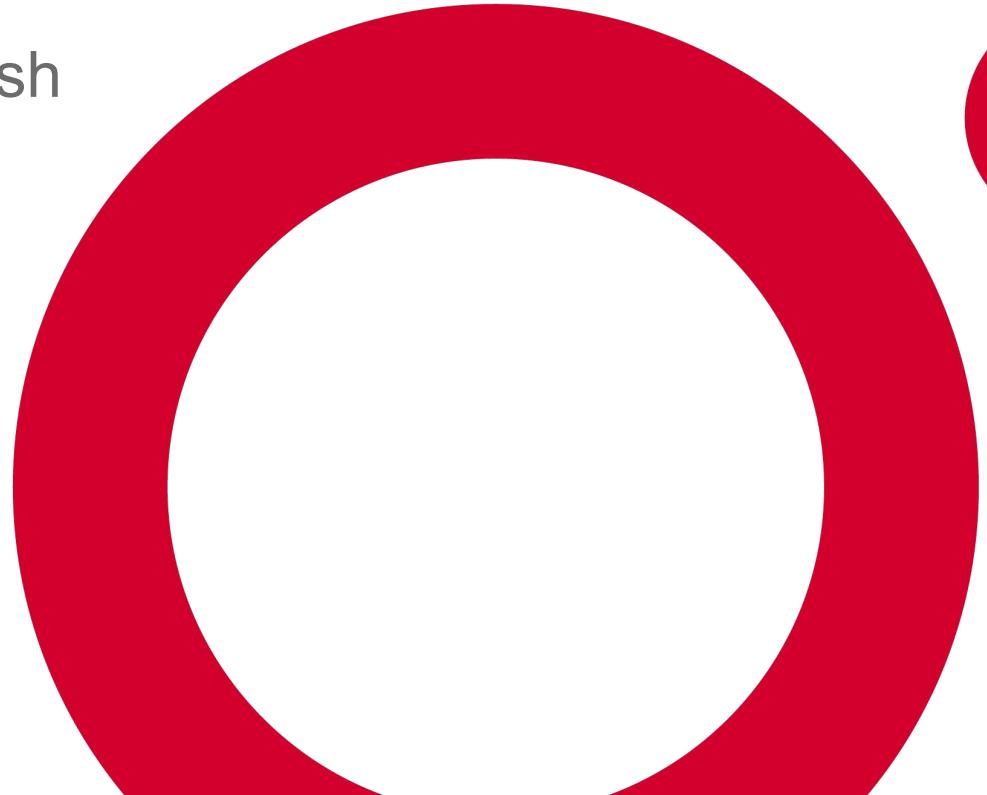




# Certified Kubernetes Administrator (CKA) Crash Course

*Curriculum September 2020*



# About the trainer



**bmuschko**



**bmuschko**



**bmuschko.com**



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

# Exam Details and Resources

Objectives, Environment, Time Management

# Kubernetes Certification Path



Associate

Entry Level



Developer

Practitioner



Administrator

Practitioner



# Exam Objectives

*“Perform typical responsibilities of a Kubernetes administrator.”*



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

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



# Exam Domains & Weights

DOMAIN	WEIGHT
Cluster Architecture, Installation & Configuration	25%
Workloads & Scheduling	15%
Services & Networking	20%
Storage	10%
Troubleshooting	30%



# The Curriculum

## 25% - Cluster Architecture, Installation & Configuration

- Manage role based access control (RBAC)
- Use Kubeadm to install a basic cluster
- Manage a highly-available Kubernetes cluster
- Provision underlying infrastructure to deploy a Kubernetes cluster
- Perform a version upgrade on a Kubernetes cluster using Kubeadm
- Implement etcd backup and restore

## 20% - Services & Networking

- Understand host networking configuration on the cluster nodes
- Understand connectivity between Pods
- Understand ClusterIP, NodePort, LoadBalancer service types and endpoints
- Know how to use Ingress controllers and Ingress resources
- Know how to configure and use CoreDNS
- Choose an appropriate container network interface plugin

