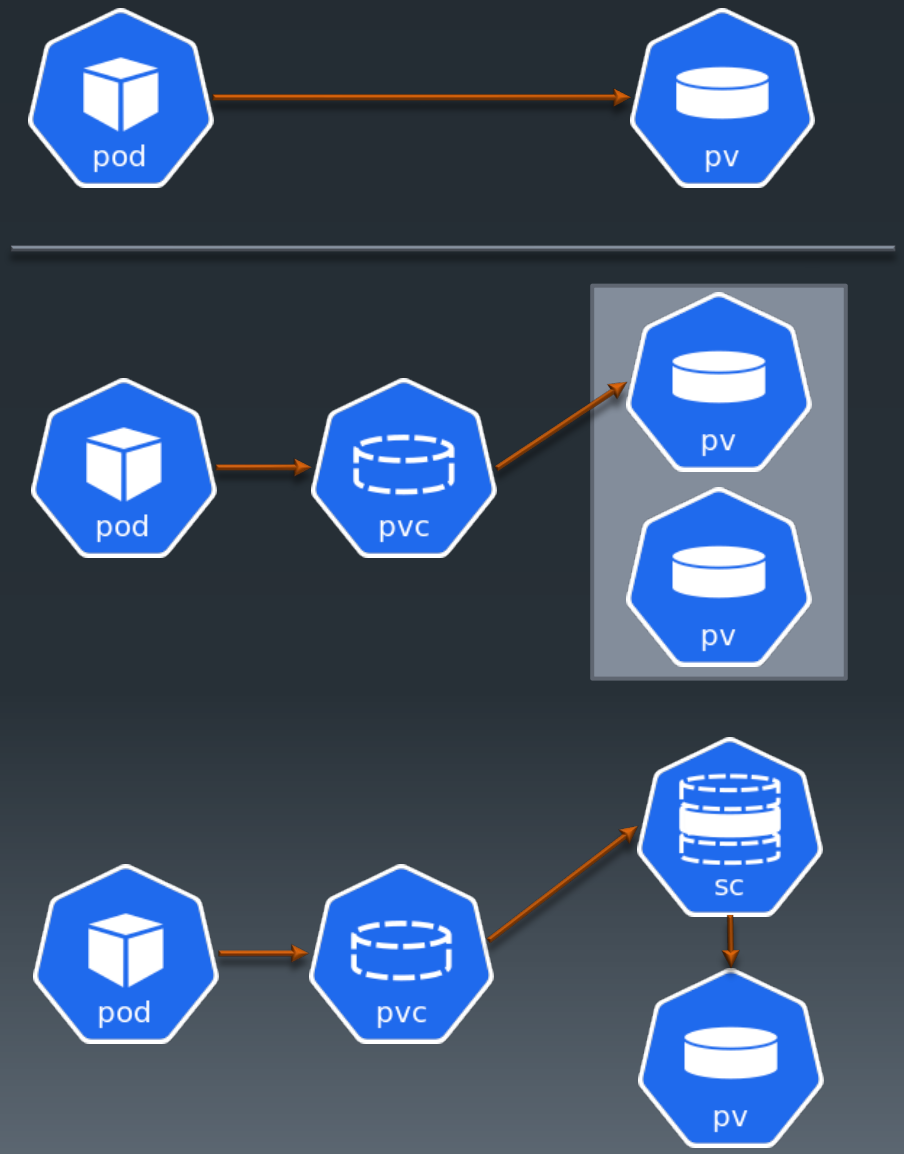
hostPath

- Mounts a file or directory from the Node's filesystem into a Pod
- Pods with identical configs may behave differently on different Nodes due to differences in files on the Nodes
- Resource-aware scheduling will not account for resources used by hostPath
- Created directories only writeable by root
  - Config options:
    - Run process as root in privileged container
    - Modify file perms on the host to write to the volume
- Unlike emptyDir, which is erased when a Pod is removed, the contents of hostPath volumes are preserved and the volume is merely unmounted

