



# Certified Kubernetes Application Developer (CKAD) Crash Course

*Kubernetes 1.26 Edition*



# About the trainer



**bmuschko**



**bmuschko**



**bmuschko.com**



 **AUTOMATED  
ASCENT**  
[automatedascent.com](http://automatedascent.com)

O'REILLY®

# Certified Kubernetes Application Developer (CKAD) Study Guide

In-Depth Guidance and Practice



Benjamin Muschko

*Companion study guide with  
practice questions*

**Released in February 2021**

Online access on O'Reilly  
learning platform:

[https://learning.oreilly.com/library/view/  
certified-kubernetes-application/9781  
492083726/](https://learning.oreilly.com/library/view/certified-kubernetes-application/9781492083726/)

# *CKAD exam practice test*

OREILLY®

## 2-1. CKAD Pods: Creating and Interacting with a Pod

◀ Step 2 of 4 ▶

### Creating a Pod

Create a new Pod named `nginx` running the image `nginx:1.17.10`.

Expose container port 80.

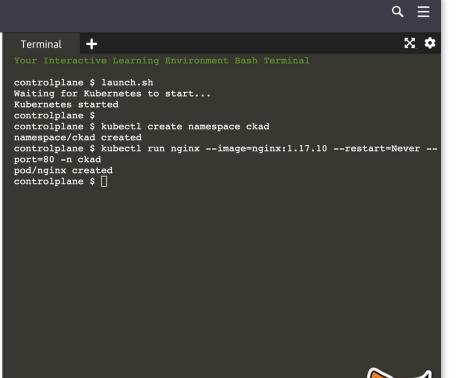
The Pod should live in the namespace `ckad`.

Wait until the Pod transitions into the "Running" status.

[Solution](#)

[Continue](#)

controlplane \$ launch.sh  
Waiting for Kubernetes to start...  
Kubernetes started  
controlplane \$  
controlplane \$ kubectl create namespace ckad  
namespace/ckad created  
controlplane \$ kubectl run nginx --image=nginx:1.17.10 --restart=Never --  
port=80 -n ckad  
pod/nginx created  
controlplane \$



## Katacoda labs

Online access on O'Reilly learning platform:

<https://learning.oreilly.com/playlists/8aa87dce-f9a9-4206-83af-c8c730fa430/>

# Exam Details and Resources

Objectives, Environment, Time Management

# Exam Objectives

*“Design, build, configure, and expose  
cloud native applications for Kubernetes”*



**The certification program allows users to demonstrate their competence in a hands-on, command-line environment.**

<https://www.cncf.io/certification/ckad/>



# The Curriculum

## 20% - Application Design and Build

- Define, build and modify container images
- Understand Jobs and CronJobs
- Understand multi-container Pod design patterns (e.g. sidecar, init and others)
- Utilize persistent and ephemeral volumes

## 20% - Application Deployment

- Use Kubernetes primitives to implement common deployment strategies (e.g. blue/green or canary)
- Understand Deployments and how to perform rolling updates
- Use the Helm package manager to deploy existing packages

## 15% - Application observability and maintenance

- Understand API deprecations
- Implement probes and health checks
- Use provided tools to monitor Kubernetes applications
- Utilize container logs
- Debugging in Kubernetes

## 20% - Services & Networking

- Demonstrate basic understanding of NetworkPolicies
- Provide and troubleshoot access to applications via services
- Use Ingress rules to expose applications

## 25% - Application Environment, Configuration and Security

- Discover and use resources that extend Kubernetes (CRD)
- Understand authentication, authorization and admission control
- Understanding and defining resource requirements, limits and quotas
- Understand ConfigMaps
- Create & consume Secrets
- Understand ServiceAccounts
- Understand SecurityContexts



# Candidate Skills



**kubernetes**

Architecture & Concepts



`kubectl`

Running Commands



container runtime

Underlying Concepts



# Exam Environment

*Online and proctored exam*

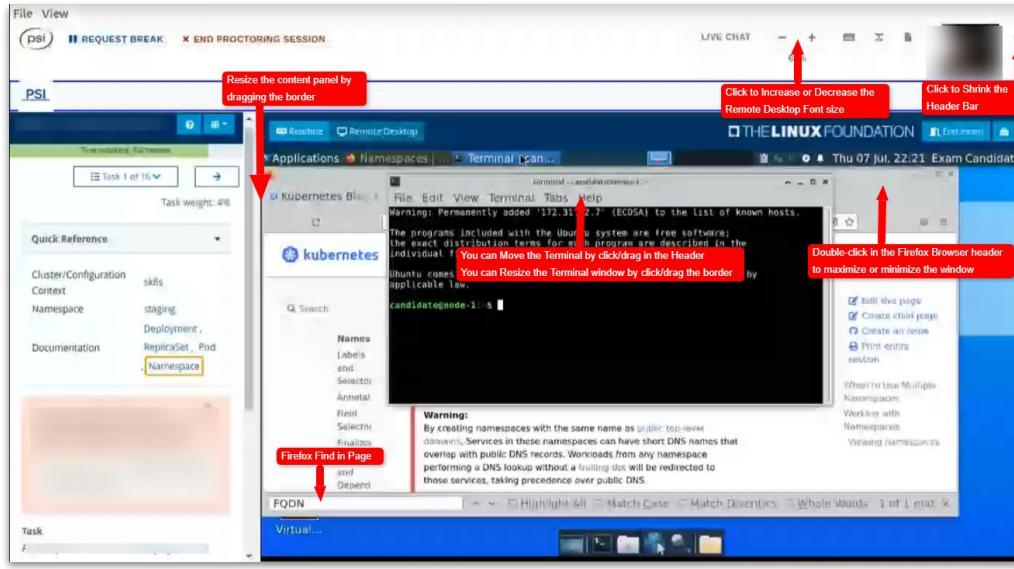


*The trinity of tooling you need to be familiar with*



# New PSI Exam Environment

*Single monitor, no more bookmarks (announcement)*



# Using Kubernetes Documentation

*Kubernetes docs and subdomains (see [FAQ](#))*

- Docs: <https://kubernetes.io/docs>
- GitHub: <https://github.com/kubernetes>
- Blog: <https://kubernetes.io/blog>



# Getting Help on a Command

*Render subcommands and options with --help*

```
$ kubectl create --help
```

Create a resource from a file or from stdin.

JSON and YAML formats are accepted.

...

Available Commands:

...

configmap	Create a configmap from a local file, directory or literal value
-----------	--

deployment	Create a deployment with the specified name.
------------	--

...

Options:

...



# Zeroing in on Command Details

*Drill into object details with the `explain` command*

```
$ kubectl explain pods.spec
KIND:     Pod
VERSION:  v1
```

```
RESOURCE: spec <Object>
```

```
DESCRIPTION:
```

```
...
```

```
FIELDS:
```

```
...
```

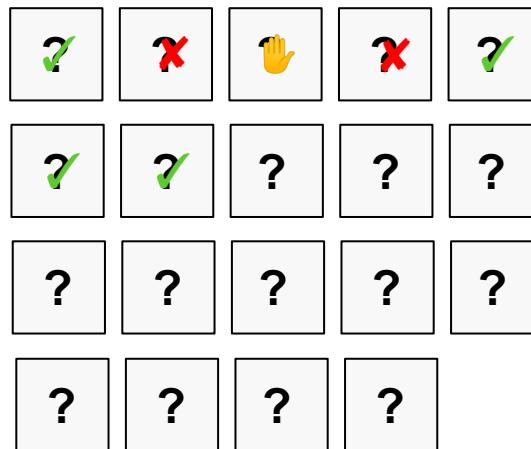
Most relevant information



---

# Time Management

*# of problems in 2 hours, use your time wisely!*



# Configuring Auto-Completion

*Allowed during exam, configurable on-demand*

```
$ kubectl cre<tab>
```

```
$ kubectl create
```



<https://kubernetes.io/docs/reference/kubectl/cheatsheet/#kubectl-autocomplete>



---

# Using an Alias for kubectl

*Preconfigured in the exam*

```
$ alias k=kubectl  
  
$ k version  
...
```



# Setting Namespace for a Context

*Questions will ask you to run a command on a specific cluster - Make sure to execute it!*

```
$ kubectl config set-context <context-of-question>--  
  --namespace=<namespace-of-question>  
$ kubectl config use-context <context-of-question>
```



# Internalize Resource Short Names

*Some API resources provide a shortcut*

```
$ kubectl get ns
```

Usage of `ns` instead  
of namespaces

```
$ kubectl describe pvc claim
```

Usage of `pvc` instead of  
`persistentvolumeclaim`

```
$ kubectl api-resources
```

Lists all API resources including  
their short names



---

# Deleting Kubernetes Objects

*Don't wait for a graceful deletion of objects...*

```
$ kubectl delete pod nginx --force
```



# Understand and Practice bash

*Practice relevant syntax and language constructs*

```
$ if [ ! -d ~/tmp ]; then mkdir -p ~/tmp; fi; while true; do echo $(date) >> ~/tmp/date.txt; sleep 5; done;
```



# Finding Object Information

*Filter configuration with context from a set of objects*

```
$ kubectl describe pods | grep -C 10 "author=John Doe"  
$ kubectl get pods -o yaml | grep -C 5 labels:
```

grep is your friend!



---

# How to Prepare

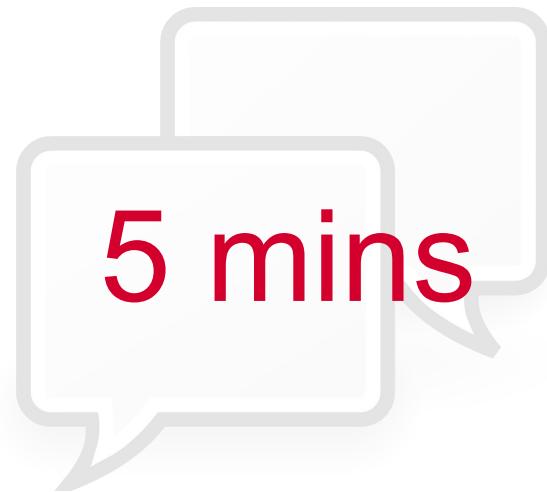
*Practice, practice, practice!*

**The key to cracking the exam**



---

# Q & A



# Image and Container Management

Defining, Building, Running, and Distributing  
Container Images

# Container Terminology

*Container image creation and distribution*

- *Blueprint of instructions*: Spells out what needs to happen when the software is built, e.g. a Dockerfile.
- *Container Image*: Packages an application into a single unit of software including its runtime environment and configuration.
- *Container Registry*: Publishes container images and makes them available for consumption, e.g. on [Docker Hub](#) or [GCR](#).



---

# Container Terminology

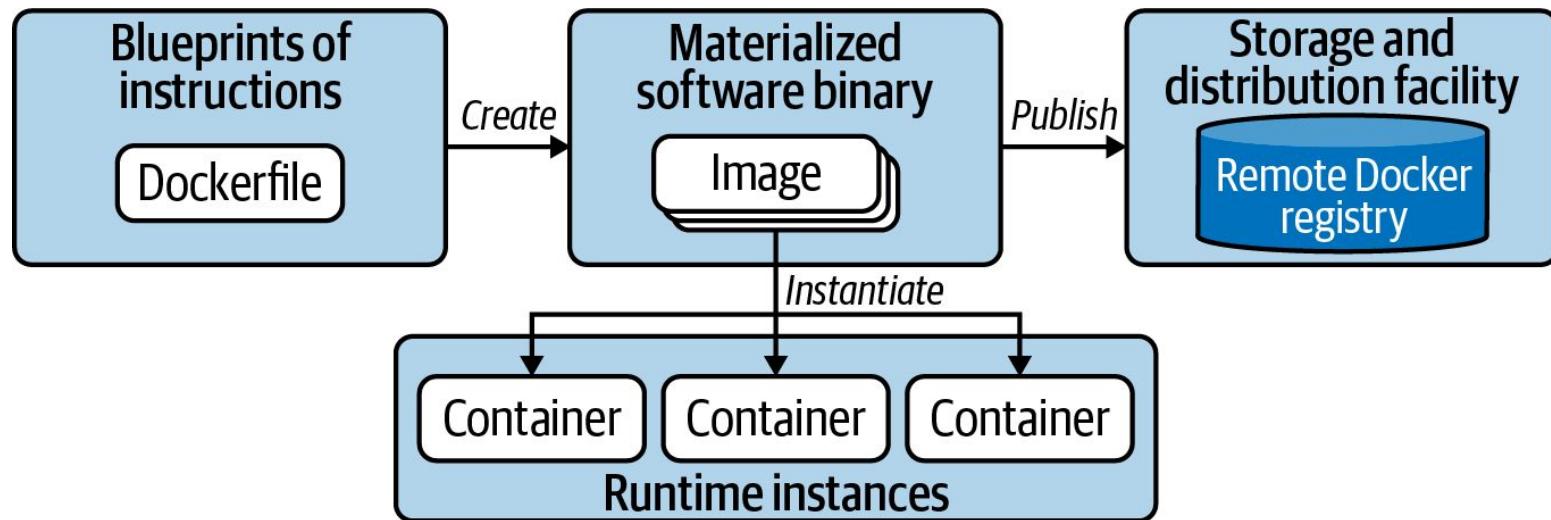
*Container runtime environment and instantiation*

- *Container Runtime Engine*: A software component that can run containers on a host operating system, e.g. Docker Engine or containerd.
- *Container*: Instantiates a container image and runs it in an isolated runtime environment. Think of it as a special type of virtual machine.
- *Container orchestrator*: Uses container runtime inside of a Pod to instantiate a container, e.g. Kubernetes.



# Containerization Process

*Building and publishing an image, running image in container*



# Image Building Instructions

A *Dockerfile* is a plain-text file with instructions

```
FROM openjdk:11-jre-slim
WORKDIR /app
COPY target/java-hello-world-0.0.1.jar java-hello-world.jar
ENTRYPOINT ["java", "-jar", "/app/java-hello-world.jar"]
EXPOSE 8080
```



# Building the Container Image

*Provide name and tag of image + context directory*

```
$ docker build -t java-hello-world:1.0.0 .  
Sending build context to Docker daemon 8.32MB  
Step 1/5 : FROM openjdk:11-jre-slim  
--> 973c18dbf567  
Step 2/5 : WORKDIR /app  
--> Using cache  
--> 31f9c5f2a019  
...  
Successfully built 3e9c22451a17  
Successfully tagged java-hello-world:1.0.0
```

Context directory

Container Image ID

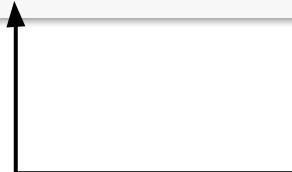


# Listing Container Images

*Render all local images including their name + hash*

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
java-hello-world	1.0.0	3e9c22451a17	About a minute ago	213MB
openjdk	11-jre-slim	973c18dbf567	20 hours ago	204MB



Base image is  
downloaded automatically



# Operating the Container

*You can interact with container once its running*

```
$ docker run -d -p 8080:8080 java-hello-world:1.0.0  
b0ee04accf078ea7c...
```

```
$ docker logs b0ee04accf078ea7c...  
Hello World!
```

```
$ docker exec -it b0ee04accf078ea7c... bash  
root@b0ee04accf078ea7c:/# pwd  
/  
root@b0ee04accf078ea7c:/# exit
```



# Tagging the Container Image

*Name + tag need to conform to registry conventions*

```
$ docker tag java-hello-world:1.0.0 bmuschko/java-hello-world:1.0.0
```

```
$ docker images
```

REPOSITORY	TAG	IMAGE ID	...
java-hello-world	1.0.0	3e9c22451a17	...
openjdk	11-jre-slim	973c18dbf567	...
bmuschko/java-hello-world	1.0.0	3e9c22451a17	...