## 30% - Troubleshooting

- Evaluate cluster and node logging
- Understand how to monitor applications
- Manage container stdout & stderr logs
- Troubleshoot application failure
- Troubleshoot cluster component failure
- Troubleshoot networking

## 15% - Workloads & Scheduling

- Understand deployments and how to perform rolling update and rollbacks
- Use ConfigMaps and Secrets to configure applications
- Know how to scale applications
- Understand the primitives used to create robust, self-healing, application deployments
- Understand how resource limits can affect Pod scheduling
- Awareness of manifest management and common templating tools

## 10% - Storage

- Understand storage classes, persistent volumes
- Understand volume mode, access modes and reclaim policies for volumes
- Understand persistent volume claims primitive
- Know how to configure applications with persistent storage



# Candidate Skills



**kubernetes**

Architecture & Concepts



`kubectl`

Running Commands



container runtime

Underlying Concepts



---

# Exam Environment

*Online and proctored exam*

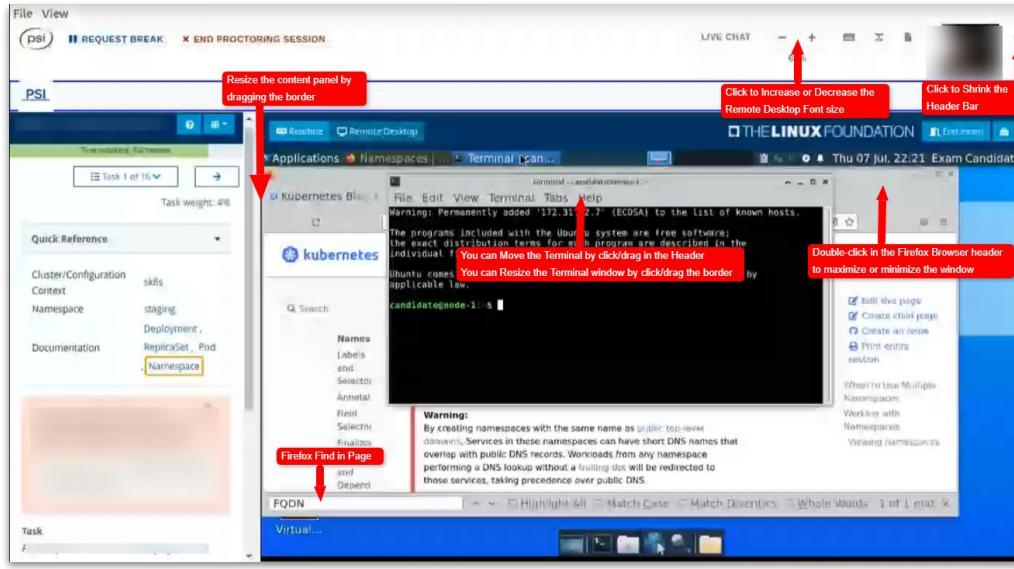


*The trinity of tooling you need to be familiar with*



# New PSI Exam Environment

*Single monitor, no more bookmarks (announcement)*



---

# Using Kubernetes Documentation

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

- Docs: <https://kubernetes.io/docs>
- 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:
```

```
...
```

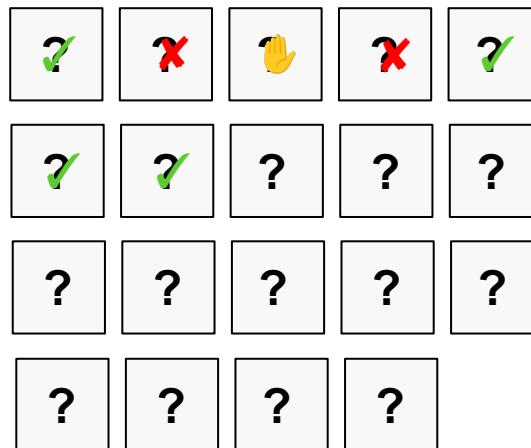
Most relevant information



---

# Time Management

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



# Configuring Auto-Completion

*Allowed during exam, preconfigured in the exam*

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