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: cadvisor
spec:
  template:
    metadata:
      labels:
        app: cadvisor
    spec:
      containers:
      - name: cadvisor
        image: google/cadvisor
        securityContext:
          privileged: true
        volumeMounts:
          - name: var-run
            mountPath: /var/run
            readOnly: false
          - name: sys
            mountPath: /sys
            readOnly: true
          - name: docker
            mountPath: /var/lib/docker
            readOnly: true
      volumes:
      - name: var-run
        hostPath:
          path: /var/run
      - name: sys
        hostPath:
          path: /sys
      - name: docker
        hostPath:
          path: /var/lib/docker
```

Decoupling Pods from Volumes

- Static pod volume linkage
  - Pods can be bound to specific preexisting volumes
    - This static relationship is simple, useful but often not flexible enough
- Dynamic pod volume linkage
  - In dynamic environments it may be more convenient to bind a Pod to a volume abstraction
  - Persistent Volume Claims [PVC] describe abstract volumes
  - Pods bind to PVCs and PVCs select an appropriate volume at Pod creation time
    - Persistent Volumes [PV]
      - PVCs can be used to select one of a number of suitable admin created Physical Volumes
    - Storage Classes [SC]
      - PVCs can also be used to dynamically allocate volumes using a Storage Class



Kubernetes Pods and Volumes

Create Persistent Volume

```
$ kubectl apply -f fiftymipv.yaml
persistentvolume/fiftymipv created

$ kubectl get pv
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
fiftymipv   50Mi       RWO            Retain           Available                                   3s

$ kubectl apply -f fiftymipv-claim.yaml
persistentvolumeclaim/fiftymipv-claim created

$ kubectl get pv
NAME        CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                     STORAGECLASS   REASON   AGE
fiftymipv   50Mi       RWO            Retain           Bound    default/fiftymipv-claim                  11s

$ kubectl get pvc -o wide
NAME              STATUS   VOLUME      CAPACITY   ACCESS MODES   STORAGECLASS   AGE     VOLUMEMODE
fiftymipv-claim   Bound    fiftymipv   50Mi       RWO                           5m16s   Filesystem

$ kubectl describe pvc fiftymipv-claim | grep Mounted
Mounted By:    <none>

$ kubectl apply -f webserver.yaml
pod/webserver created

$ kubectl describe pvc fiftymipv-claim
Mounted By:    webserver\
```

Create Persistent Volume Claim

- PVC binds to PV

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fiftymipv
spec:
  capacity:
    storage: 50Mi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /web-files
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: webserver
spec:
  containers:
  - name: webserver
    image: httpd
    ports:
    - name: http
      containerPort: 80
    volumeMounts:
    - name: html-files
      mountPath: /var/www/html
  volumes:
  - name: html-files
    persistentVolumeClaim:
      claimName: fiftymipv-claim
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fiftymipv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Mi
```

Storage Classes

- Admins set the name and other parameters of a class when first creating StorageClass objects
  - Cannot be updated once they are created
- `provisioner` – determines what volume plugin is used for provisioning PVs
  - ex: GCEPersistentDisk, AWSElasticBlockStore, VsphereVolume, etc.
  - Internal provisioners prefixed with `kubernetes.io` and shipped with Kubernetes
  - External provisioners are independent programs that follow a specification defined by Kubernetes
    - ex: NFS doesn't provide an internal provisioner, but an external provisioner can be used
- `parameters` – describe attributes of volumes belonging to the storage class
  - Are provisioner specific though some may be similar
    - Best to examine docs of a given provisioner for details on parameters
- Default StorageClass can be specified for PVCs that don't request any particular class to bind to