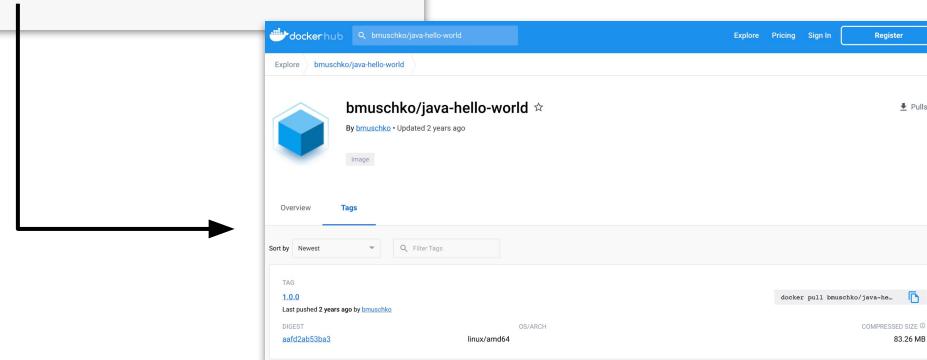
Tagged image



# Publishing the Container Image

*Authenticate with registry and push a tag*

```
$ docker login --username=bmuschko  
$ docker push bmuschko/java-hello-world:1.0.0
```



---

# Q & A

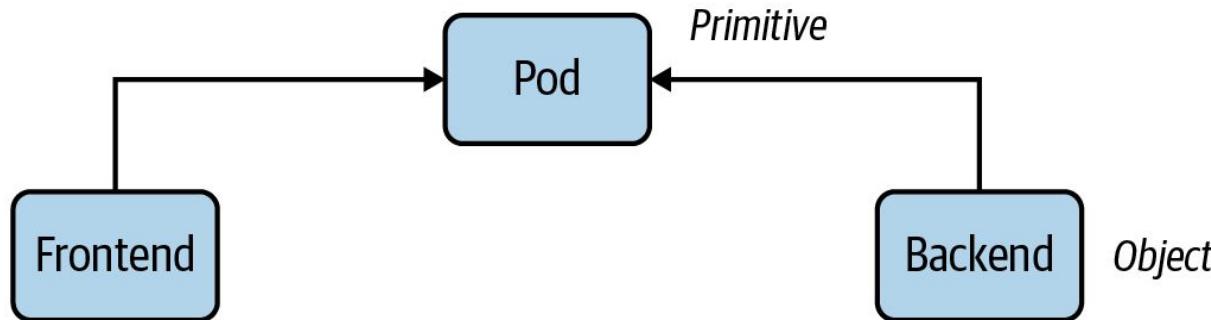


# Pod and Namespace Management

Kubernetes Primitives, Typical Operations and Configuration

# Kubernetes Primitives

*Basic building blocks for creating and operating an application*



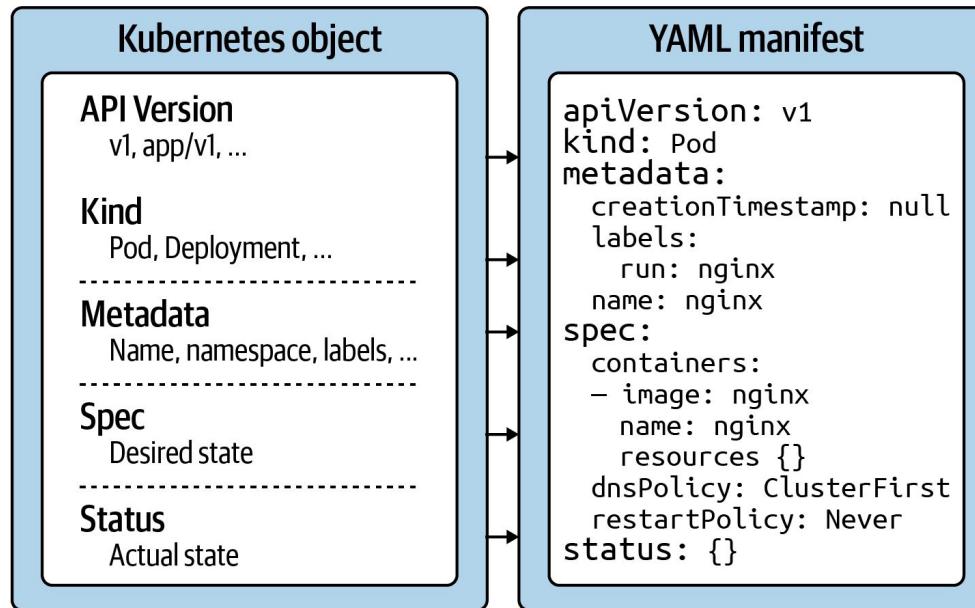
UID: 4ef7b090-37ed-4b33-8fb7-c5693a48eef5

UID: b674f2e0-f0bc-40be-af8e-7985442c21a2



# Kubernetes Object Structure

*General structure followed by most primitives*



---

# General Primitive Sections

*Each section has a specific function*

- *API version*: Defines the structure of a primitive and uses it to validate the correctness of the data.
- *Kind*: The kind defines the type of primitive, e.g. a Pod or a Service.
- *Metadata*: Describes higher-level information about the object.
- *Spec/Status*: Desired and actual state of object.



# Using kubectl

*Primary tool to interact with the Kubernetes clusters from the CLI*

```
kubectl [command] [TYPE] [NAME] [flags]  
        get      pod     app   -o yaml
```



# Imperative Object Management

*Fast but requires detailed knowledge, no track record*

```
$ kubectl create namespace ckad  
namespace/ckad created
```

Creates an object  
e.g. a namespace

```
$ kubectl run nginx --image=nginx -n ckad  
pod/nginx created
```

Creates a Pod object  
in a namespace

```
$ kubectl edit pod nginx -n ckad  
pod/nginx edited
```

Modifies the live  
object e.g. a Pod



# Declarative Object Management

*Suitable for more elaborate changes, tracks changes*

```
$ kubectl create -f nginx-deployment.yaml  
deployment.apps/nginx-deployment created
```

Creates object if it  
doesn't exist yet

```
$ kubectl apply -f nginx-deployment.yaml  
deployment.apps/nginx-deployment configured
```

Creates object or  
update if it already  
exists

```
$ kubectl delete -f nginx-deployment.yaml  
deployment.apps "nginx-deployment" deleted
```

Deletes object



# Hybrid Approach

*Generate YAML file with kubectl but make further edits*

```
$ kubectl run nginx --image=nginx  
  --dry-run=client -o yaml > nginx-pod.yaml  
  
$ vim nginx-pod.yaml  
  
$ kubectl create -f nginx-pod.yaml  
pod/nginx created
```



Capture YAML for Pod object in file



Create the object from YAML file



---

# What is a Pod?

*Runs a process or application in container(s)*

- The smallest, most basic deployable objects in Kubernetes.
- It contains one or more containers, such as Docker containers.
- Kubernetes enhances the basic functionality of a container with features such as security, storage, and more.



# Pod YAML Manifest

*Defines containers and their images running in them*

```
apiVersion: v1
kind: Pod
metadata:
  name: hazelcast
spec:
  containers:
    image: hazelcast/hazelcast
    name: hazelcast
    ports:
      - containerPort: 5701
  restartPolicy: Never
```

← Container image

← Exposed container port(s)



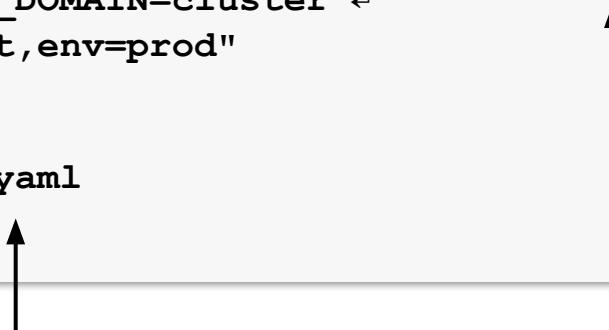
# Creating a Pod

*Imperative vs. declarative approach*

```
$ kubectl run hazelcast --image=hazelcast/hazelcast --restart=Never  
  --port=5701 --env="DNS_DOMAIN=cluster"  
  --labels="app=hazelcast,env=prod"  
pod/hazelcast created  
  
$ kubectl create -f pod.yaml  
pod/hazelcast created
```

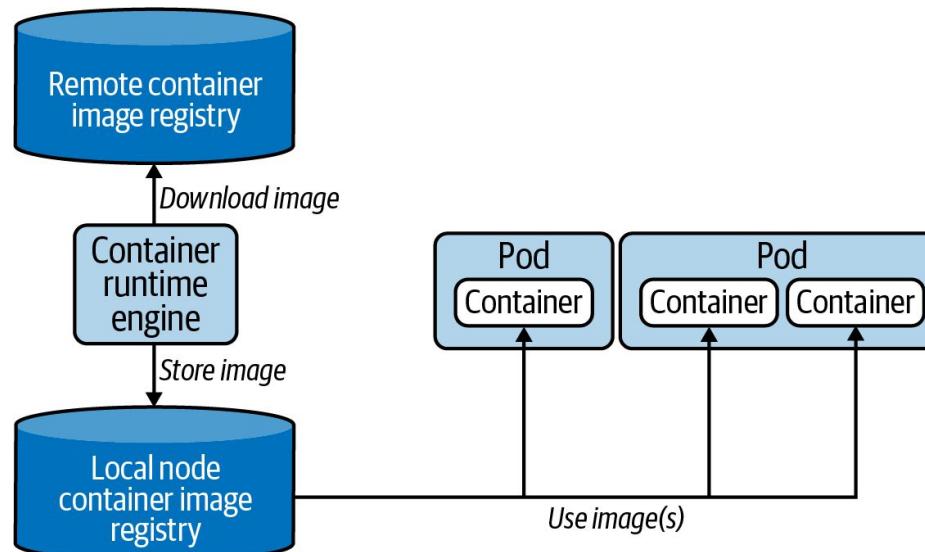
*Declarative:* Point to  
YAML manifest

*Imperative:* Spell out  
configuration with parameters



# Container Image Retrieval

*Same process for imperative and declarative Pod creation*



# Listing Pods

*Get high-level information on all Pods or a specific one*

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
hazelcast	1/1	Running	0	17s

```
$ kubectl get pods hazelcast
```

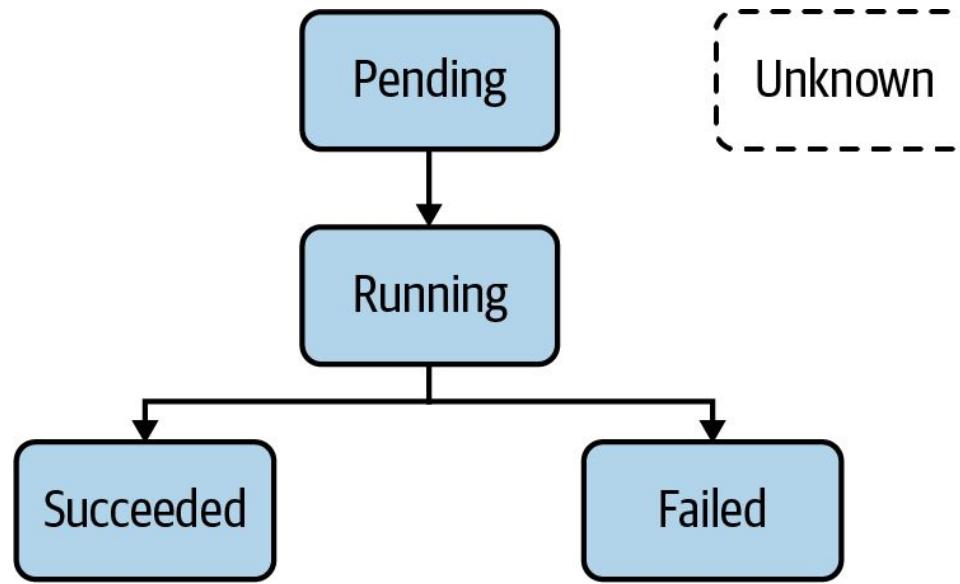
NAME	READY	STATUS	RESTARTS	AGE
hazelcast	1/1	Running	0	17s

Specific Pod name



# Pod Life Cycle Phases

*Indicates operational status of Pod*



# Rendering Pod Details

*Shows Pod configuration, events, and status*

```
$ kubectl describe pods hazelcast
Name:           hazelcast
Namespace:      default
...
Status:         Running
IP:             10.1.0.41
Containers:
...
Events:
```

Virtual IP address assigned to  
Pod to communicate with it



# Interacting with Container

*Rendering logs and shelling into container*

```
$ kubectl logs hazelcast
...
May 25, 2020 3:36:26 PM com.hazelcast.core.LifecycleService
INFO: [10.1.0.46]:5701 [dev] [4.0.1] [10.1.0.46]:5701 is STARTED

$ kubectl exec -it hazelcast -- /bin/sh
# ...
```



# Declaring Environment Variables

*Define zero to many key-value pairs*

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-boot-app
spec:
  containers:
    - image: bmuschko/spring-boot-app:1.5.3
      name: spring-boot-app
      env:
        - name: SPRING_PROFILES_ACTIVE
          value: prod
        - name: VERSION
          value: '1.5.3'
```

Typical naming conventions  
for keys are not enforced



# Declaring Command and Args

*Assign or overwrite container entrypoint and arguments*

```
apiVersion: v1
kind: Pod
metadata:
  name: spring-boot-app
spec:
  containers:
    - image: bmuschko/spring-boot-app:1.5.3
      name: spring-boot-app
      command: ["/bin/sh"]
      args: ["-c", "while true; do date; sleep 10; done"]
```



# Deleting a Pod

*Graceful deletion can take up to 30 seconds*

```
$ kubectl delete pod hazelcast  
pod "hazelcast" deleted
```

```
$ kubectl delete -f pod.yaml  
pod "hazelcast" deleted
```

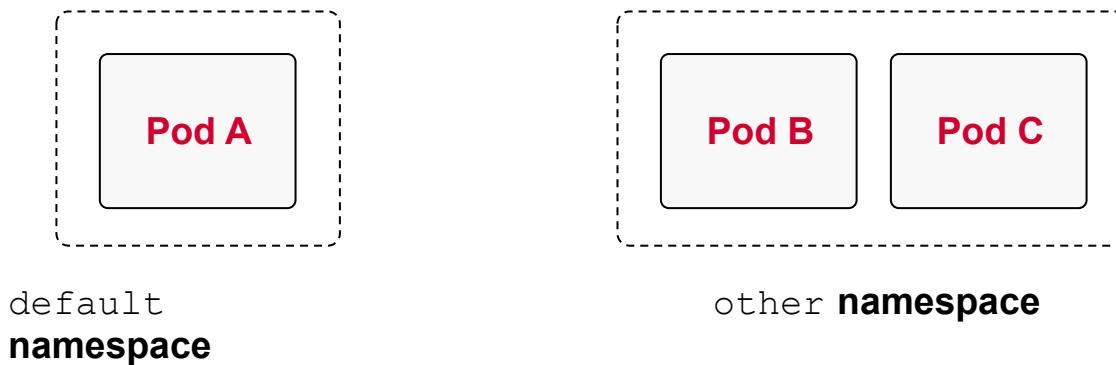
← Imperative approach

← Declarative approach



# Understanding Namespaces

*Prevent naming collisions and a scope for object names*



# Creating and Using a Namespace

*Reference namespace with --namespace or -n*

```
$ kubectl create namespace code-red
namespace/code-red created

$ kubectl run nginx --image=nginx --restart=Never -n code-red
pod/nginx created

$ kubectl get pods -n code-red
NAME      READY     STATUS    RESTARTS   AGE
nginx    1/1      Running   0          13s
```



---

# Namespace YAML Manifest

*Only defines the name, no additional details*

```
apiVersion: v1
kind: Namespace
metadata:
  name: code-red
```



# Deleting a Namespace

*Cascade-deletes containing objects*

```
$ kubectl delete namespace code-red
namespace "code-red" deleted

$ kubectl get pods -n code-red
No resources found in code-red namespace.
```



---

# EXERCISE

Creating a Pod  
and Inspecting it



---

# Q & A



# Jobs and CronJobs

Differences to Pods, One-Time Operations,  
Scheduled Operations

# Pods vs. Jobs vs. CronJobs

*Kubernetes primitives with different use cases*

- *Pod*: Continuous operation of an application kept running without interrupts if possible.
- *Job*: Runs functionality until a specified number of completions has been reached, making it a good fit for one-time operations like import/export data processes or I/O-intensive processes with a finite end.
- *CronJob*: Essentially a Job, but it's run periodically based a schedule.



# Creating a Job

*"Increment a counter and render its value on the terminal"*

```
$ kubectl create job counter --image=nginx -- /bin/sh -c
'counter=0; while [ $counter -lt 3 ]; do
  counter=$((counter+1)); echo "$counter"; sleep 3; done;'
job.batch/counter created
```



# Job YAML Manifest

```
apiVersion: batch/v1
kind: Job
metadata:
  name: counter
spec:
  completions: 1
  parallelism: 1
  backoffLimit: 6
  template:
    spec:
      restartPolicy: OnFailure
      containers:
        - args:
            - /bin/sh
            - -c
            - ...
          image: nginx
          name: counter
```

Define # of successful completions and whether task should be run in parallel

How many times do we try before Job is marked failed?

Restart Pod upon failure or start a new Pod



# Inspecting a Job

*Job is completed when configured # of execution has been reached*