```
$ kubectl create
```



<https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/>



---

# 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 --now
```



# 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!



O'REILLY®

# Certified Kubernetes Administrator (CKA) Study Guide

In-Depth Guidance and Practice



Benjamin Muschko



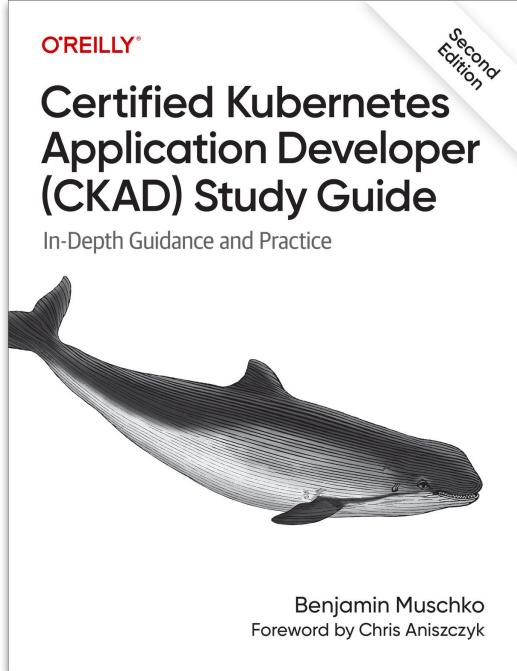
Amazon

<https://www.amazon.com/Certified-Kubernetes-Administrator-Study-Depth/dp/1098107225>



Online access on O'Reilly learning platform

<https://oreillymedia.pxf.io/c/5266844/1962804/15173>



**Amazon**

<https://www.amazon.com/Certified-Kubernetes-Application-Developer-Study-ebook/dp/B0D4X23FSY>



**Online access on O'Reilly learning platform**

<https://oreillymedia.pxf.io/c/5266844/1962801/15173>

# *CKA exam practice test*

The screenshot shows a Katacoda labs interface. On the left, a sidebar titled "CKA Prep: Setting up RBAC for a Service Account" indicates "Step 1 of 4". It contains sections for "Creating a ServiceAccount" and "Create the ServiceAccount named `api-access` in a new namespace called `appc`". Below these are "Show Solution" and "Continue" buttons. On the right, a terminal window titled "Your Interactive Learning Environment Bash Terminal" displays the following command sequence:

```
$ launch.sh
$ kubectl get namespaces to start...
Namespaces listed:
$ kubectl create namespace appc
namespace/appc created
$ kubectl create serviceaccount api-access --namespace=appc
serviceaccount/api-access created
$ kubectl get serviceaccounts --namespace=appc
NAME          SECRETS   AGE
default        1         6s
$
```

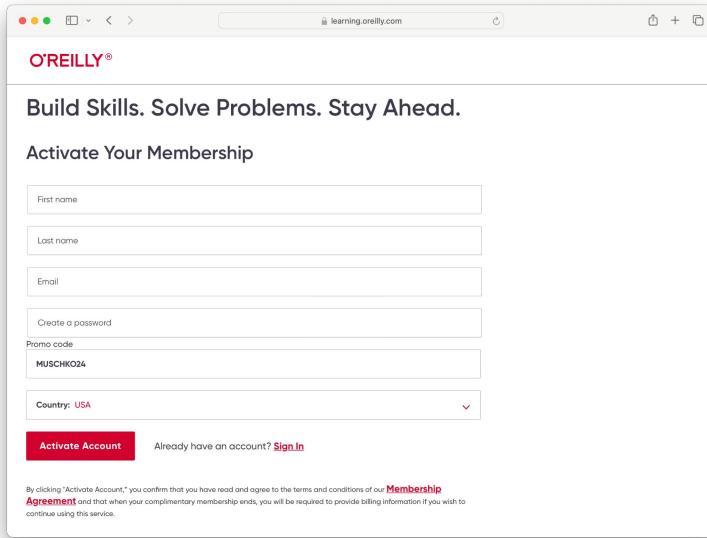


## Katacoda labs

Online access on O'Reilly learning platform:

<http://oreillymedia.pxf.io/c/5266844/1963805/15173>

# 30-Day Trial Access



A screenshot of a web browser window showing the O'Reilly membership activation page at [learning.oreilly.com](https://learning.oreilly.com). The page has a white header with the O'REILLY logo and the tagline "Build Skills. Solve Problems. Stay Ahead." Below this, a section titled "Activate Your Membership" contains several input fields: "First name", "Last name", "Email", "Create a password", "Promo code" (with the value "MUSCHKO24" entered), and "Country" (set to "USA"). At the bottom left is a red "Activate Account" button, and at the bottom right is a link "Already have an account? [Sign In](#)". A small legal note at the bottom states: "By clicking "Activate Account," you confirm that you have read and agree to the terms and conditions of our [Membership Agreement](#) and that when your complimentary membership ends, you will be required to provide billing information if you wish to continue using this service."



Activate your membership with promo code  
**MUSCHKO2024**

<https://oreillymedia.pxf.io/c/5266844/1962779/15173>

Disclosure: This is an affiliate code.



# Buying the Exam Voucher

**Cost \$395 | Online Exam**

**REGISTER FOR EXAM**

Get a 20% with code  
**ASCENT20** at checkout

Disclosure: This is an affiliate code.



---

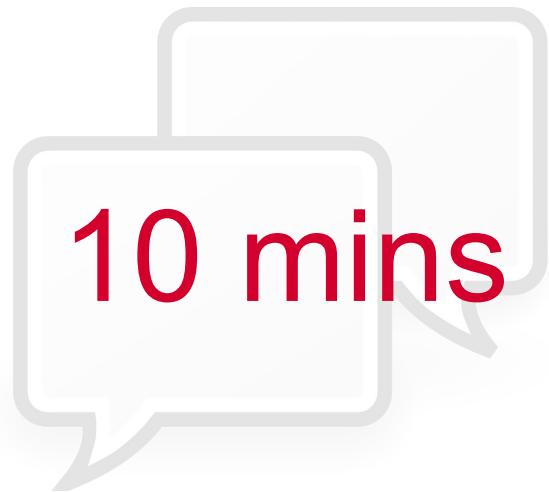
# How to Prepare

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



---

# Q & A



---

# BREAK



# Cluster Architecture, Installation & Configuration

RBAC, Kubeadm, HA, etcd Backup and Restore

# What is RBAC?

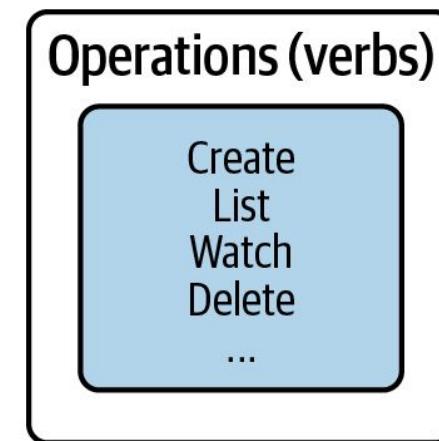
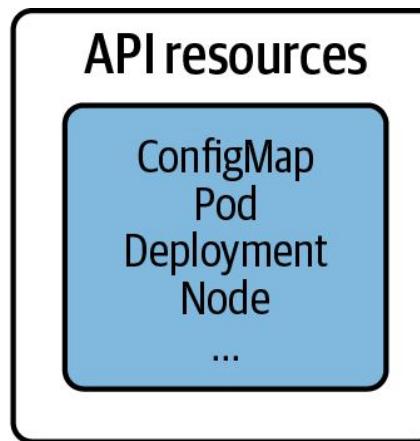
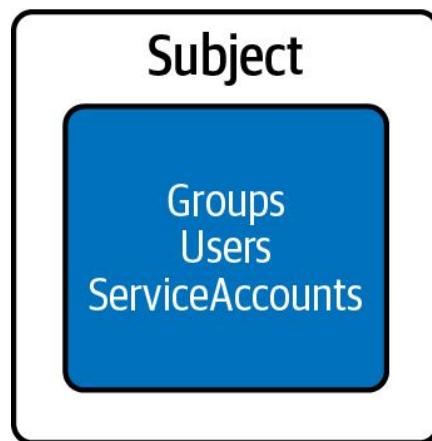
*Restricting access control for clients interacting with API server*

- 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.”