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gce-pd
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  zones: us-central1-a, us-central1-b
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: aws-ebs
provisioner: ebs.csi.aws.com
parameters:
  csi.storage.k8s.io/fstype: ext4
  type: gp2
```

Persistent Volumes

- Provisioning
  - Static – cluster admin creates a number of PVs
    - Include the details of the real storage & exist as resources in the API
  - Dynamic – PersistentVolume Provisioner may try to provision a PV when none of the static PVs matches a user's PVC
- Capacity – PVs will have a storage capacity set using the PV's `capacity` attribute

- Access Modes – can be set to modes supported by a given provider
  - `ReadWriteOnce` (RWO) – can be mounted as read-write by a single node
  - `ReadWriteOncePod` (RWOP) - can be mounted as read-write by a single Pod
  - `ReadOnlyMany` (ROX) – can be mounted read-only by many nodes
  - `ReadWriteMany` (RWX) – can be mounted as read-write by many nodes

  - PV can only be mounted using one access mode at a time, even if it supports many

- Reclaim policy for a PV tells the cluster what to do with the volume after it has been released of its claim, can be:
  - Retained – previous claimant's data remains on the volume until cleaned or the PV is deleted
  - Recycled (deprecated) – performs a scrub (`rm -rf /vol/*`) on the volume and makes it available again for a new claim
  - Deleted – removes both the PV object as well as deleting the associated external storage, such as an AWS EBS, GCE PD, Azure Disk, or Cinder volume
    - Dynamically provisioned PVs default to Delete

- Class (optional) – ensures that the PV will only be bound to PVCs requesting the given Storage Class

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: fiftygigpv
spec:
  capacity:
    storage: 50Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: bronze
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Persistent Volume Claims

- Created with a specific amount of storage and an access mode:
  - `resources` – PVCs request quantities of storage
  - `accessModes` – PVCs use the same conventions as PVs
    - Mounting claims with "Many" modes (ROX, RWX) is only possible within one namespace

- `storageClassName` (optional) – selects only PVs of the given class

- `selector` (optional) – enables granular filtering of available PVs
  - Only volumes with labels that match the selector can be bound to the claim
  - `matchLabels` – volume must have a label with this value
  - `matchExpressions` – list of requirements made by specifying key, a list of values, and an operator that relates the key and values
    - Valid operators: `In`, `NotIn`, `Exists`, `DoesNotExist`
  - Requirements are ANDed together—all must be satisfied to match

- Control loop matches new PVCs with PVs
  - A dynamically-provisioned PV will *always* bind to the PVC that requested it
- Will remain unbound indefinitely if a matching PV does not exist
  - Cluster with many 50Gi PVs would not match any to a 100Gi PVC
- PVCs have *beta* support for online resizing as of v1.15
- PVCs have support for cloning when PVs are backed by CSI drivers

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fiftygigpvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 50Gi
  storageClassName: bronze
  selector:
    matchLabels:
      type: "cloudvol"
    matchExpressions:
    - {key: environment, operator: In, values: [dev,test]}
```

Pods and PVCs

- Pods bind to PVs through PVCs in the same way that they use normal volumes
  - A volume is identified as a PVC using the `persistentVolumeClaim` key
  - PVCs are identified by `claimName`

```
apiVersion: v1
kind: Pod
metadata:
  name: frontendpod
spec:
  containers:
    - name: myfrontend
      image: nginx
      volumeMounts:
      - mountPath: "/var/www/html"
        name: abstractedstorage
  volumes:
    - name: abstractedstorage
      persistentVolumeClaim:
        claimName: fiftygigpvc
```

Generic Ephemeral Volumes

- Similar to emptyDir but backed by PVCs/PVs
  - All PVC fields supported
  - PVC deleted when pod is deleted
  - PV deleted according to reclaim policy
- Can be local or network-attached
- Can have a fixed size
- Supports features of PVCs/PVs when the driver supports them:
  - Snapshotting, cloning, resizing
- Name of auto-generated PVC is: `<pod_name>-<volume_name>`

```
apiVersion: v1
kind: Pod
metadata:
  name: ephemeral-scratch-pod
spec:
  containers:
    - name: frontend
      image: nginx
      volumeMounts:
      - mountPath: "/scratch"
        name: scratch
  volumes:
    - name: scratch
      ephemeral:
        volumeClaimTemplate:
          metadata:
            labels:
              type: frontend-vol
          spec:
            accessModes: [ "ReadWriteOnce" ]
            storageClassName: scratch-class
            resources:
              requests:
                storage: 5Gi
```

CSI Dynamic Provisioning


```
~$ nano ebs-sc.yaml && cat ebs-sc.yaml

kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
parameters:
  csi.storage.k8s.io/fstype: ext4
  type: gp2

~$ kubectl apply -f ebs-sc.yaml

storageclass.storage.k8s.io/ebs-sc created

~$ nano dynamic-pvc.yaml && cat dynamic-pvc.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: dynamic-pvc
spec:
  storageClassName: ebs-sc
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi

~$ kubectl apply -f dynamic-pvc.yaml

persistentvolumeclaim/dynamic-pvc created

~$ kubectl get pvc

NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
dynamic-pvc   Bound    pvc-82deee92-85d3-40b7-be95-982985a4a843   1Gi        RWO            ebs-sc         10s

~$ kubectl get pv

NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   AGE
pvc-82deee92-85d3-40b7-be95-982985a4a843   1Gi        RWO            Delete           Bound    default/dynamic-pvc   ebs-sc         29s
```