```
$ kubectl get jobs
```

NAME	COMPLETIONS	DURATION	AGE
counter	0/1	13s	13s

```
$ kubectl get jobs
```

NAME	COMPLETIONS	DURATION	AGE
counter	1/1	15s	19s

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
counter-z6kdj	0/1	Completed	0	51s



# Different Types of Jobs

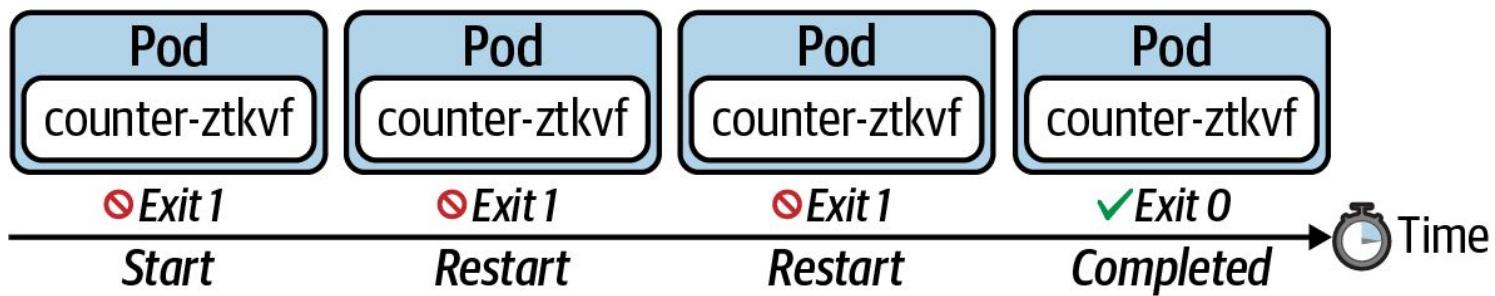
```
spec.completions: x  
spec.parallelism: y
```

Type	Completion criteria
Non-parallel	Complete as soon as its Pod terminates successfully
Parallel with fixed completion count	Complete when specified number of tasks finish successfully
Parallel with a work queue	Once at least one Pod has terminated with success and all Pods are terminated



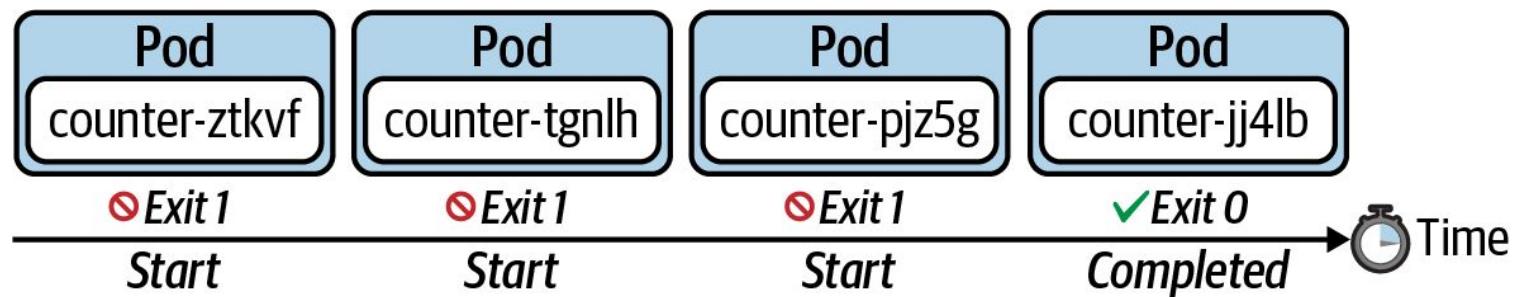
# Restarting a Container on Failure

`spec.template.spec.restartPolicy: OnFailure`



# New Pod on Failure

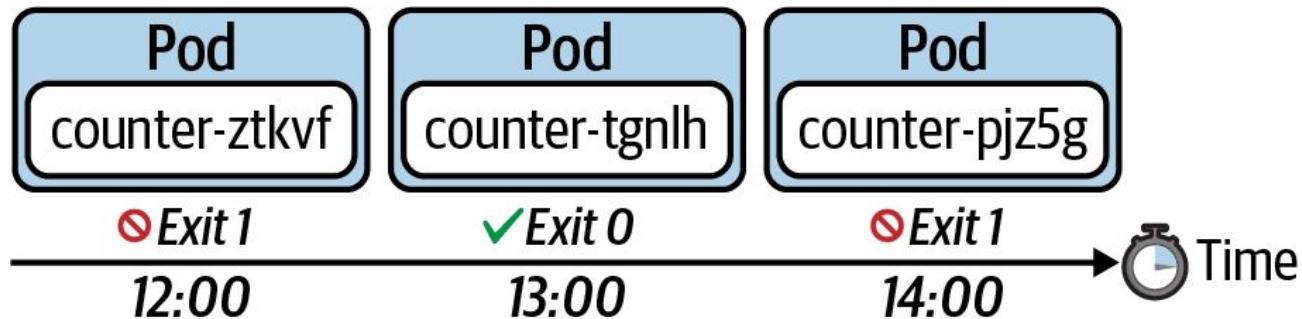
```
spec.template.spec.restartPolicy: Never
```



# Understanding CronJobs

*Similar to Job but task is run on a predefined schedule*

```
spec.schedule: "0 * * * *"
```



# Creating a CronJob

*"Render the current date on standard output every hour"*

```
$ kubectl create cronjob current-date --schedule="* * * * *" --image=nginx -- /bin/sh -c 'echo "Current date: $(date)"'
```

cronjob.batch/current-date created



# CronJob YAML Manifest

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: counter
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: Never
          containers:
            - args:
                - /bin/sh
                - -c
                - ...
              image: nginx
              name: counter
```

The crontab expression used to run CronJob periodically

Run in a new Pod



# Inspecting CronJobs

*Keeps a history of failed and successful Pods*

```
$ kubectl get cronjobs
```

NAME	SCHEDULE	SUSPEND	ACTIVE	LAST SCHEDULE	AGE
counter	*/1 * * * *	False	0	26s	1h

```
$ kubectl get jobs --watch
```

NAME	COMPLETIONS	DURATION	AGE
counter-1557334380	1/1	3s	2m24s
counter-1557334440	1/1	3s	84s
counter-1557334500	1/1	3s	24s



---

# EXERCISE

Creating a  
Scheduled  
Container Operation



---

# Q & A



# Multi-Container Pods

Common Design Patterns

# Single vs. Multiple Containers

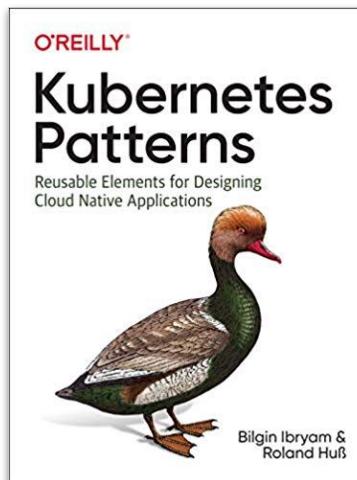
*Used for implementing specific use cases*

- The general rule of thumb is to deploy one microservice per Pod in a *single container* which helps with rolling out new versions of a microservice without necessarily interrupting other parts of the system.
- Use cases for *multiple containers* in a single Pod
  - Run preconfiguration procedure before the application container should start.
  - Helper functionality that runs alongside the application.



# Multi-Container Patterns

*Understand patterns on a high-level*

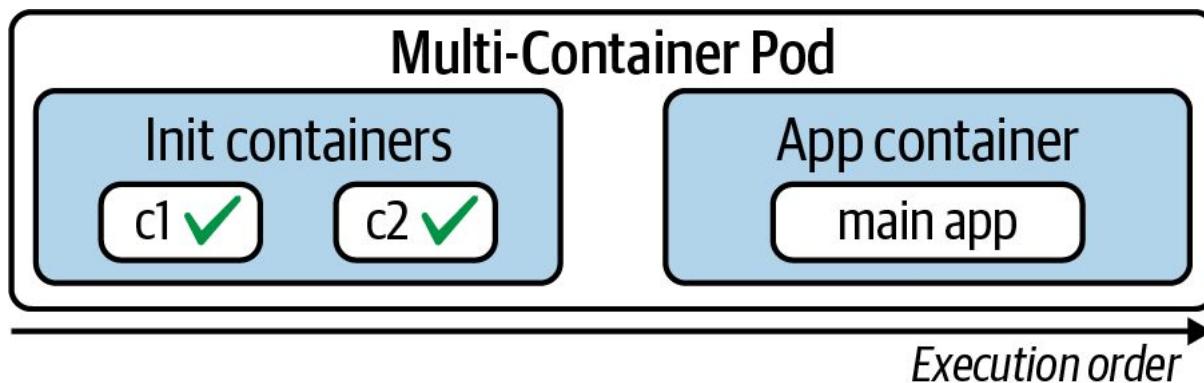


- Init container
- Sidecar
- Adapter
- Ambassador



# Why Use an Init Container?

*Initialization logic before main application containers*



# Defining an Init Container

*Declared adjacent to main application containers*

```
apiVersion: v1
kind: Pod
metadata:
  name: multi-container
spec:
  initContainers:
    - image: init:3.2.1
      name: app-initializer
  containers:
    - image: nginx
      name: web-server
```



Uses the same attributes as  
main application containers



# Creating a Pod with Init Container

*Init container execution renders in status column*

```
$ kubectl create -f init.yaml  
pod/business-app created
```

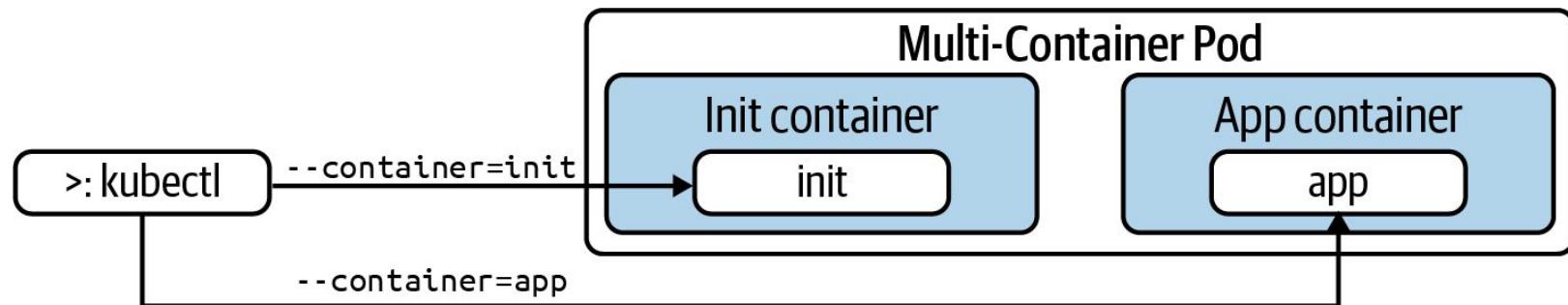
```
$ kubectl get pod business-app  
NAME      READY  STATUS          RESTARTS   AGE  
business-app  0/1    PodInitializing  0          2s
```

```
$ kubectl get pod business-app  
NAME      READY  STATUS          RESTARTS   AGE  
business-app  1/1    Running        0          8s
```



# Interacting with an Init Container

*Run typical kubectl with --container or -c CLI option*



---

# EXERCISE

Creating an Init  
Container



# Defining Multiple Containers

*Shared container lifecycle and resources*

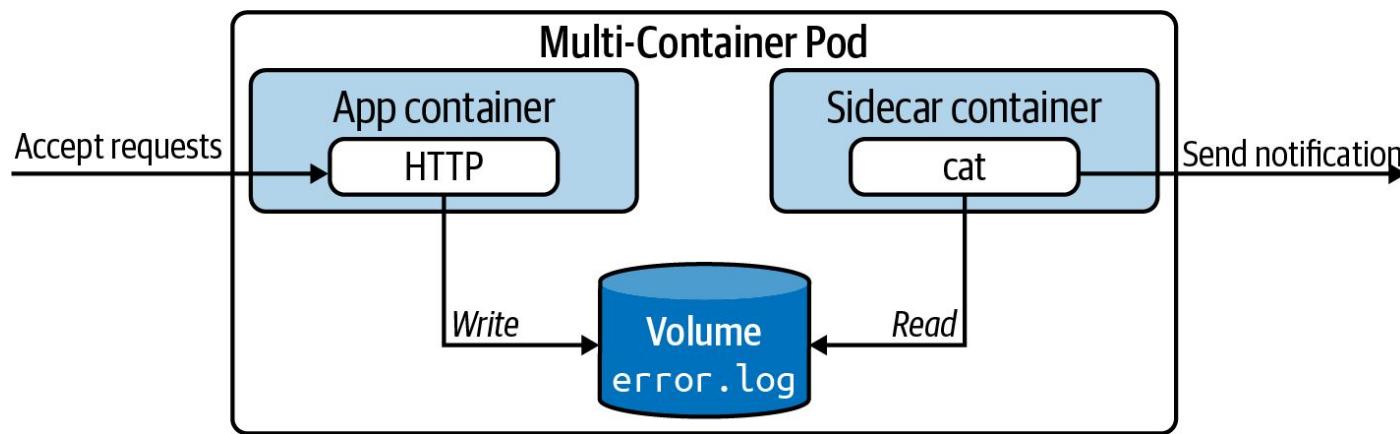
```
apiVersion: v1
kind: Pod
metadata:
  name: multi-container
spec:
  containers:
    - image: nginx:1.13.0
      name: web-server
    - image: transformer:1.0.0
      name: helper
```

Defines a second container  
running alongside the  
application container



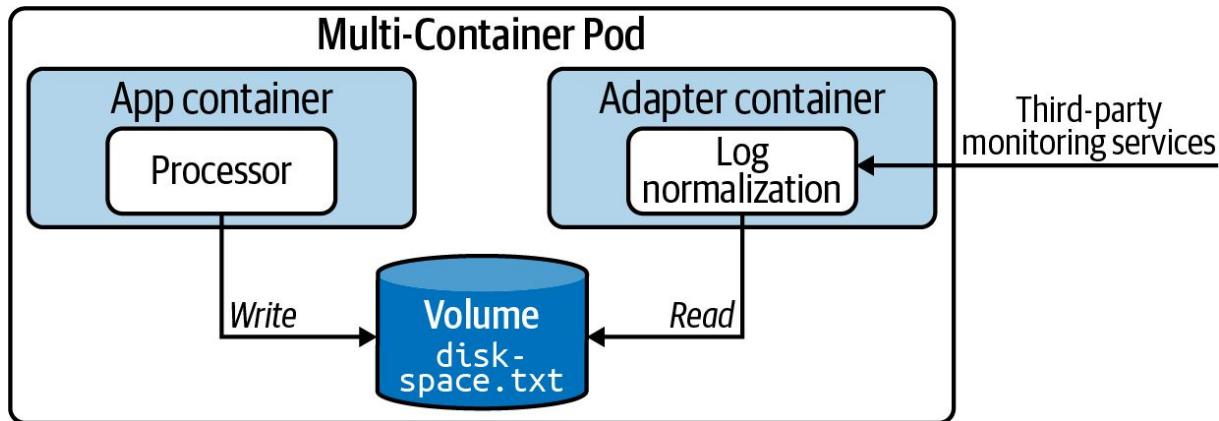
# Sidecar Pattern

*Not part of the main traffic or API of the primary application*



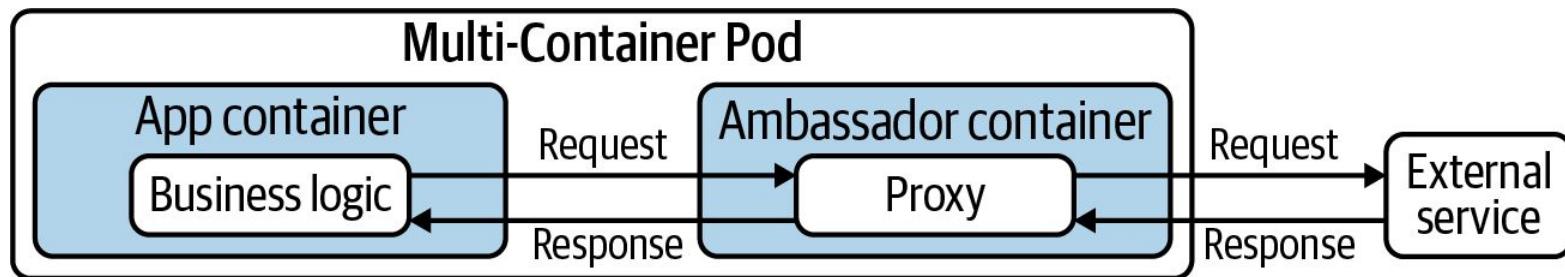
# Adapter Pattern

*Standardizes and normalizes application output read by external monitoring service*



# Ambassador Pattern

*Hide and/or abstract the complexity of interacting with other parts of the system*



---

# EXERCISE

Implementing the  
Adapter Pattern



---

# Q & A

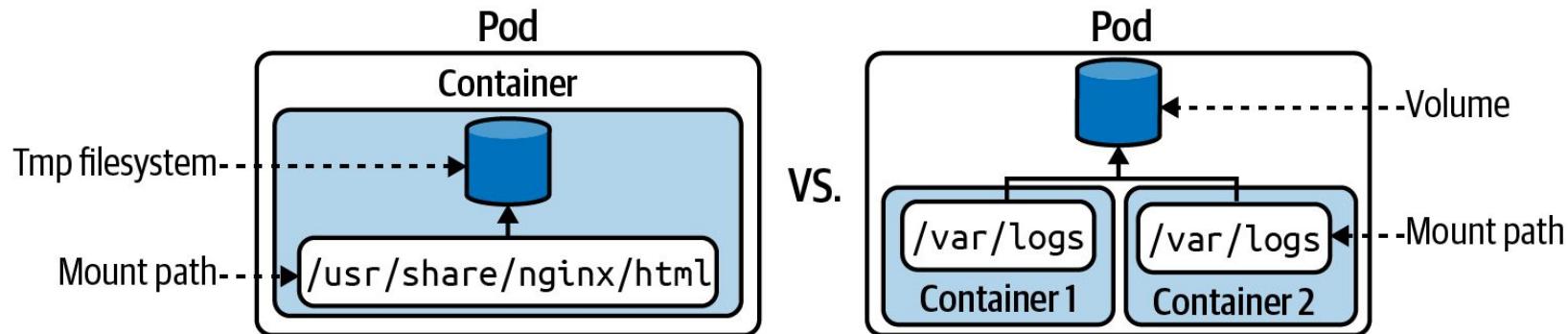


# **State Persistence**

Persistent and Ephemeral Volumes

# Understanding Volumes

*By default, a container manages files in a temporary file system*



# Types of Volumes

Type	Description
emptyDir	Empty directory in Pod. Only persisted for the lifespan of a Pod.
hostPath	File or directory from the host node's filesystem into your Pod.
configMap, secret	Provides a way to inject configuration data and secrets into Pods.
nfs	An existing NFS (Network File System) share to be mounted into your Pod. Preserves data after Pod restart.
Cloud provider solutions	Provider-specific implementation for AWS, GCE or Azure.