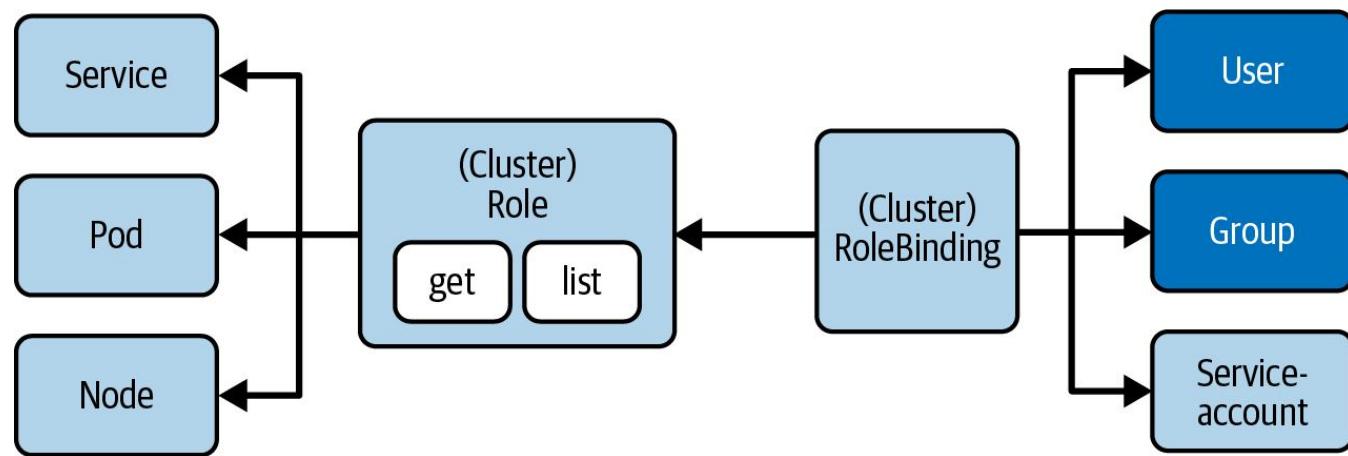
# RBAC High-Level Overview

*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: read-only
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.

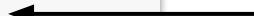


# Aggregated ClusterRoles

*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



# What is Kubeadm?

*Tool for creating and managing Kubernetes clusters*

- Needs to be installed separately from other tools like `kubectl`.
- Deals with cluster bootstrapping but not provisioning.
- Representative use cases
  - Bootstrap a control plane node.
  - Bootstrap worker nodes and join them to the cluster.
  - Upgrade a cluster to a newer version.