Admin creates StorageClass that points to provisioner, the CSI plugin

Dynamic provisioning is triggered by a PVC using the StorageClass

Volume ID	vol-05ad6da212cd3d73c	Outposts ARN	-
Alarm status	*None*	Size	1 GiB
Snapshot	-	Created	October 20, 2020 at 8:28:27 AM UTC-7
Availability Zone	us-east-1f	State	available
Encryption	Not Encrypted	Attachment information	
KMS Key ID		Volume type	gp2
KMS Key Aliases		Product codes	-
KMS Key ARN		IOPS	100
Multi-Attach Enabled	No		

Provisioner creates a corresponding volume on the backend

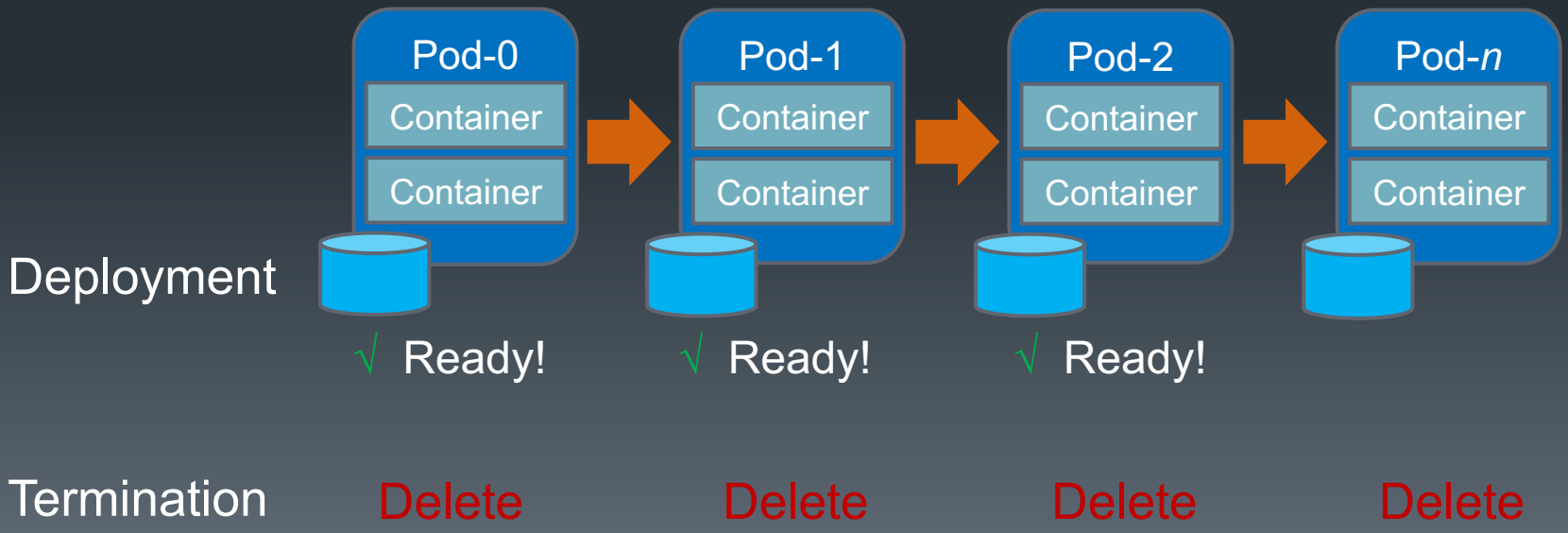
StatefulSets

- Pods have a unique ordinal index (thus stable network identity)
  - Each Pod derives its hostname from the name of the StatefulSet and the ordinal of the Pod
  - StatefulSet “web” would have pods: web-0,web-1, web-2, …
- StatefulSet Pods have Stable persistent storage – one PV for each VolumeClaimTemplate
  - PVs associated with the Pods’ PVCs are not deleted when the Pods, or StatefulSet are deleted & must be removed manually
  - Name of auto-generated PVC is: `<volume_name>-<pod_name>`