# Defining and Mounting a Volume

*Provide type and assign mount path per container*

```
apiVersion: v1
kind: Pod
metadata:
  name: my-container
spec:
  volumes:
    - name: logs-volume
      emptyDir: {}
  containers:
    - image: nginx
      name: my-container
      volumeMounts:
        - mountPath: /var/logs
          name: logs-volume
```

← Define Volume name and type

← Specify mount path in container



# Using a Volume

*New files can be created in mounted path of Volume*

```
$ kubectl create -f pod-with-vol.yaml
pod/my-container created

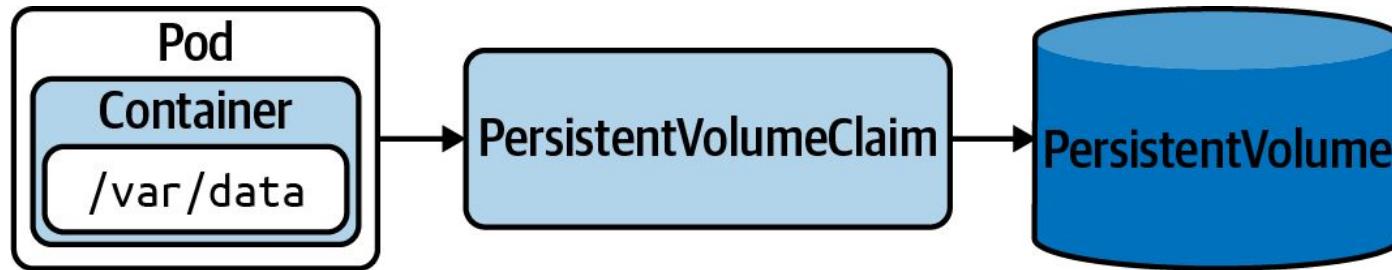
$ kubectl exec -it my-container -- /bin/sh
# cd /var/logs
# pwd
/var/logs
# touch app-logs.txt
# ls
app-logs.txt
```

Mount path became  
accessible in container



# Understanding PersistentVolumes

*Persist data that outlives a Pod, node, or cluster restart*



# Static vs. Dynamic Provisioning

*PersistentVolume object is created manually or automatically*

- *Static:* Storage device needs to be created first. The PersistentVolume object references the storage device and needs to be created manually.
- *Dynamic:* The PersistentVolumeClaim object references a storage class. The PersistentVolume object is created automatically.
- *Storage Class:* Object that knows how to provision a storage device with specific performance requirements.



# Defining a PersistentVolume

*Define storage capacity, access mode, and host path*

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: db-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /data/db
```



# Access Mode & Reclaim Policy

*Configuration options for PersistentVolume*

## Access Mode

Type	Description
ReadWriteOnce	Read-write access by a single node.
ReadOnlyMany	Read-only access by many nodes.
ReadWriteMany	Read-write access by many nodes.

## Reclaim Policy

Type	Description
Retain	Default. When PVC is deleted, PV is “released” and can be reclaimed.
Delete	Deletion removes PV and associated storage.
Recycle	Deprecated. Use dynamic binding instead.



# Creating the PersistentVolume

*Listing the object show many of its configuration options*

```
$ kubectl create -f db-pv.yaml
persistentvolume/db-pv created

$ kubectl get pv db-pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      ...
db-pv    1Gi          RWO           Retain        Available   ...
```



The object hasn't been  
consumed by a claim yet



# Defining a Claim

*Will use a PersistentVolume based on resource request*

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: db-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 256Mi
storageClassName: ""
```

Assign an empty storage class name to  
use a statically-created PersistentVolume



# Creating the PersistentVolumeClaim

*Listing the object show many of its configuration options*

```
$ kubectl create -f db-pvc.yaml
persistentvolumeclaim/db-pvc created
```

```
$ kubectl get pvc db-pvc
NAME      STATUS    VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
db-pvc   Bound     pvc       512m       RWO          standard     7s
```



Binding to the PersistentVolume  
object was successful



# Mounting a Claim in a Pod

*Use Volume type persistentVolumeClaim*

```
apiVersion: v1
kind: Pod
metadata:
  name: app-consuming-pvc
spec:
  volumes:
    - name: app-storage
      persistentVolumeClaim:
        claimName: db-pvc
  containers:
    - image: alpine
      ...
      volumeMounts:
        - mountPath: "/mnt/data"
          name: app-storage
```

Reference the Volume by claim name



# Creating the Pod

*The Volume is now listed in Pod details*

```
$ kubectl create -f app-consuming-pvc.yaml
pod/app-consuming-pvc created

$ kubectl describe pod app-consuming-pvc
...
Volumes:
  app-storage:
    Type:      PersistentVolumeClaim (a reference to a
               PersistentVolumeClaim in the same namespace)
    ClaimName: db-pvc
    ReadOnly:   false
...
```



# Dynamic Provisioning

*Creates PersistentVolume object automatically via storage class*

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 256Mi
  storageClassName: standard
```



---

# EXERCISE

Creating a  
Persistent Volume  
with Static Binding



---

# Q & A



# Labels and Annotations

Commonalities and Differences, Use Cases, Usage

---

# What is a Label?

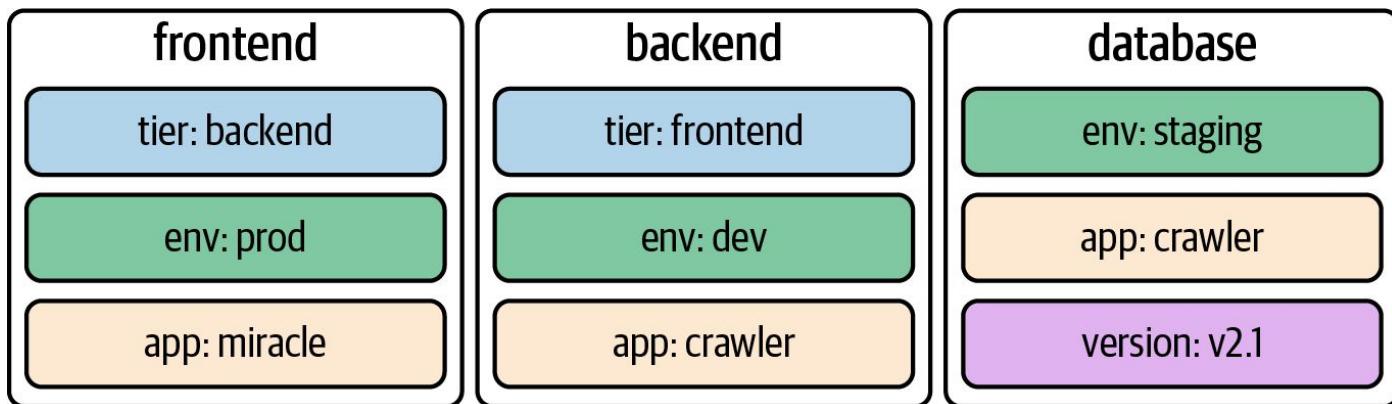
*Essential to querying, filtering and sorting Kubernetes objects*

- Key-value pairs assigned to Kubernetes objects.
- Not meant for elaborate, multi-word descriptions of its functionality.
- Kubernetes limits the length of a label to a maximum of 63 characters and a range of allowed alphanumeric and separator characters.



# Example Labels

*Three different Pods with each three key-value pairs*



# Assigning Labels in YAML

*Use the `metadata.labels` attribute*

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  labels:
    tier: backend
    env: prod
    app: miracle
spec:
  ...
```

← Declares three label  
key-value pairs



# Assigning Labels on CLI

*Use the `kubectl label` command*

```
$ kubectl label pod nginx tier=backend env=prod app=miracle  
pod/nginx labeled
```

← Declares three label key-value pairs

```
$ kubectl get pods --show-labels  
NAME    ... LABELS  
pod1   ... tier=backend,env=prod,app=miracle
```

```
$ kubectl label pod nginx tier-  
pod/nginx labeled
```

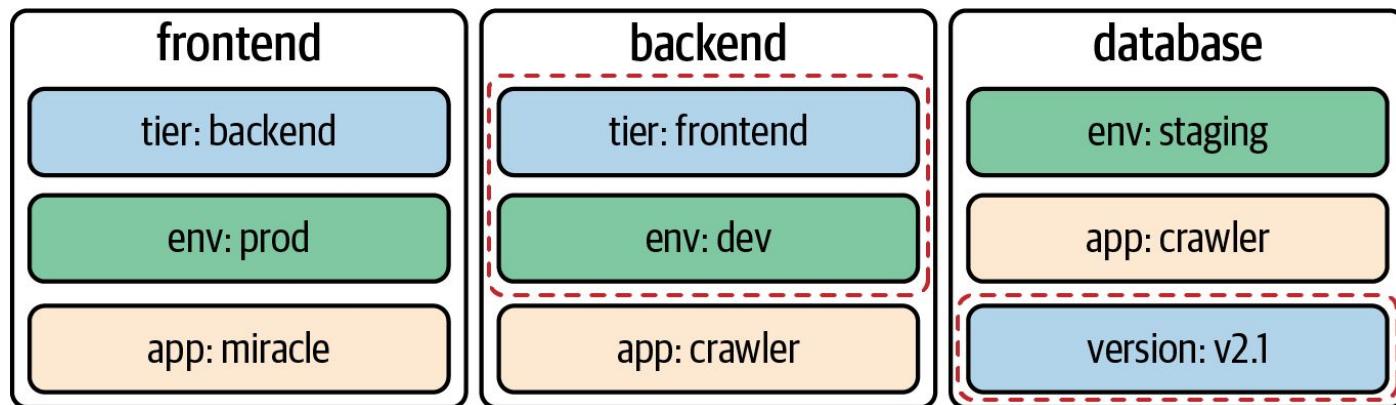
← Removes the label with key tier

```
$ kubectl get pods --show-labels  
NAME    ... LABELS  
pod1   ... env=prod,app=miracle
```



# Selecting Labels

*Label selector uses a set of criteria to query for objects*



# Label Selection from the CLI

*You can specify equality-based and set-based requirements*

```
$ kubectl get pods -l tier=frontend,env=dev --show-labels
```

```
NAME ... LABELS  
pod2 ... app=crawler, env=dev, tier=frontend
```

Boolean AND expression

```
$ kubectl get pods -l version --show-labels
```

```
NAME ... LABELS  
pod3 ... app=crawler, env=staging, version=v2.1
```

Key-only expression

```
$ kubectl get pods -l 'tier in (frontend,backend),env=dev' --show-labels
```

```
NAME ... LABELS  
pod2 ... app=crawler, env=dev, tier=frontend
```

Set-based expression follows Boolean OR



# Label Selection in YAML

*Certain objects provide label selection via their API*

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: frontend-network-policy
spec:
  podSelector:
    matchLabels:
      tier: frontend
...
...
```

Select Pods by labels that this network policy should apply to



---

# What is an Annotation?

*Descriptive metadata without the ability to be queryable*

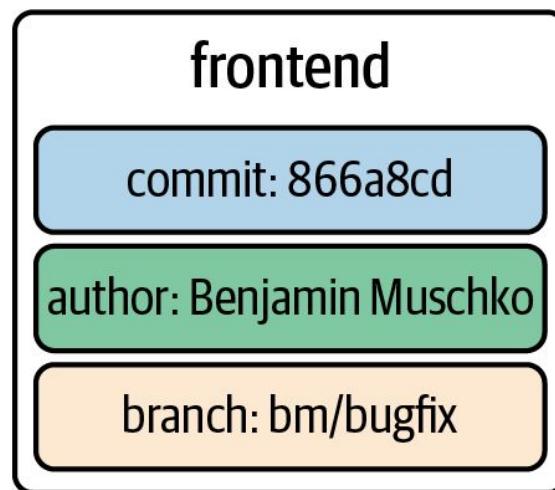
- Key-value pairs for providing descriptive, human-readable metadata.
- Cannot be used for querying or selecting objects.
- Make sure to put the value of an annotation into single- or double-quotes if it contains special characters or spaces.



---

# Example Annotations

*A Pod with three key-value pairs*



# Assigning Annotations in YAML

*Use the `metadata.annotations` attribute*

```
apiVersion: v1
kind: Pod
metadata:
  name: pod1
  annotations:
    commit: 866a8dc
    author: 'Benjamin Muschko'
    branch: 'bm/bugfix'
spec:
  ...
```

← Declares three annotation  
key-value pairs



# Assigning Annotations on CLI

*Use the `kubectl annotations` command*

```
$ kubectl annotate pod nginx commit='866a8dc' branch='bm/bugfix'  
pod/nginx annotated
```

← Declares three annotation key-value pairs

```
$ kubectl describe pods my-pod  
Annotations: branch: bm/bugfix  
            commit: 866a8dc
```

← Removes the annotation with key commit

```
$ kubectl annotate pod nginx commit-  
pod/nginx annotated
```

```
$ kubectl describe pods my-pod  
Annotations: branch: bm/bugfix
```



---

# EXERCISE

Defining and  
Querying Labels  
and Annotations



---

# Q & A



# Application Deployment

Deployments, Manual and Automatic Deployment,  
Rollout Strategies, Using Helm to Deploy an  
Application

---

# What is a Deployment?

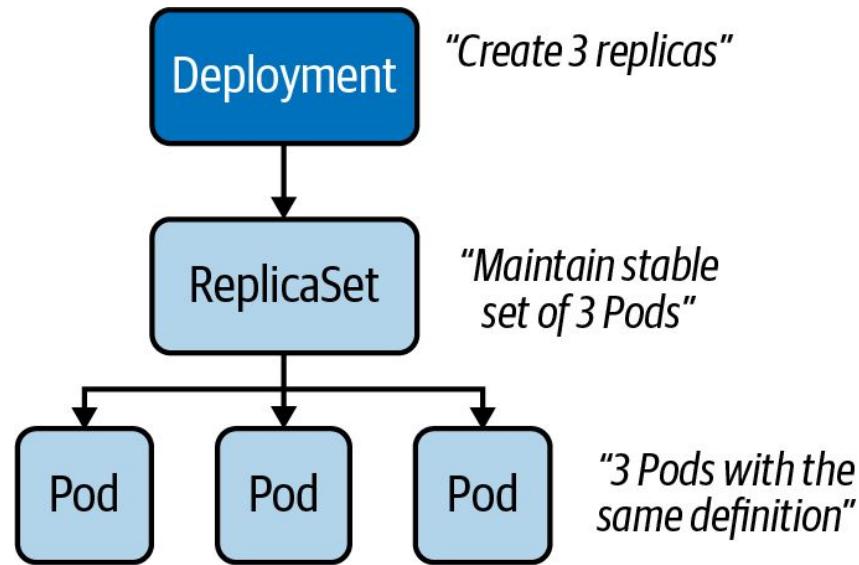
*Scaling and replication features for a set of Pods*

- Controls a predefined number of Pods with the same configuration, so-called *replicas*.
- The number of replicas can be scaled up or down to fulfill load requirements.
- Updates to the replica configuration can be updated easily and is rolled out automatically.



# Example Deployment

*A Deployment that controls three replicas*



# Creating a Deployment

*Creates objects for the Deployment, ReplicaSet and Pods*

```
$ kubectl create deployment my-deploy --image=nginx:1.14.2 --replicas=3  
deployment.apps/my-deploy created
```

The default number of replicas is 1 if the parameter wasn't provided



# Deployment YAML Manifest

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-deploy
    name: my-deploy
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: backend
  template:
    metadata:
      labels:
        tier: backend
    spec:
      containers:
        - image: nginx:1.14.2
          name: nginx
```

Label selector and  
template label assignment  
have to match



# Listings Deployments

*ReplicaSet and Pods can be identified by name prefix*

```
$ kubectl get deployments,pods,replicasets
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/my-deploy	1/1	1	1	7m56s

NAME	READY	STATUS	RESTARTS	AGE
pod/my-deploy-8448c488b5-mzx5g	1/1	Running	0	7m56s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/my-deploy-8448c488b5	1	1	1	7m56s



# Rendering Deployment Details

*Object details show relationship between parent and child*

```
$ kubectl describe deployment.apps/my-deploy
Replicas:           1 desired | 1 updated | 1 total | 1 available |
                  0 unavailable
NewReplicaSet:    my-deploy-8448c488b5 (1/1 replicas created)
```

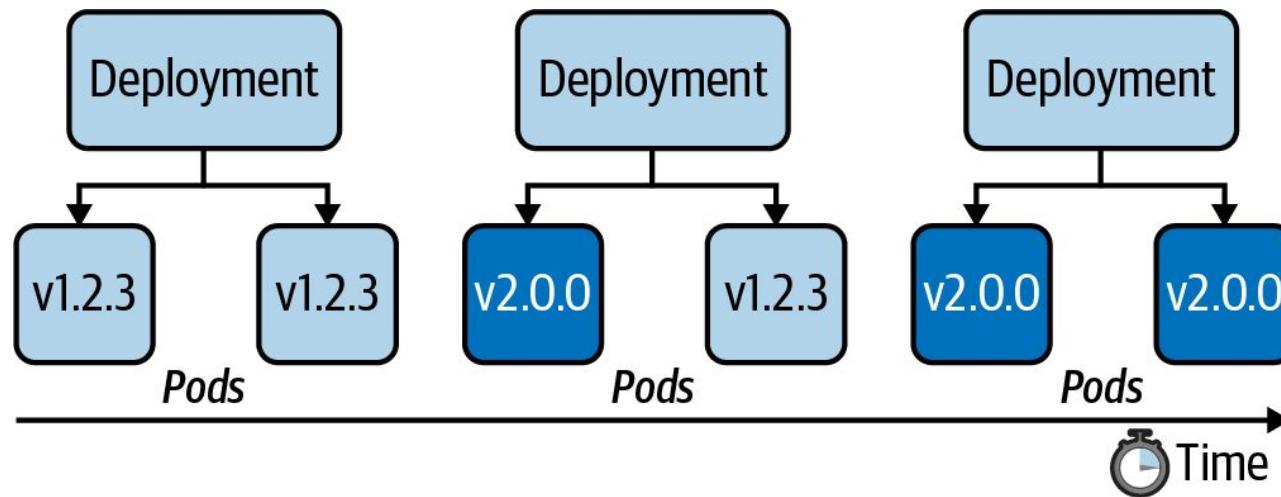
```
$ kubectl describe replicaset.apps/my-deploy-8448c488b5
Controlled By:  Deployment/my-deploy
```