---

# Installing a Cluster

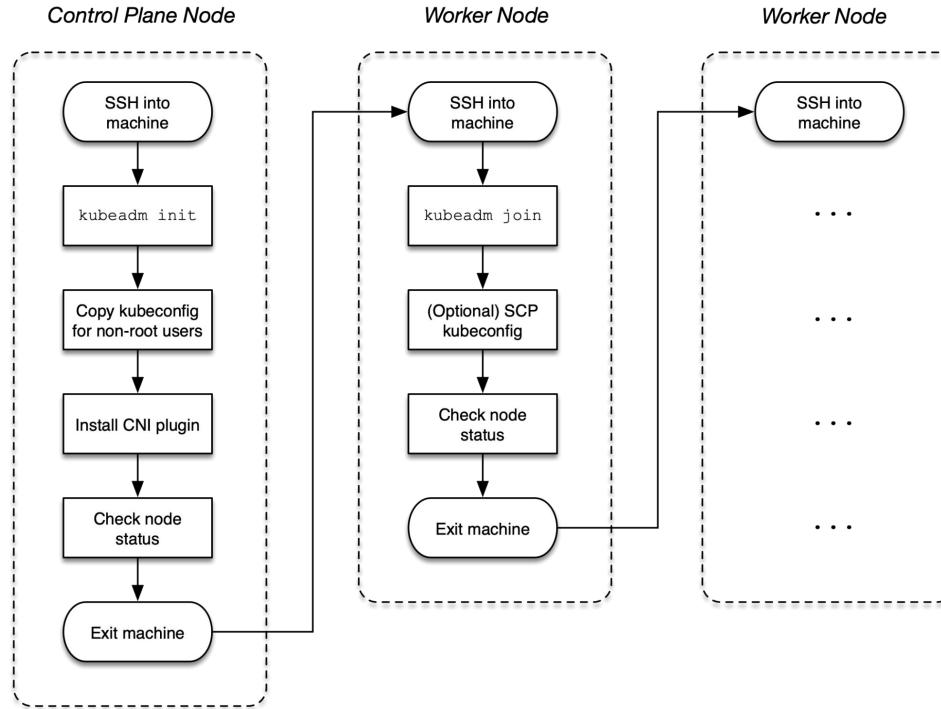
*Start with control plane, join nodes*

- Initialize the control plane on node using `kubeadm init`.
- Install a Pod network add-on.
- Join worker nodes using `kubeadm join`.

[Detailed installation instructions](#)



# Installing a Cluster



---

# DEMO

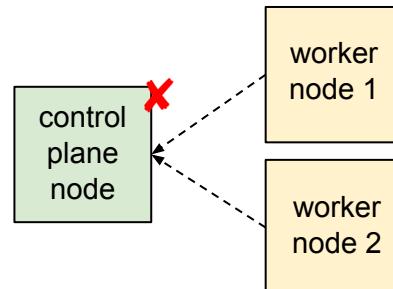
Creating a Cluster  
with Kubeadm



# Single-Control Plane Cluster

*Losing the control plane node causes issues*

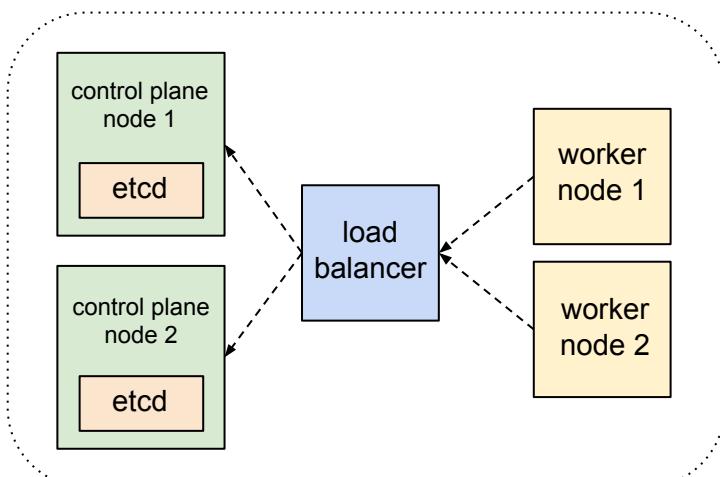
- A ReplicaSet cannot recreate failing Pod as the worker node can't talk back to scheduler on control plane node.
- Cluster cannot be accessed externally as API server is not available anymore.



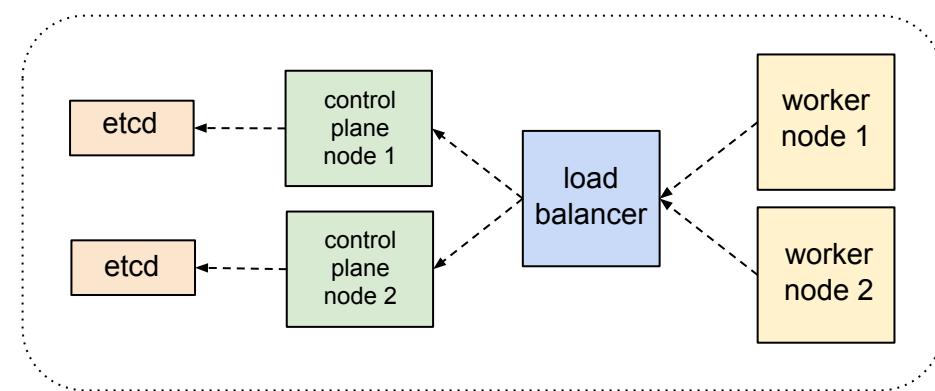
# High-Availability Cluster Setup

*Two configuration options available*

**Stacked etcd topology**



**External etcd topology**



[Detailed installation instructions](#)