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: cassandra
spec:
  serviceName: cassandra
  replicas: 6
  selector:
    matchLabels:
      app: cassandra
  template:
    metadata:
      labels:
        app: cassandra
    spec:
      containers:
      - name: cassandra
        image: cassandra:3.11
        ports:
        - name: cql
          containerPort: 9042
        volumeMounts:
        - name: data
          mountPath: /cassandra/data/
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: local
      resources:
        requests:
          storage: 50Gi
```

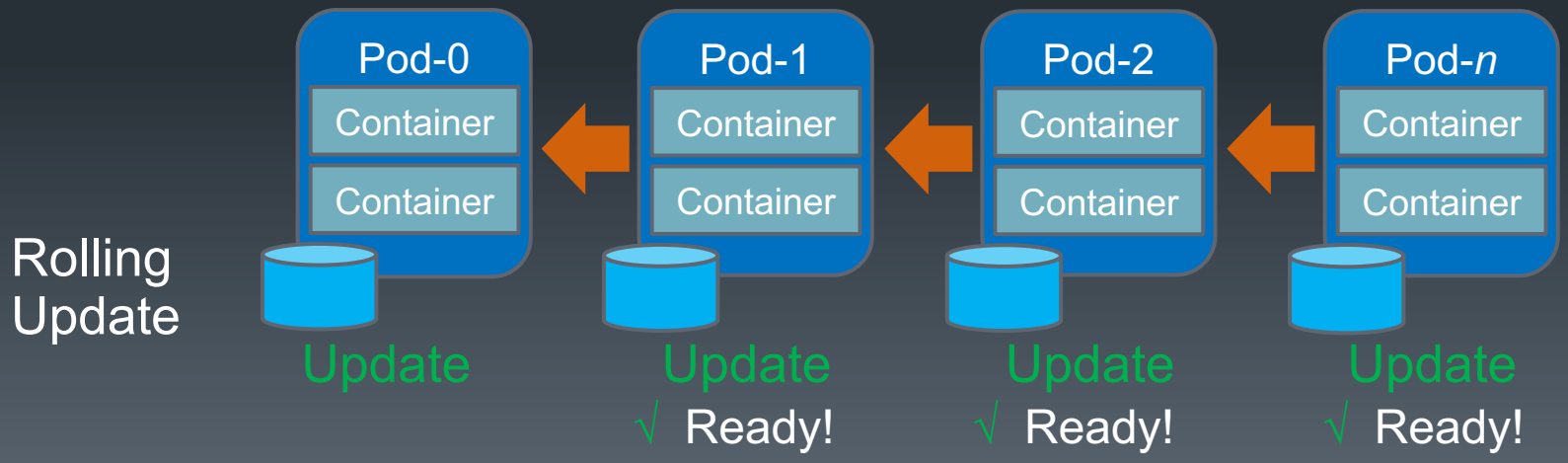
StatefulSet Deployments

- Pod ordinals provide guarantees about ordering
- `podManagementPolicy` determines ordering:
  - `OrderedReady` – ordinal 0 must be "running and ready" prior to the following pods start
  - `Parallel` – launch or terminate all Pods in parallel (like the Deployment controller)
- No guarantee during termination when a StatefulSet is deleted (workaround: scale to 0 first!)



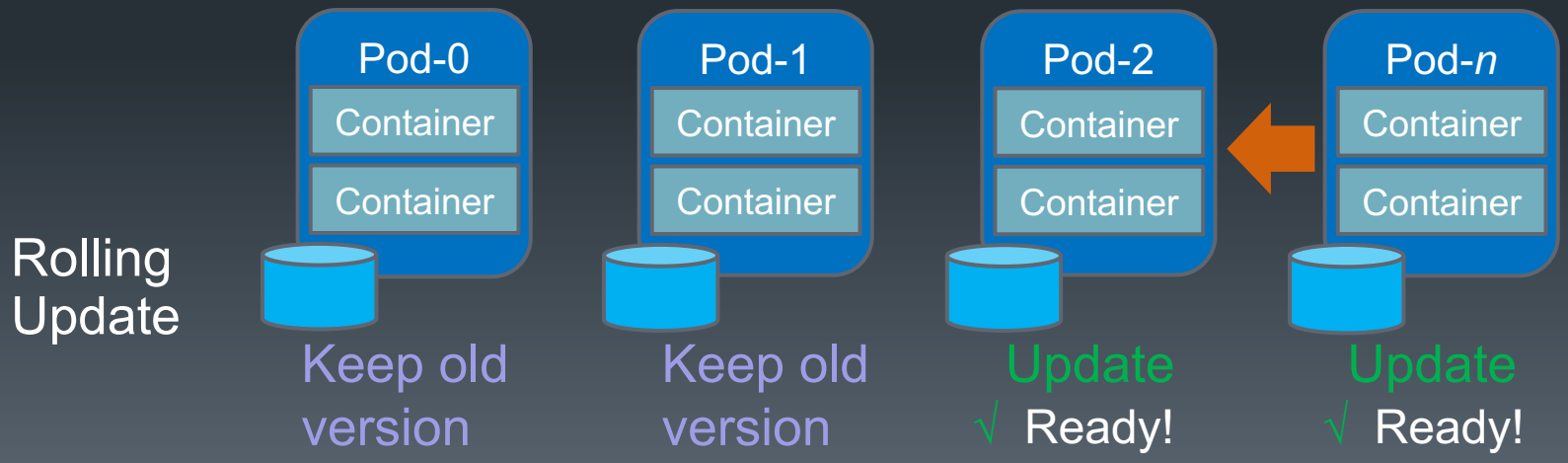
StatefulSet Updates

- Pods are deleted and recreated/replaced on the same node
- Eliminates the need to detach/attach network volumes from/to Nodes
- Strategies:
  - `OnDelete` – users manually delete Pods to cause the controller to create new Pods reflecting modifications made to the template
  - `RollingUpdate` (default) – deletes and recreates each Pod in the same order as Pod termination
    - Waits until an updated Pod is Running and Ready prior to updating its predecessor
- Forced rollbacks sometimes necessary
  - If a Pod never reaches Ready state, reverting the change will not rectify the Pod, it must be manually deleted to force the rollback



Partitioned Updates

- Updates can be partitioned, by specifying `.spec.updateStrategy.rollingUpdate.partition`
  - Pods with an ordinal that >= to the partition number will be updated
  - Pods with an ordinal that is < the partition will not be updated—even if deleted!