```
$ kubectl describe pod/my-deploy-8448c488b5-mzx5g
Controlled By:  ReplicaSet/my-deploy-8448c488b5
```



# Roll Out and Roll Back Feature

*“Look ma, shiny new features. Let’s deploy them to production!”*



# Rolling Out a New Revision

*The rollout history keeps track of changes*

```
$ kubectl rollout history deployment my-deploy
deployment.apps/my-deploy
REVISION  CHANGE-CAUSE
1          <none>
```

```
$ kubectl set image deployment my-deploy nginx=nginx:1.19.2
deployment.apps/my-deploy image updated
```

```
$ kubectl rollout history deployment my-deploy
deployment.apps/my-deploy
REVISION  CHANGE-CAUSE
1          <none>
2          <none>
```

Changes the assigned image for Pod template



# Rolling Back to Previous Revision

*You can revert to any revision available in the history*

```
$ kubectl rollout undo deployment my-deploy --to-revision=1
deployment.apps/my-deploy rolled back
```

```
$ kubectl rollout history deployment my-deploy
deployment.apps/my-deploy
REVISION  CHANGE-CAUSE
2          <none>
3          <none>
```



Kubernetes deduplicates a revision  
if it points to the same changes



---

# Common Deployment Strategies

*Choosing the right deployment procedure depends on the needs*

- *Recreate*: Terminate the old version and release the new one.
- *Ramped*: Release a new version on a rolling update fashion, one after the other.
- *Blue/green*: Release a new version alongside the old version then switch traffic.
- *Canary*: Release a new version to a subset of users, then proceed to a full rollout.



# Recreate Deployment Strategy

*Terminates all the running instances then recreate them*

```
spec:  
  replicas: 3  
  strategy:  
    type: Recreate
```

← Set the strategy explicitly

- *Pros:* Good for development environments as everything is renewed at once
- *Cons:* Can cause downtime while applications are starting up



# Ramped Deployment Strategy

*Secondary ReplicaSet is created with new version of application*

```
spec:  
  replicas: 3  
  strategy:  
    type: RollingUpdate  
    rollingUpdate:  
      maxSurge: 2  
      maxUnavailable: 0
```



Determines how update  
should be performed

- *Pros:* New version is slowly distributed over time, no downtime
- *Cons:* Breaking API changes can make consumers incompatible



# Blue/Green Deployment Strategy

*Deployment object with new version is created alongside*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-deploy
    version: 1.0.0
  name: blue
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: my-deploy
    version: 1.1.0
  name: green
```

- *Pros:* No downtime, traffic can be routed when ready
- *Cons:* Resource duplication, configuration of network routing



# Canary Deployment Strategy

*Two Deployment objects with different traffic distribution*

```
spec:  
  replicas: 3
```

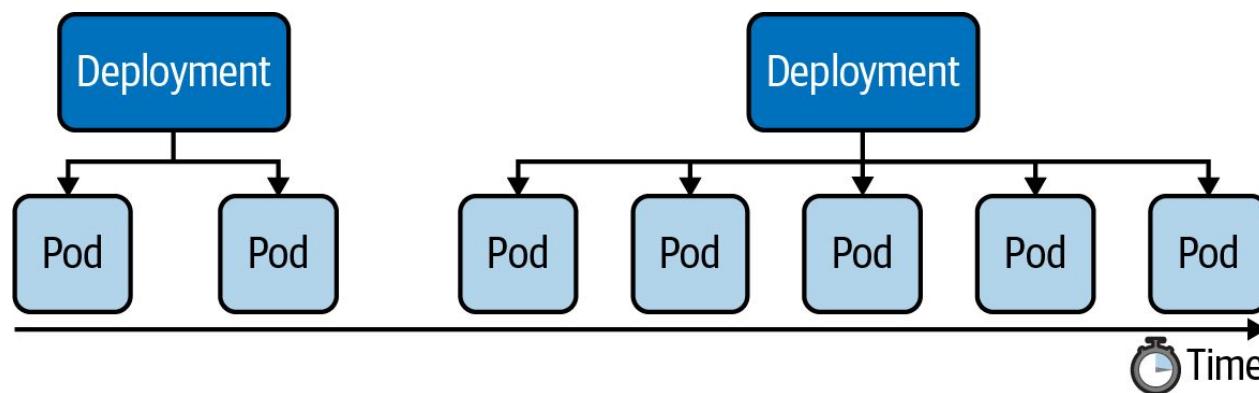
```
spec:  
  replicas: 1
```

- *Pros:* New version released to subset of users, A/B testing of features and performance
- *Cons:* May require a load balancer for fine-grained distribution



# Manually Scaling a Deployment

*“Load is increasing. We need to scale up the application.”*



# Changing the Number of Replicas

*Use `scale` command or change `replicas` attribute in live object*

```
$ kubectl scale deployment my-deploy --replicas=5
deployment.apps/my-deploy scaled
```

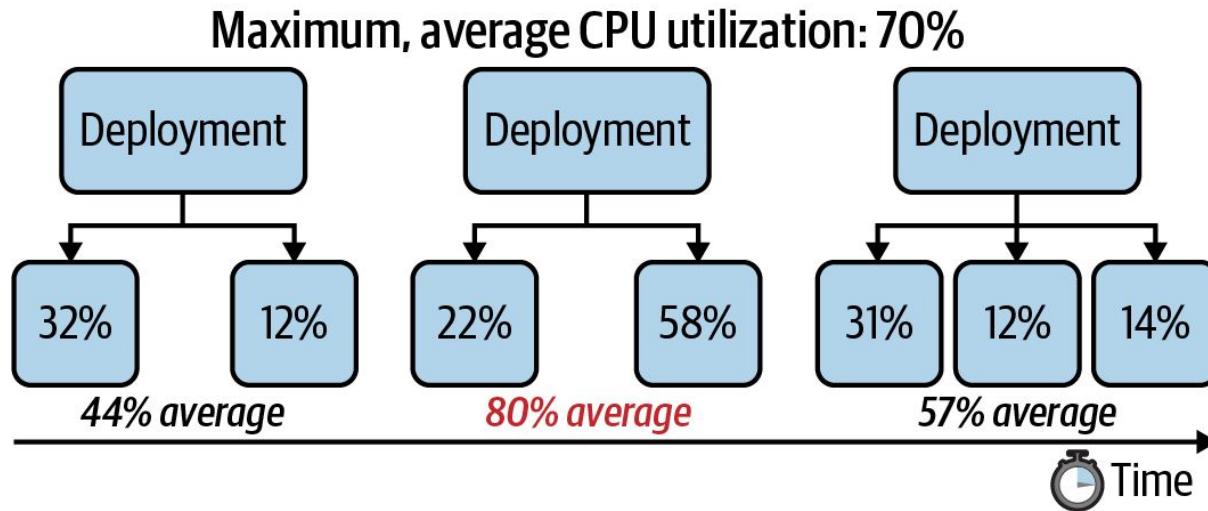
```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
my-deploy-8448c488b5-5f5tg	1/1	Running	0	44s
my-deploy-8448c488b5-9xplx	1/1	Running	0	44s
my-deploy-8448c488b5-d8q4t	1/1	Running	0	44s
my-deploy-8448c488b5-f5kkm	1/1	Running	0	44s
my-deploy-8448c488b5-mzx5g	1/1	Running	0	3d19h



# Autoscaling a Deployment

*“Don’t make me think. Auto-scale based on CPU utilization.”*



# Types of Autoscalers

*Decisions are made based on metrics*

- *Horizontal Pod Autoscaler (HPA)*: A standard feature of Kubernetes that scales the number of Pod replicas based on CPU and memory thresholds.
- *Vertical Pod Autoscaler (VPA)*: Scales the CPU and memory allocation for existing Pods based on historic metrics. Supported by a cloud provider as an add-on or needs to be installed manually.
- Both autoscalers use the metrics server. You need to install the component.



# Creating a HPA

*The imperative command is a quick way to create the object*

```
$ kubectl autoscale deployments my-deploy --cpu-percent=70 --min=2 --max=8
horizontalpodautoscaler.autoscaling/my-deploy autoscaled
```

```
$ kubectl get hpa my-deploy
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
my-deploy	Deployment/my-deploy	<unknown>/70%	2	8	2	37s



Changes with  
increasing traffic



---

# EXERCISE

Performing Rolling  
Updates and Scaling  
a Deployment



# Deploying an Application with Helm

Use Cases, Commands, Workflow



# What is Helm?

*Templating engine and package manager for a set of manifests*

- The artifact produced by the Helm executable is a so-called *chart file* bundling the manifests that comprise the API resources of an application.
- At runtime, it replaces placeholders in YAML template files with actual, end-user defined values.
- Chart files can be distributed to a chart repository e.g. [central chart repository](#) and consumed from there.



# Finding a Chart

*Searches the Artifact Hub for a chart with a matching name*

```
$ helm search hub jenkins
```

URL

<https://artifacthub.io/packages/helm/bitnami/jenkins>

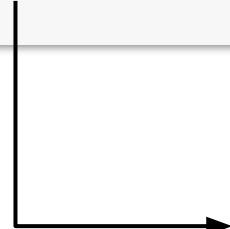
...

CHART

11.0.6

VERSION APP

2.361.2



The screenshot shows the Jenkins chart page on ArtifactHub. At the top, there's a navigation bar with links for DOCS, STATS, SIGN UP, and SIGN IN. Below the navigation, there's a search bar and a link to the Bitnami GitHub repository. The main content area features a large thumbnail of the Jenkins logo, followed by the text "jenkins" and "Jenkins packaged by Bitnami". A detailed description follows: "Jenkins is an open source Continuous Integration and Continuous Delivery (CI/CD) server designed to automate the building, testing, and deploying of any software project." Below this is an "OVERVIEW OF JENKINS" section. To the right of the main content, there are several buttons: INSTALL, TEMPLATES, DEFAULT VALUES, VALUES SCHEMA, and CHANGES LOG. At the bottom, there's a "T.L.D.R." section with command-line installation instructions: "helm repo add my-repo https://charts.bitnami.com/bitnami" and "helm install my-jenkins my-repo/jenkins". On the far right, there's a sidebar titled "APPLICATION VERSION" showing "2.361.2" and a "DRAFT VERSIONS" section listing three versions: "11.0.6" (the current version), "11.0.4" (published 20 Mar 2022), and "11.0.4" (published 20 Mar 2022). A "See all logs" link is also present.



# Installing a Chart

*You may have to add an external repository that hosts chart file*

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories

$ helm install jenkins bitnami/jenkins
...
CHART NAME: jenkins
CHART VERSION: 11.0.6
APP VERSION: 2.361.2

** Please be patient while the chart is being deployed **
```



# Listing Installed Charts

*Information about chart details*

```
$ helm list
```

NAME	NAMESPACE	REVISION	...	STATUS	CHART	APP VERSION
jenkins	default	1		deployed	jenkins-11.0.6	2.361.2

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
pod/bitnami-jenkins-764b44cc99-twf6f	1/1	Running	0	7m34s

```
...
```



# Uninstalling a Chart

*Deletes Kubernetes objects controlled by chart*

```
$ helm uninstall jenkins
release "jenkins" uninstalled

$ kubectl get pods
No resources found in default namespace.
```



---

# Q & A



# Probes and Health Checks

Types of Probes, Use Cases, Configuration Options

---

# Understanding Health Probing

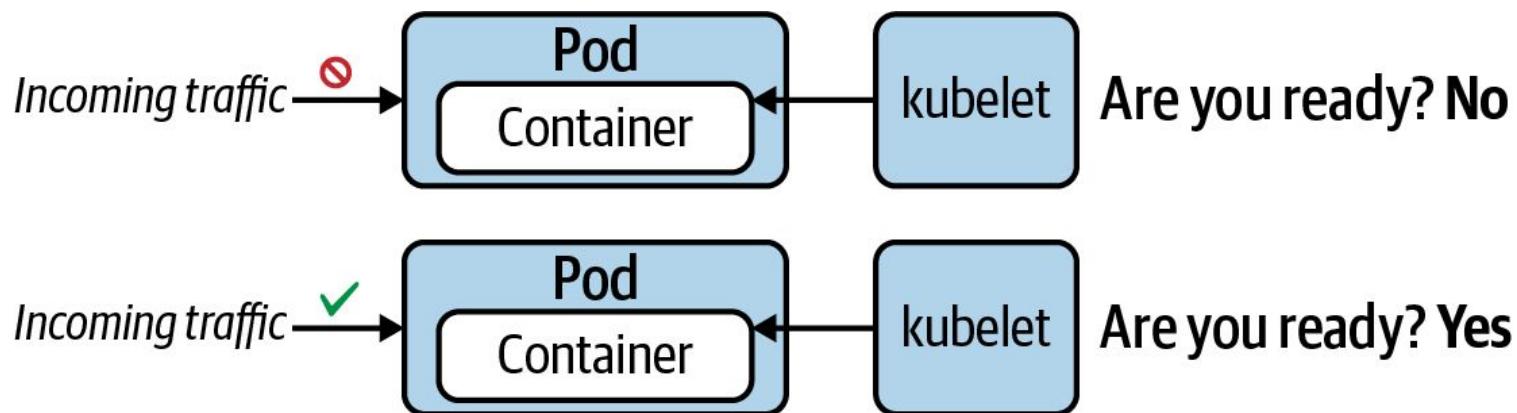
*Runtime issues can arise, but you check for them*

- Proper monitoring of applications running in production environments.
- Health probing automates the detection and correction of runtime issues.
- You can configure a container with *health probes* to execute a periodic mini-process that checks for certain conditions.



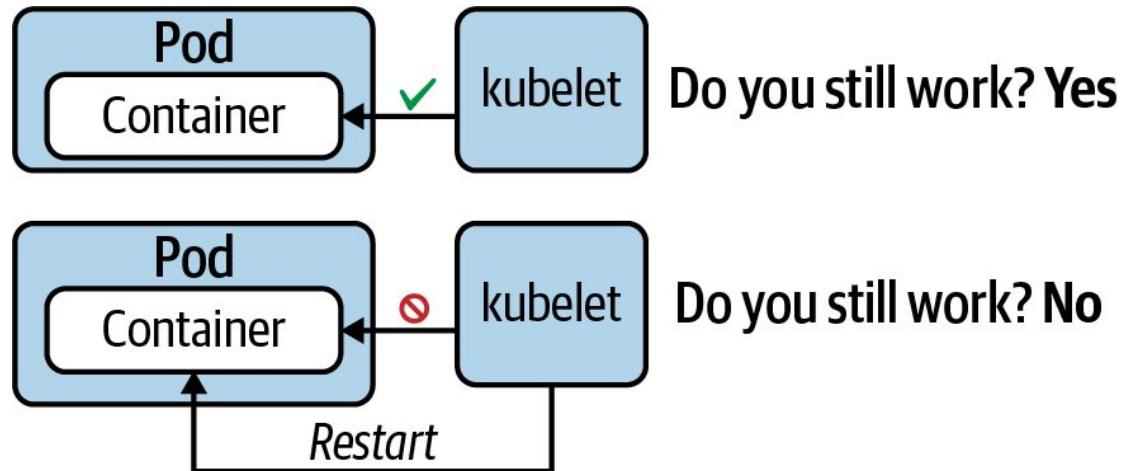
# Readiness Probe

*“Is application ready to serve requests?”*



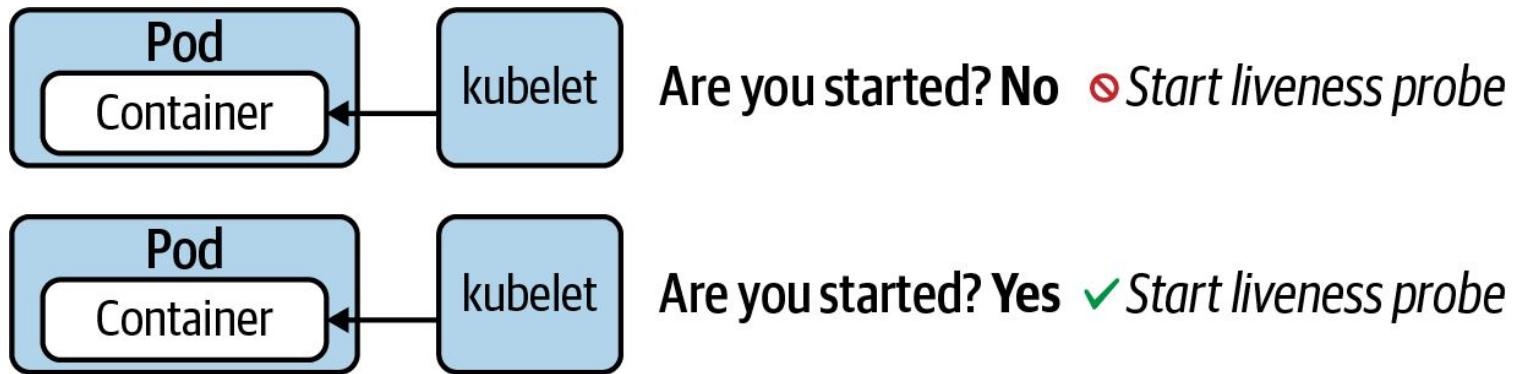
# Liveness Probe

*“Does the application still function without errors?”*



# Startup Probe

*“Legacy application may need longer to start. Hold off on probing.”*



# Health Verification Methods

Method	Option	Description
Custom Command	exec.command	Executes a command inside of the container e.g. a cat command and checks its exit code. Kubernetes considers an zero exit code to be successful. A non-zero exit code indicates an error.
HTTP GET Request	httpGet	Sends an HTTP GET request to an endpoint exposed by the application. A HTTP response code in the range of 200 and 399 indicates success. Any other response code is regarded as an error.
TCP socket connection	tcpSocket	Tries to open a TCP socket connection to a port. If the connection could be established, the probing attempt was successful. The inability to connect is accounted for as an error.



# Health Check Attributes

Attribute	Default Value	Description
initialDelaySeconds	0	Delay in seconds until first check is executed.
periodSeconds	10	Interval for executing a check (e.g., every 20 seconds).
timeoutSeconds	1	Maximum number of seconds until check operation times out.
successThreshold	1	Number of successful check attempts until probe is considered successful after a failure.
failureThreshold	3	Number of failures for check attempts before probe is marked failed and takes action.



# Defining a Readiness Probe

*HTTP probes are very helpful for web applications*