---

# Upgrading a Cluster Version

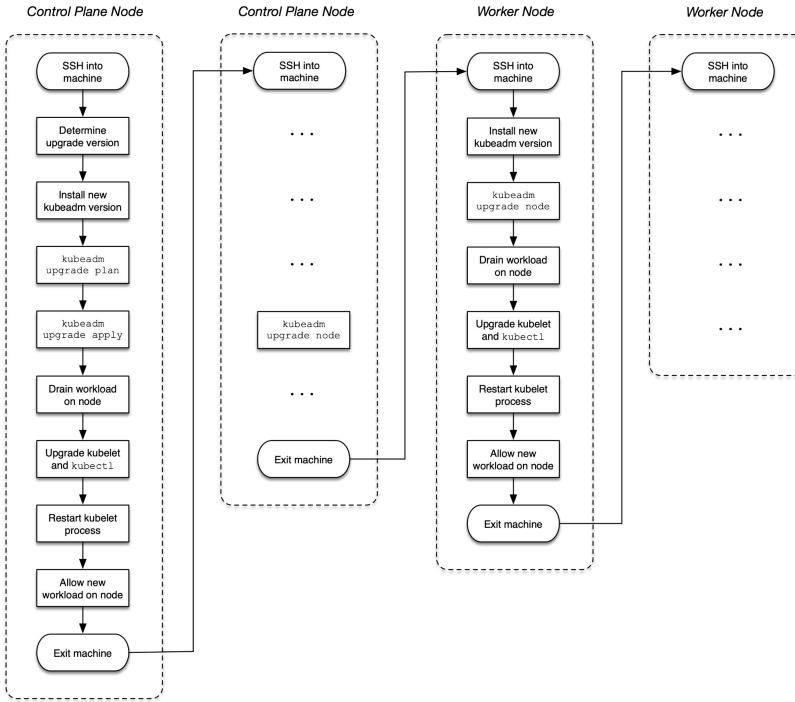
*Upgrading should be done in version increments*

- Determine which version to upgrade to.
- Upgrade control plane nodes.
- Upgrade worker nodes.
- Verify the status of the cluster.

[Detailed installation instructions](#)



# Upgrading a Cluster Version



---

# DEMO

Upgrading a  
Cluster Version with  
Kubeadm



# Backing up & Restoring etcd

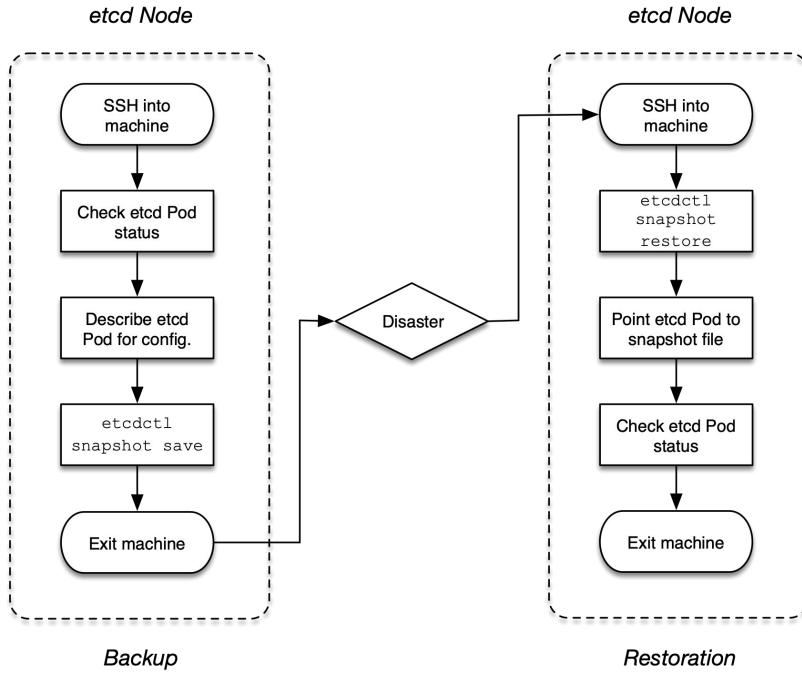
*Get etcdctl utility if it's not already present*