- Enables staged updates, canary, or phased roll outs



Headless Services

- Services with `.spec.clusterIP` set to `None` will not have a cluster IP allocated with no load balancing and no proxying
- DNS is automatically configured for headless Services
- DNS configuration differs if selectors are defined
  - Headless services with selectors:
    - DNS is configured to return Endpoint records that point directly to the Pods of the Service. The endpoint controller creates the Endpoint records in the API
  - Headless services without selectors:
    - No Endpoint records are created but DNS looks for and configures for either CNAME records for ExternalName type of Services or for records of any Endpoints that share a name with the Service
- Typically used with StatefulSets
  - Option for client-based routing when L3 load balancing is inadequate
  - Provides fast initial IP identification via DNS
    - The 'smart-client' uses that coupled with app specific information for potentially better decision making
  - Knowing a DNS endpoint where the IPs can change is better for long term survival

```
# Headless service
apiVersion: v1
kind: Service
metadata:
  name: cassandra
  labels:
    app: cassandra
spec:
  ports:
  - name: cql
    port: 9042
  clusterIP: None
  selector:
    app: cassandra
```

Summary

- Kubernetes volumes have the same lifetime as the pods that declare them
  - Kubernetes volumes are not exactly the same as Docker volumes
- Kubernetes supports many volume types & providers
- Kubernetes provides powerful storage abstraction using PVs, PVCs & StorageClasses
  - The PersistentVolume subsystem decouples storage from the pod lifecycle
- StatefulSets leverage abstracted storage for stateful applications
  - StatefulSet rollouts and rollbacks have features suited to clustered state stores
- Headless services enable clustered state stores to communicate

Lab 8

- Volumes, Persistent Volumes, Persistent Volume Claims, StatefulSets

The End

Many thanks for attending!

rx-m cloud native training & consulting