```
apiVersion: v1
kind: Pod
metadata:
  name: web-app
spec:
  containers:
  - name: web-app
    image: eshop:4.6.3
    readinessProbe:
      httpGet:
        path: /
        port: 8080
      initialDelaySeconds: 5
      periodSeconds: 2
```

Successful if HTTP status  
code is between 200 and 399



# Defining a Liveness Probe

*An event log can be queried by a custom command*

```
apiVersion: v1
kind: Pod
metadata:
  name: web-app
spec:
  containers:
    - name: web-app
      image: eshop:4.6.3
      livenessProbe:
        exec:
          command:
            - test `find /tmp/heartbeat.txt -mmin -1` > /dev/null
        initialDelaySeconds: 10
        periodSeconds: 5
```

Does the file get updated periodically by the application process?



# Defining a Startup Probe

*TCP socket connection if exposed by application*

```
apiVersion: v1
kind: Pod
metadata:
  name: startup-pod
spec:
  containers:
    - image: httpd:2.4.46
      name: http-server
      startupProbe:
        tcpSocket:
          port: 80
        initialDelaySeconds: 3
        periodSeconds: 15
```



Tries to open a TCP socket  
connection to a port



---

# EXERCISE

Defining a Pod's  
Readiness and  
Liveness Probe



---

# Q & A



# Monitoring and Troubleshooting

Metrics Server, Finding and Fixing Issues in Applications

---

# Monitoring Kubernetes

*What metrics are of interest?*

- Number of nodes in the cluster.
- Health status of nodes.
- Node performance metrics like CPU, memory, disk space, network.
- Pod-level performance metrics like CPU, memory consumption.



# Monitoring Solution

*Relevant to the exam: metrics server*

## Commercial Products



## Free Solutions

Heapster  
**X Retired**

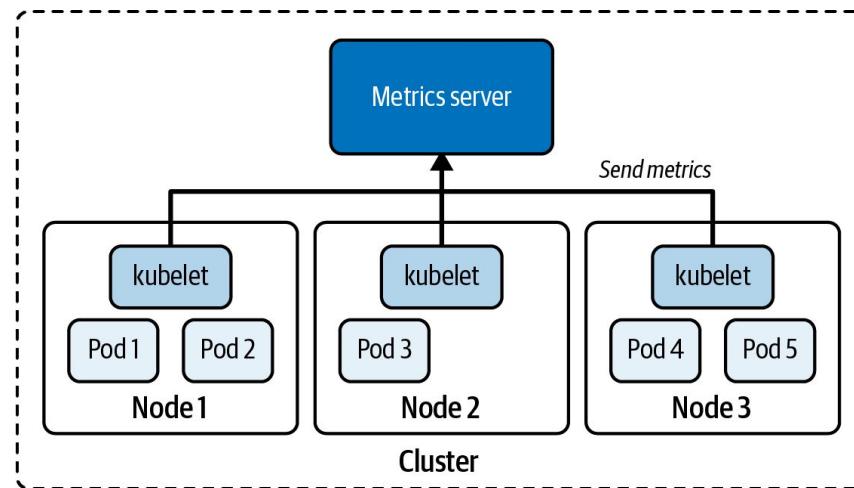
Metrics  
Server

VS.



# Metrics Server

*Cluster-wide metrics aggregator*



# Installing the Metrics Server

*Add on-component for Minikube or creating objects*



```
$ minikube addons enable metrics-server  
The 'metrics-server' addon is enabled
```



```
$ kubectl apply -f  
https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```



# Using the Metrics Server

*The kubectl top command can query nodes and Pods*

```
$ kubectl top nodes
NAME          CPU (cores)   CPU%     MEMORY (bytes)   MEMORY%
minikube      283m         14%      1262Mi          32%
```

```
$ kubectl top pod frontend
NAME          CPU (cores)   MEMORY (bytes)
frontend      0m           2Mi
```



# Verifying YAML Manifests

*List of [tools](#) available, [kubeval](#) is helpful for CI/CD pipelines*

```
$ kubeval -v 1.24.6 -s
  https://raw.githubusercontent.com/yannh/kubernetes-json-schema/master
  -d .

PASS - ch02/ingress/setup.yaml contains a valid Namespace (t75)
PASS - ch02/ingress/setup.yaml contains a valid Deployment
(t75.nginx-deployment)
PASS - ch02/ingress/setup.yaml contains a valid Service
(t75.accounting-service)
...
```



# Troubleshooting Pods

*Check the status first - is it running?*

```
$ kubectl get pods  
NAME      READY     STATUS    RESTARTS   AGE  
myapp     1/1       Running   0          12m
```

✓ Healthy status



# Common Pod Error Statuses

Status	Root cause	Potential fix
ImagePullBackOff or ErrImagePull	Image could not be pulled from registry	Check correct image name, check that image name exists in registry, verify network access from node to registry, ensure proper
CrashLoopBackOff	Application or command run in container crashes	Check command executed in container, ensure that image can properly execute (e.g. by creating a container with Docker)
CreateContainerConfigError	ConfigMap or Secret references by container cannot be found	Check correct name of the configuration object, verify the existence of the configuration object in the namespace



# Troubleshooting Pods

*Check the event log - does it indicate issues?*

```
$ kubectl describe pod myapp
...
Events:
  Type      Reason     Age            From           Message
  ----      -----     --            ----          -----
Normal    Scheduled   <unknown>      default-scheduler  Successfully<
assigned  default/secret-pod to minikube
Warning   FailedMount 3m15s          kubelet, minikube  Unable to<
attach or mount volumes: unmounted volumes=[mysecret], unattached<
volumes=[default-token-bf8rh mysecret]: timed out waiting for the condition
Warning   FailedMount  68s (x10 over 5m18s)  kubelet, minikube  ↴
MountVolume.SetUp failed for volume "mysecret" : secret "mysecret" not found
...

```

**✗ Failed mount**



# Troubleshooting Pods

*Check the container logs - do you see anything suspicious?*

```
$ kubectl logs myapp
...
2019-03-05 10:57:51.112 DEBUG  Receiving order
2019-03-05 10:57:51.112 INFO   Processing payment with ID 345993
2019-03-05 10:57:51.112 ERROR  Can't connect to payment system
```

✗ Connectivity issues

*Use the command line option --previous to gets the logs from the previous instantiation of a container after a restart*



---

# EXERCISE

Troubleshooting a  
Misconfigured Pod



---

# Q & A



# Custom Resource Definitions (CRD)

Discovery and Usage

# What is a CRD?

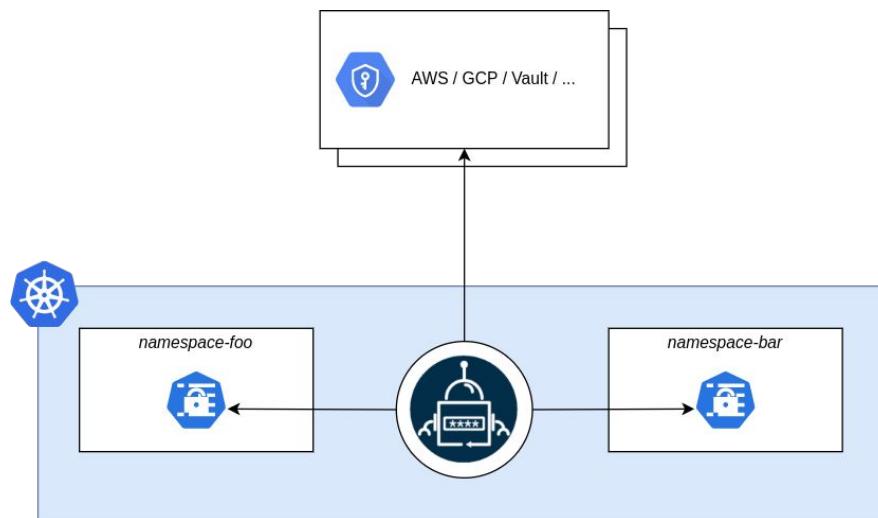
*Extension point for introducing custom API primitives*

- Some use cases with custom requirements cannot be fulfilled with the built-in Kubernetes primitives and functionality.
- A CustomResourceDefinition (CRD) allows you to define, create, and persist a custom object and expose it with the Kubernetes API.
- CRDs are combined with *controllers* to make them actually useful. Controllers enhance the core Kubernetes functionality.



# Example CRD

*Retrieving secrets from an external source ([Github](#))*



# Discovering CRDs

Once installed, you can list all available CRDs

```
$ kubectl get crd
```

NAME	CREATED AT
clusterexternalsecrets.external-secrets.io	2022-10-27T13:30:50Z
clustersecretstores.external-secrets.io	2022-10-27T13:30:50Z
externalsecrets.external-secrets.io	2022-10-27T13:30:50Z
secretstores.external-secrets.io	2022-10-27T13:30:50Z



# Discovering API Resources

*The installed CRD adds multiple custom resources*

\$ kubectl api-resources				
NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
clusterexternalsecrets	ces	external-secrets.io/v1beta1	false	ClusterExternalSecret
clustersecretstores	css	external-secrets.io/v1beta1	false	ClusterSecretStore
externalsecrets	es	external-secrets.io/v1beta1	true	ExternalSecret
secretstores	ss	external-secrets.io/v1beta1	true	SecretStore



# Inspecting a CRD

*Describes the API resource metadata and schema*

```
$ kubectl get describe secretstores.external-secrets.io -o yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: secretstores.external-secrets.io
spec:
  group: external-secrets.io
  scope: Namespaced
  names:
    categories:
      - externalsecrets
    kind: SecretStore
    listKind: SecretStoreList
    plural: secretstores
    shortNames:
      - ss
    singular: secretstore
  versions:
    name: v1beta1
    schema:
      openAPIV3Schema:
        ...

```



# Using a CRD

*SecretStore resource captures access to external secret store*

```
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: secretstore-sample
spec:
  provider:
    aws:
      service: SecretsManager
      region: us-east-1
      auth:
        secretRef:
          accessKeyIDSecretRef:
            name: awssm-secret
            key: access-key
          secretAccessKeySecretRef:
            name: awssm-secret
            key: secret-access-key
```

Usage of a custom kind  
and its API version

Usage of attributes  
defined by CRD



# Creating a SecretStore

*The object can be created and interacted with as usual*

```
$ kubectl create -f aws-secretstore.yaml
secretstore.external-secrets.io/secretstore-sample created

$ kubectl get secretstores
NAME           AGE      STATUS          READY
secretstore-sample   8m47s  InvalidProviderConfig  False

$ kubectl delete secretstore secretstore-sample
secretstore.external-secrets.io "secretstore-sample" deleted
```



# Role-Based Access Control

Defining Roles and RoleBindings for Users and Service Accounts

# What is RBAC?

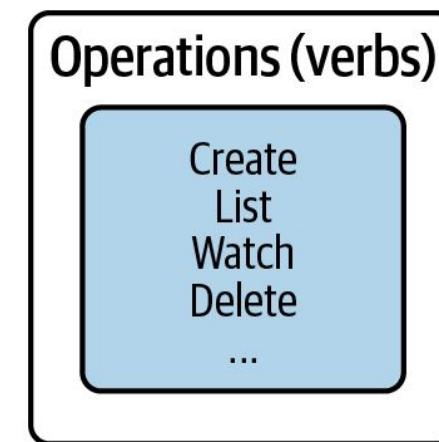
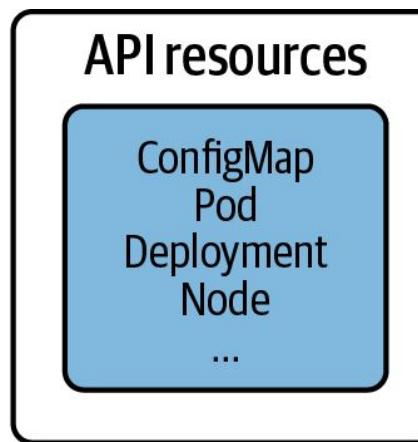
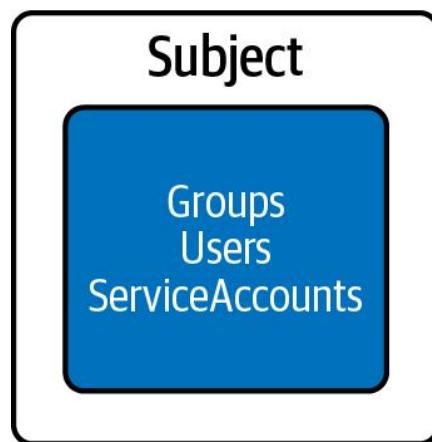
*Restricting access control for clients interacting with API server*

- Defines policies for users, groups, and processes by allowing or disallowing access to manage API resources.
- Enabling and configuring RBAC is mandatory for any organization with a strong emphasis on security.
- *Example:* “The human user john is only allowed to list and create Pods, and Deployments but nothing else.”



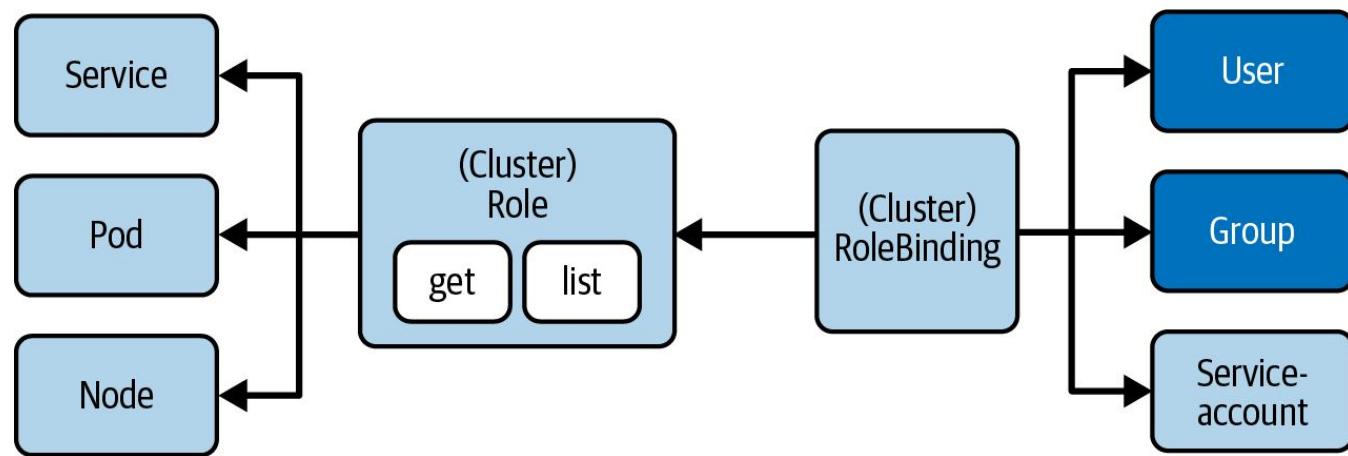
# RBAC High-Level Overview

*Three key elements for understanding concept*



# Involved RBAC Primitives

*Restrict access to API resources based on user roles*



# Creating a Role

*Defines verbs and resources in comma-separated list*

```
$ kubectl create role read-only --verb=list,get,watch  
  --resource=pods,deployments,services  
role.rbac.authorization.k8s.io/read-only created
```

Resources: Primitives the operations should apply to

Operations: Only allow listing, getting, watching



# Role YAML Manifest

*Can define a list of rules in an array*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["list", "get", "watch"]
- apiGroups: ["apps"] ←
  resources: ["deployments"]
  verbs: ["list", "get", "watch"]
```

Resources with groups need  
to spelled out explicitly



# Getting Role Details

*Maps objects of a type to verbs*

```
$ kubectl describe role read-only
Name:           read-only
Labels:         <none>
Annotations:   <none>
PolicyRule:
  Resources      Non-Resource URLs  Resource Names  Verbs
  -----          -----
  pods           [ ]                  [ ]            [list get watch]
  services        [ ]                  [ ]            [list get watch]
  deployments.apps [ ]                [ ]            [list get watch]
```



# Creating a RoleBinding

*Map subject to Role*

```
$ kubectl create rolebinding read-only-binding --role=read-only  
  --user=johndoe  
rolebinding.rbac.authorization.k8s.io/read-only-binding created
```

*Subject:* Binds a user to  
the Role

*Role:* Reference the name of  
the object



# RoleBinding YAML Manifest

*Roles can be mapped to multiple subjects if needed*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-only-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: read-only
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: john Doe
```

← Reference to Role

← Reference to User



# Getting RoleBinding Details

*Only shows mapping between Role and subjects*

```
$ kubectl describe rolebinding read-only-binding
Name:           read-only-binding
Labels:         <none>
Annotations:   <none>
Role:
  Kind:  Role
  Name:  read-only
Subjects:
  Kind  Name      Namespace
  ----  --       -----
  User  johndoe
```



# Cluster-wide RBAC

*Apply Roles and RoleBindings across multiple namespaces*

- The configuration elements are effectively the same. The only difference is the value of the `kind` attribute.
- To define a cluster-wide Role, use the imperative subcommand `clusterrole` or the kind `ClusterRole` in the YAML manifest.
- To define a cluster-wide RoleBinding, use the imperative subcommand `clusterrolebinding` or the kind `ClusterRoleBinding` in the YAML manifest.



# Aggregated ClusterRoles

*Combine multiple ClusterRole rules into one*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pods-services-aggregation-rules
  namespace: rbac-example
aggregationRule:
  clusterRoleSelectors:
  - matchLabels:
      rbac-pod-list: "true"
  - matchLabels:
      rbac-service-delete: "true"
rules: []
```

Select other  
ClusterRoles by  
labels



# What is a Service Account?

*Process in a container needs to interact with Kubernetes API*

- Pods use a Service Account to authenticate with the API server through an authentication token.
- If not assigned explicitly, a Pod uses the default Service Account. The default Service Account has the same permissions as an unauthenticated user.
- *Example:* “I have a CI/CD process that should interact with the Kubernetes API using an authentication token.”



# Service Account as Subject

*Usage needs to be configured in Pod*



# ServiceAccount YAML Manifest

*Enables authentication token auto-mounting by default*

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa-api
  namespace: k97
```

The token file will be available to any Pod that uses the Service Account at  
`/var/run/secrets/kubernetes.io/serviceaccount/token`



# Using Service Account in Pod

*Enables authentication token auto-mounting by default*

```
apiVersion: v1
kind: Pod
metadata:
  name: list-pods
  namespace: k97
spec:
  serviceAccountName: sa-api
  containers:
    - name: pods
      image: alpine/curl:3.14
      command: ['sh', '-c', 'while true; do curl -s -k -m 5 -H<br>"Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/<br>serviceaccount/token)" https://kubernetes.default.svc.cluster.<br>local/api/v1/namespaces/k97/pods; sleep 10; done']
```

Assigned name of  
Service Account

List all Pods in the  
namespace k97  
via an API call

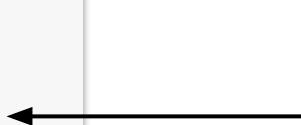


# RoleBinding YAML Manifest

*Default RBAC policies don't span beyond kube-system*

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: serviceaccount-pod-rolebinding
  namespace: k97
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: list-pods-clusterrole
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: ServiceAccount
  name: sa-api
```

Maps the Service  
Account as a  
subject



---

# EXERCISE

Regulating Access  
to API Resources  
with RBAC



---

# Q & A



# Resource Requirements and Boundaries

Container Resource Requests and Limits, Resource Quotas for Namespaces

# Handling Resources

*It's best practice to make statement about resource consumption*

- Containers can define a minimum amount of resources needed to run the application, as well as the maximum amount of resources the application is allowed to consume.
- Application developers should determine the right-sizing with load tests or at runtime by monitoring the resource consumption.
- Enforcement of resources can be constrained on a namespace-level with a ResourceQuota.



# Resource Units in Kubernetes

*CPU units and memory as fixed-point number or power-of-two equivalents*

Kubernetes measures CPU resources in millicores and memory resources in bytes. That's why you might see resources defined as 600m or 100Mib.

For a deep dive on those resource units, it's worth cross-referencing the section "[Resource units in Kubernetes](#)" in the official documentation.



---

# Container Resource Requests

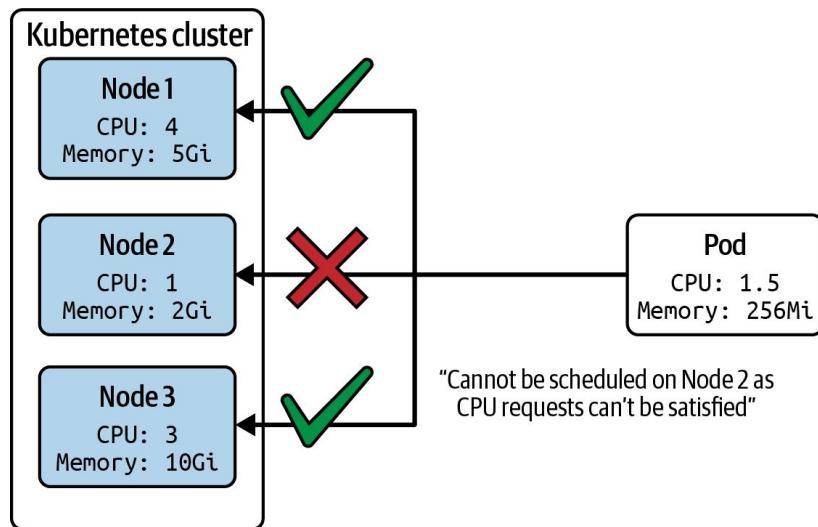
*Definition and runtime effects*

- Containers can define a minimum amount of resources needed to run the application via `spec.containers[] .resources.requests`.
- Resource types include CPU, memory, huge page, ephemeral storage.
- If Pod cannot be scheduled on any node due to insufficient resource availability, you may see a `PodExceedsFreeCPU` or `PodExceedsFreeMemory` status.



# Example Scheduling Scenario

*Node's resource capacity needs to be able to fulfill it*



# Resource Requests YAML Manifest

*Minimum amount of resources define for a container*

```
apiVersion: v1
kind: Pod
metadata:
  name: rate-limiter
spec:
  containers:
    - name: business-app
      image: bmuschko/nodejs-business-app:1.0.0
      ports:
        - containerPort: 8080
      resources:
        requests:
          memory: "256Mi"
          cpu: "1"
```



---

# Container Resource Limits

## *Definition and runtime effects*

- Containers can define a maximum amount of resources allowed to be consumed by the application via `spec.containers[] .resources.limits`.
- Resource types include CPU, memory, huge page, ephemeral storage.
- Container runtime decides how to handle situation where application exceeds allocated capacity, e.g. termination of application process.



# Resource Limits YAML Manifest

*Do not allow more than the allotted resource amounts*

```
apiVersion: v1
kind: Pod
metadata:
  name: rate-limiter
spec:
  containers:
    - name: business-app
      image: bmuschko/nodejs-business-app:1.0.0
      ports:
        - containerPort: 8080
      resources:
        limits:
          memory: "512Mi"
          cpu: "2"
```



# Pod Scheduling Details

*After scheduling a Pod, you can check on runtime information*

```
$ kubectl get pod rate-limiter -o yaml | grep nodeName:  
nodeName: worker-3  
  
$ kubectl describe node worker-3  
...  
Non-terminated Pods:           (3 in total)  
  Namespace          Name            CPU Requests  CPU Limits  ...  
  ----              --            -----        -----      ...  
  default           rate-limiter   1250m (62%)  0 (0%)    ...
```



# What is a ResourceQuota?

*Defines resource constraints per namespace*

- Constraints that limit aggregate resource consumption or limit the quantity of objects that can be created.
- `requests.cpu/memory`: Across all pods in a non-terminal state, the sum of CPU/memory requests cannot exceed this value.
- `limits.cpu/memory`: Across all pods in a non-terminal state, the sum of CPU/memory limits cannot exceed this value.



# ResourceQuota YAML Manifest

*# of objects and limit aggregate resource consumption*

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: awesome-quota
  namespace: team-awesome
spec:
  hard:
    pods: 2
    requests.cpu: "1"
    requests.memory: 1024Mi
    limits.cpu: "4"
    limits.memory: 4096Mi
```



# No Requests/Limits Definition

*Scheduler rejects the creation of the object*

```
$ kubectl create -f nginx-pod.yaml -n team-awesome
Error from server (Forbidden): error when creating "nginx-pod.yaml": pods "nginx" is forbidden: failed quota: awesome-quota: must specify limits.cpu,limits.memory,requests.cpu,requests.memory
```



# Exceeds Requests/Limits

*Scheduler rejects the creation of the object*

```
$ kubectl create -f nginx-pod.yaml -n team-awesome
Error from server (Forbidden): error when creating "nginx-pod.yaml":  
pods "nginx" is forbidden: exceeded quota: awesome-quota, requested:  
pods=1,requests.cpu=500m,requests.memory=512Mi, used: pods=2,requests.cpu=1,  
requests.memory=1024Mi, limited: pods=2,requests.cpu=1,requests.memory=1024Mi
```



# Inspecting a ResourceQuota

*Consumed resources and defined hard limits*

```
$ kubectl describe resourcequota awesome-quota -n team-awesome
Name:           awesome-quota
Namespace:      team-awesome
Resource        Used   Hard
-----
limits.cpu      2       4
limits.memory   2048m  4096m
pods            2       2
requests.cpu    1       1
requests.memory 1024m  1024m
```



---

# EXERCISE

Defining a Pod's  
Resource  
Requirements



---

# Q & A



# ConfigMaps and Secrets

Creating and Consuming Configuration Data

# Defining Configuration Data

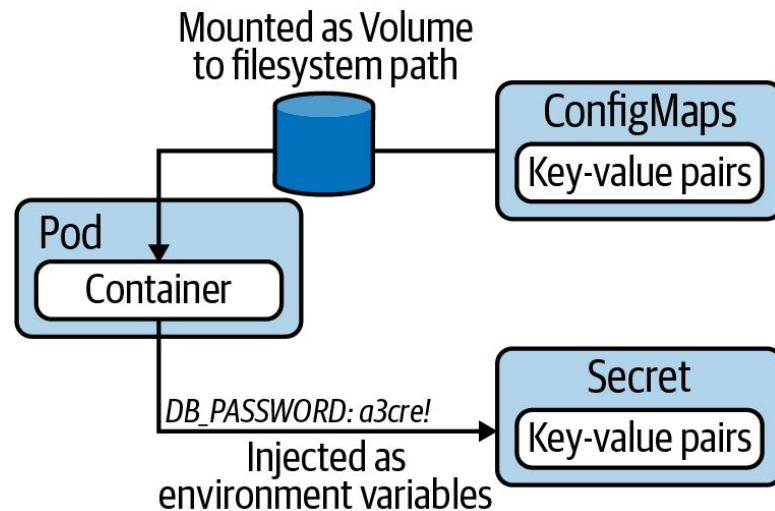
*Control runtime behavior with centralized information*

- Stored as key-value pairs in dedicated Kubernetes primitives with a lifecycle decoupled from consuming Pod.
- *ConfigMap*: Plain-text values suited for configuring application e.g. flags, or URLs.
- *Secret*: Base64-encoded values suited for storing sensitive data like passwords, API keys, or SSL certificates. Values are not encrypted!



# Consuming ConfigMap & Secret

*Mounting as a Volume or injecting as env. variables*



# ConfigMap Data Sources

*Imperative command provides options for sourcing data*

- `--from-literal`: Literal values, which are key-value pairs as plain text.
- `--from-env-file`: A file that contains key-value pairs and expects them to be environment variables.
- `--from-file`: A file with arbitrary contents.
- `--from-file`: A directory with one or many files.



# Creating a ConfigMap

*Fast, easy and flexible, can point to different sources*

```
$ kubectl create configmap db-config --from-literal=db=staging  
configmap/db-config created
```

```
$ kubectl create configmap db-config --from-env-file=config.env  
configmap/db-config created
```

```
$ kubectl create configmap db-config --from-file=config.txt  
configmap/db-config created
```

```
$ kubectl create configmap db-config --from-file=app-config  
configmap/db-config created
```



# ConfigMap YAML Manifest

*Definition of a ConfigMap is fairly short and on point*

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  db: staging
  username: jdoe
```



# Consuming as Env. Variables

*Convenient if ConfigMap reflects the desired syntax*

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - image: nginx
      name: backend
      envFrom:
        - configMapRef:
            name: db-config
```

```
$ kubectl exec -it nginx -- env
db=staging
username=jdoe
...
```



# Consuming as a Volume

*Use Volume type configMap*

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - name: backend
      image: nginx
      volumeMounts:
        - name: config-volume
          mountPath: /etc/config
volumes:
  - name: config-volume
    configMap:
      name: db-config
```

```
$ kubectl exec -it backend -- /bin/sh
# ls /etc/config
db
username
# cat /etc/config/db
staging
```



---

# EXERCISE

Configuring a Pod  
to Use a ConfigMap



# Secret Options

*Imperative command:* `kubectl create secret`

Option	Description
<code>generic</code>	Creates a secret from a file, directory, or literal value.
<code>docker-registry</code>	Creates a secret for use with a Docker registry.
<code>tls</code>	Creates a TLS secret.



---

# Secret Data Sources

*Imperative generic command provides options for sourcing data*

- `--from-literal`: Literal values, which are key-value pairs as plain text.
- `--from-env-file`: A file that contains key-value pairs and expects them to be environment variables.
- `--from-file`: A file with arbitrary contents.
- `--from-dir`: A directory with one or many files.



# Creating a Secret

*Values are base64-encoded automatically*

```
$ kubectl create secret generic db-creds --from-literal=pwd=s3cre!  
secret/db-creds created  
  
$ kubectl create secret generic db-creds --from-env-file=secret.env  
secret/db-creds created  
  
$ kubectl create secret generic db-creds --from-file=id_rsa=~/.ssh/id_rsa  
secret/db-creds created
```



# Secret YAML Manifest

*Value has to be base64-encoded manually*

```
$ echo -n 's3cret' | base64  
czNjcmUh
```

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: mysecret  
type: Opaque  
data:  
  pwd: czNjcmUh
```

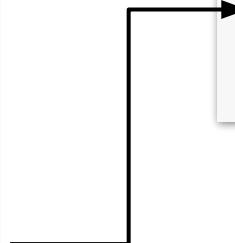


# Consuming as Volume

*Inside of the container, values are base64-decoded*

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - name: backend
      image: nginx
      volumeMounts:
        - name: secret-volume
          mountPath: /etc/secret
  volumes:
    - name: secret-volume
      secret:
        secretName: mysecret
```

```
$ kubectl exec -it backend -- /bin/sh
# ls /etc/secret
pwd
# cat /etc/secret/pwd
s3cre!
```



# Specialized Secret Types

*The table only lists some, more in [docs](#)*

Type	Description
kubernetes.io/basic-auth	Credentials for basic authentication
kubernetes.io/ssh-auth	Credentials for SSH authentication
kubernetes.io/service-account-token	ServiceAccount token



# Plain-Text Secret Values

*The `stringData` attribute allows for plain-text values*

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth
stringData:
  username: admin
  password: t0p-Secret
```



---

# EXERCISE

Configuring a Pod  
to Use a Secret



---

# Q & A



# Security Contexts for Pods

Defining Privilege and Access Control

---

# What is a Security Context?

*Privilege and access control settings for a Pod or container*

- Modeled as a set of attributes under the attribute `securityContext` within the Pod specification.
- Can be applied to all containers in Pod or for individual containers.
- *Example:* “This container needs to run with a non-root user.”



# Defining a Security Context

*Pod- vs. container-level definition*

```
apiVersion: v1
kind: Pod
metadata:
  name: secured-pod
spec:
  securityContext:
    runAsUser: 1000
  containers:
    - image: nginx:1.18.0
      name: secured-container
      securityContext:
        runAsGroup: 3000
```

Defined on the Pod-level

Defined on the container-level



# Security Context API

*Only partial overlap for Pod- and container attributes*

API	Description
<a href="#"><u>PodSecurityContext</u></a>	Defines Pod-level security attributes.
<a href="#"><u>SecurityContext</u></a>	Defines container-level security attributes.



---

# EXERCISE

Creating a Security  
Context for a Pod



---

# Q & A



# Services

Service Types and Troubleshooting

# What is a Service?

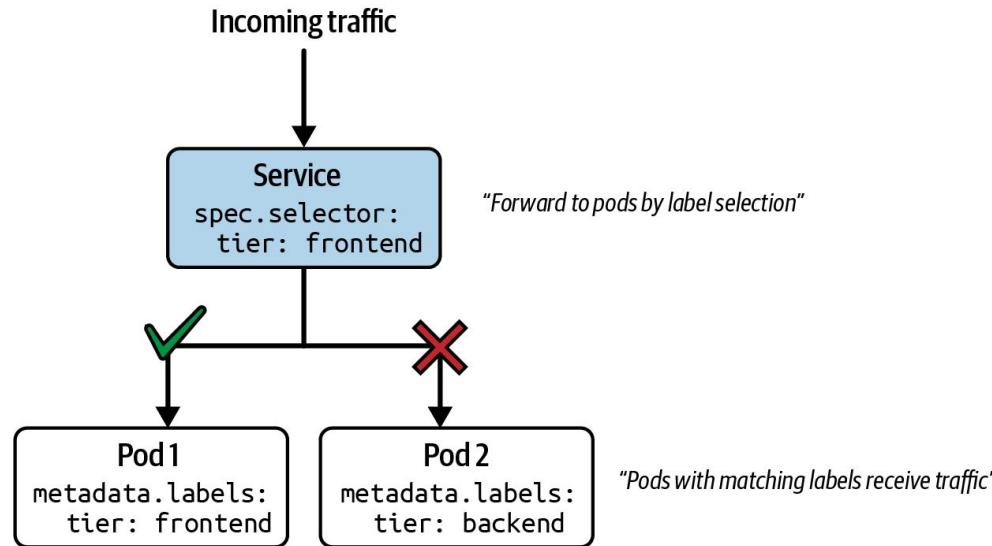
*Discoverable networking capability for Pods*

- A Pod's IP address is not considered stable over time. A Pod restart leases a new IP address.
- Building a microservices architecture on Kubernetes requires network interfaces stable over time.
- A Service provides discoverable names and load balancing to a set of Pods.



# Request Routing

*“How does a service decide which Pod to forward the request to?”*



# Different Types of Services

Type	Behavior
ClusterIP	Exposes the service on a cluster-internal IP. Only reachable from within the cluster.
NodePort	Exposes the service on each node's IP at a static port. Accessible from outside of the cluster.
LoadBalancer	Exposes the service externally using a cloud provider's load balancer.



# Creating a Service

*Ensure that Pod labels and Service label selector match*

```
$ kubectl run echoserver --image=k8s.gcr.io/echoserver:1.10 --restart=Never  
  --port=8080  
pod/echoserver created

$ kubectl create service clusterip echoserver --tcp=80:8080  
service/echoserver created

$ kubectl run echoserver --image=k8s.gcr.io/echoserver:1.10 --restart=Never  
  --port=8080 --expose  
service/echoserver created  
pod/echoserver created
```



Shortcut for creating a Pod + Service



# Service YAML Manifest

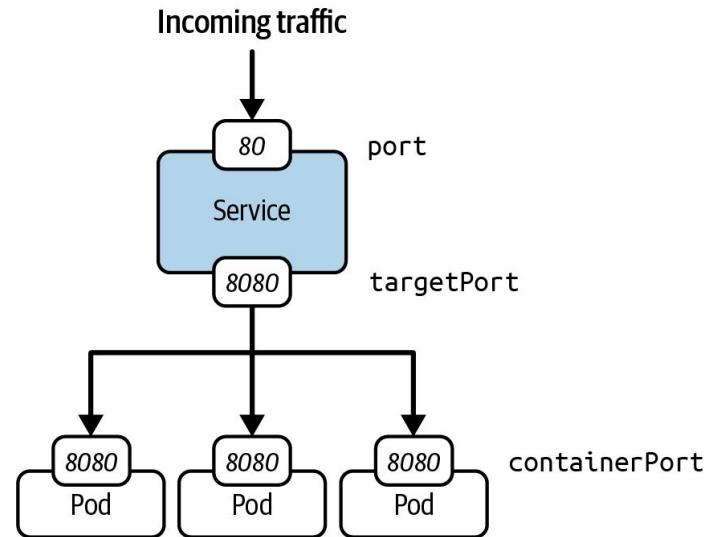
*Defines containers and their images running in them*

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
spec:
  type: ClusterIP ← Service type
  selector:
    app: echoserver ← Label selection for Pods
  ports:
  - port: 80
    targetPort: 8080 ← Mapping of incoming to outgoing port
```



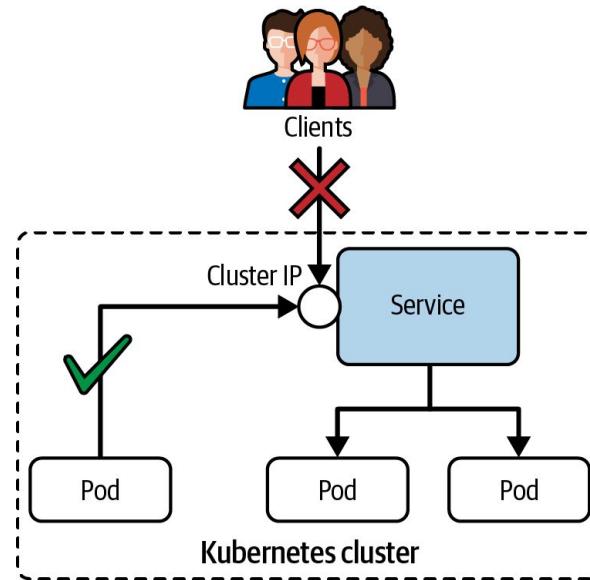
# Port Mapping

*“How to map the service port to the container port in Pod?”*



# ClusterIP Service Type

*Only reachable from within the cluster, e.g. another Pod*



# Accessing a ClusterIP Service

*Use the combination of cluster IP and incoming service port*

```
$ kubectl get pod,service
NAME                  READY   STATUS    RESTARTS   AGE
pod/echoserver        1/1     Running   0          23s

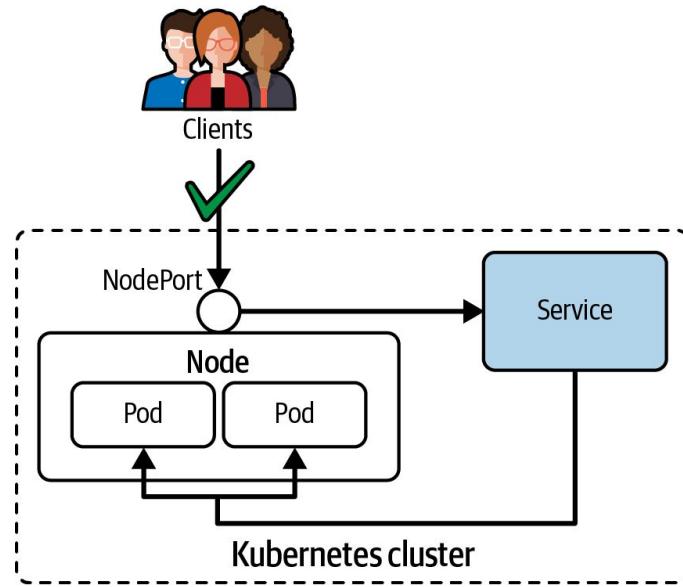
NAME            TYPE      CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
service/echoserver  ClusterIP  10.96.254.0   <none>       5005/TCP   8s

$ kubectl run tmp --image=busybox --restart=Never -it --rm<
  -- wget 10.96.254.0:5005
Connecting to 10.96.254.0:5005 (10.96.254.0:5005)
...
```



# NodePort Service Type

*Can be resolved from outside of the Kubernetes cluster*



# Accessing a NodePort Service

*Use the combination of node IP and statically-assigned port*

```
$ kubectl get pod,service
NAME                  READY   STATUS    RESTARTS   AGE
pod/echoserver        1/1     Running   0          23s

NAME              TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
service/echoserver  NodePort  10.96.254.0  <none>        5005:30158/TCP   8s

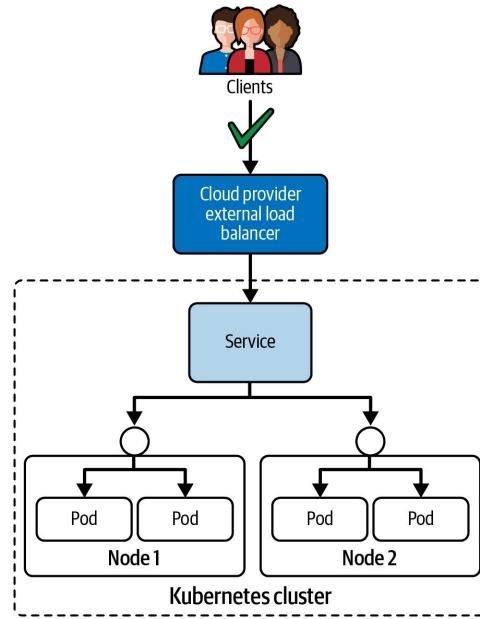
$ kubectl get nodes -o<
  jsonpath='{ $.items[*].status.addresses[?(@.type=="InternalIP")].address }'
192.168.64.15

$ wget 192.168.64.15:30158
Connecting to 192.168.64.15:30158... connected.
...
```



# LoadBalancer Service Type

*Routes traffic from existing, external load balancer to Service*



# Accessing a LoadBalancer Service

*Use the combination of node IP and statically-assigned port*

```
$ kubectl get pod,service
NAME                  READY   STATUS    RESTARTS   AGE
pod/echoserver        1/1     Running   0          23s

NAME              TYPE            CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
service/echoserver  LoadBalancer  10.109.76.157  10.109.76.157  5005:30642/TCP   5s
$ wget 10.109.76.157:5005
Connecting to 10.109.76.157:5005... connected.
...
```



# Troubleshooting Services

*Check Service type and call endpoint*

```
$ kubectl get service nginx
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
nginx    NodePort  10.105.201.83  <none>        80:30184/TCP  3h
```

```
$ curl http://nginx:30184
curl: (6) Could not resolve host: nginx
```

 Connectivity issue



# Troubleshooting Services

*Ensure correct label selection*

```
$ kubectl describe service myapp
Name:           myapp
Namespace:      default
Labels:         app=myapp
Annotations:    <none>
Selector:    app=myapp
Type:          ClusterIP
IP:            10.102.22.26
Port:          80-80  80/TCP
TargetPort:    80/TCP
Endpoints:     10.0.0.115:80
Session Affinity: None
Events:        <none>
```

```
$ kubectl describe pod
myapp
Name:           myapp
...
Labels:        app=myapp
...
```

✓ Matching labels



---

# EXERCISE

Routing traffic to  
Pods from Inside  
and Outside of a  
Cluster



---

# Q & A



# Ingress

Purpose, Routing Rules, Access

# What is an Ingress?

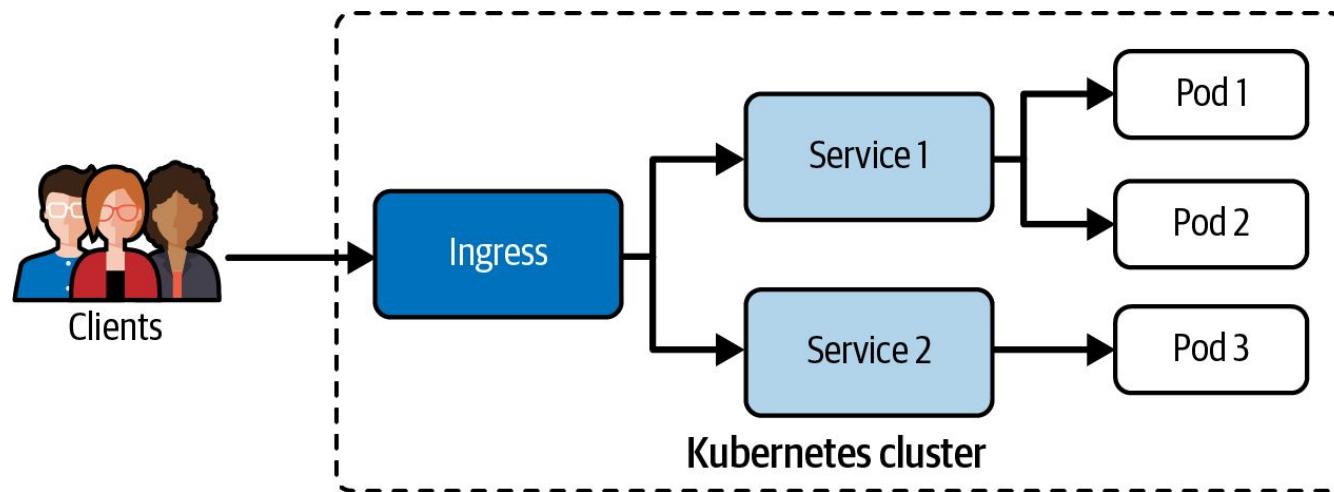
*Route HTTP(S) traffic from the outside of cluster to Service*

- It's not a specific Service type, nor should it be confused with the Service type LoadBalancer.
- Defines rules for mapping URL context path to Service object.
- An Ingress cannot work without an Ingress controller. The Ingress controller evaluates the collection of rules defined by an Ingress that determine traffic routing.



# Ingress Traffic Routing

*Can involve one or many Services as backend*



# Creating an Ingress

*The rule definition requires intricate knowledge of syntax*

```
$ kubectl create ingress corellian  
  --rule="star-alliance.com/corellian/api=corellian:8080"  
ingress.networking.k8s.io/corellian created
```



# Ingress YAML Manifest

*The host definition is optional*

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: corellian
spec:
  rules:
  - host: star-alliance.com
    http:
      paths:
      - backend:
          service:
            name: corellian
            port:
              number: 8080
        path: /corellian/api
        pathType: Exact
```



# Ingress Rules

*Traffic routing is controlled by rules defined on Ingress resource*

- Optional host. If no host is defined, then all inbound HTTP traffic is handled.
- A list of paths e.g. /testpath.
- The backend, a combination of Service name and port.



# Path Types

*Incoming URLs match based on type*

	<i>Rule</i>	<i>Request</i>
<b>Exact</b>	/foo	/foo ✓ /bar ✗
<b>Prefix</b>	/foo	/foo, /foo/ ✓ /bar ✗



# Accessing an Ingress

*Use host name and context path*

```
$ wget star-alliance.com/corellian/api/ --timeout=5 --tries=1
Resolving star-alliance.com (star-alliance.com) ... 192.168.64.15
Connecting to star-alliance.com (star-alliance.com)|192.168.64.15|:80... connected.
...
...
```



---

# EXERCISE

Defining and Using  
an Ingress



---

# Q & A



# Network Policies

Purpose, Rules, Best Practices

# What is a Network Policy?

*Firewall rules for Pod-to-Pod communication*

- Communication between Pods in the same cluster is unrestricted. You can use the Pod's IP address to communicate with it.
- Network policies implement the principle of least privilege. Only allow communication is needed to fulfill requirement of architecture.
- You can disallow and allow network communication with fine-grained rules.

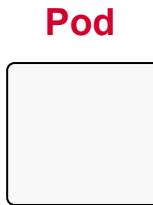


# Example Network Policies

*“Network Policies control traffic from and to the Pod”*

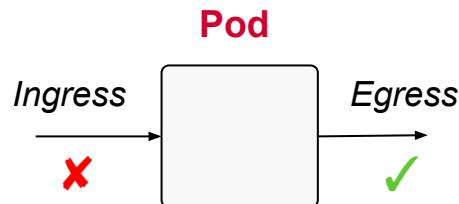


# Network Policy Rules



tier: frontend

*Which Pod does the rule apply to?* ✓



*Which direction of traffic?  
Who is allowed?*



# Deny All Policy YAML Manifest

*Start with default deny policy and then allow traffic*

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



# Allow Policy YAML Manifest

*kubectl doesn't provide an imperative command*

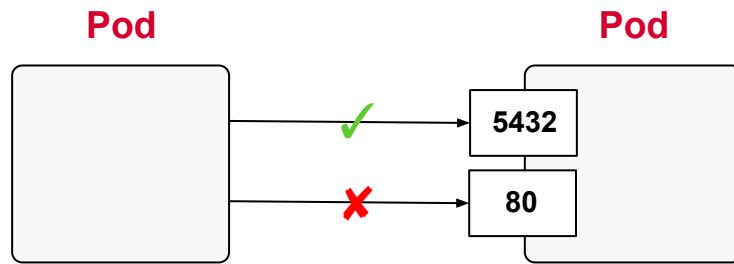
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow
spec:
  podSelector:
    matchLabels:
      app: payment-processor
      role: api
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: coffeeshop
```

Selects the labels of Pods  
the rule should apply to

Only defines rules for  
incoming traffic



# Restricting Access to Ports



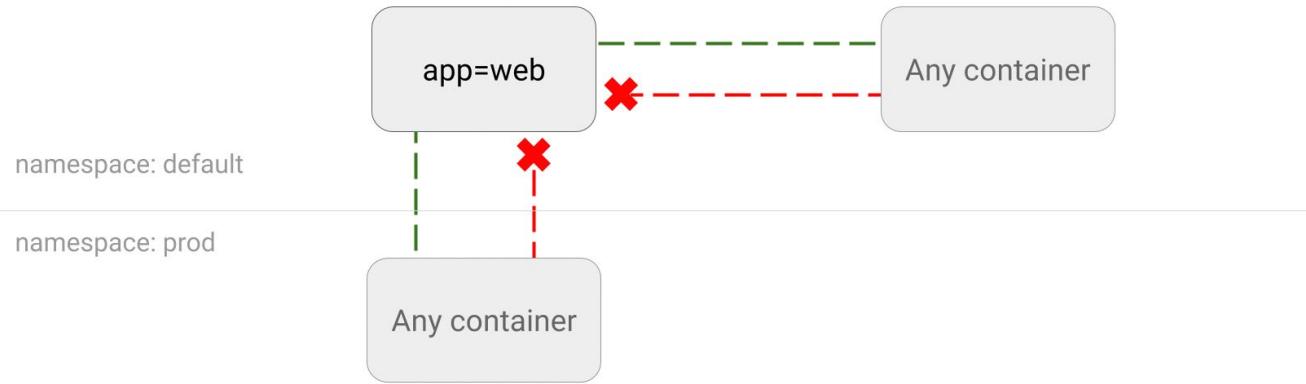
*By default all ports are open*

```
...  
ingress:  
- from:  
  - podSelector:  
    matchLabels:  
      tier: backend  
  ports:  
    - protocol: TCP  
    port: 5432  
...
```



# Additional Learning Resource

*Network Policies explained by use case and visualization*



<https://github.com/ahmetb/kubernetes-network-policy-recipes>



---

# EXERCISE

Restricting Access  
to and from a Pod



---

# Q & A



# Summary & Wrap Up

Last words of advice...

---

# Gaining confidence

- Run through practice exams as often as you can
- Read through online documentation start to end
- Know your tools (especially vim, bash, YAML)
- Pick time you are most comfortable, get enough sleep
- Take your first attempt easy but give it your best



---

# Q & A





Thank you