- Create backup with `etcdctl snapshot save` command. The options `--cert`, `--cacert` and `--key` are mandatory.
- Restore backup with `etcdctl snapshot restore` command. The option `--data-dir` is mandatory. Modify the `volumes.hostPath.path` in `/etc/kubernetes/manifests/etcd.yaml` to point to directory.

[Detailed installation instructions](#)



# Backing up & Restoring etcd



---

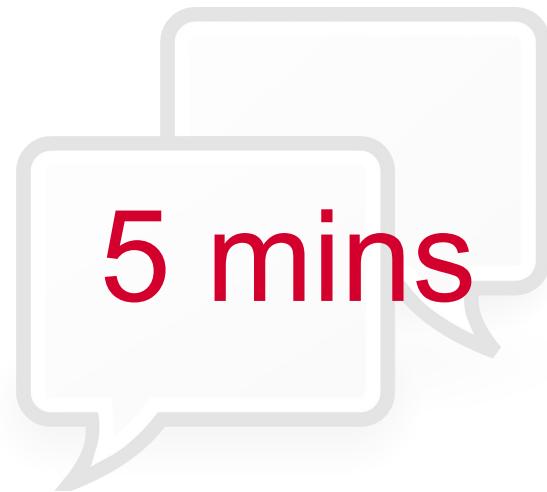
# DEMO

Backing Up and  
Restoring etcd



---

# Q & A



---

# BREAK



# Workloads & Scheduling

Deployments, ConfigMaps & Secrets, Health  
Probes, Pod Resource Limits, Node Affinity, Taints &  
Tolerations

---

# What is a Deployment?

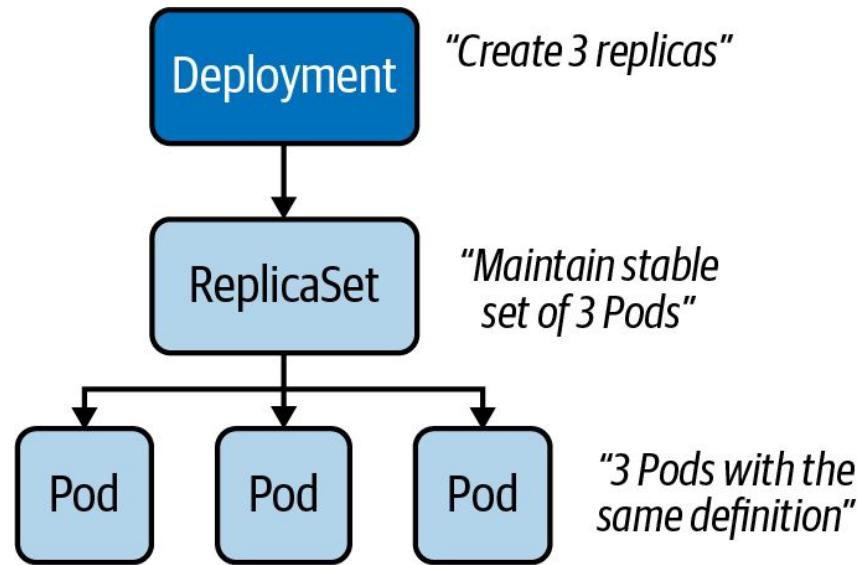
*Scaling and replication features for a set of Pods*

- 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)
```

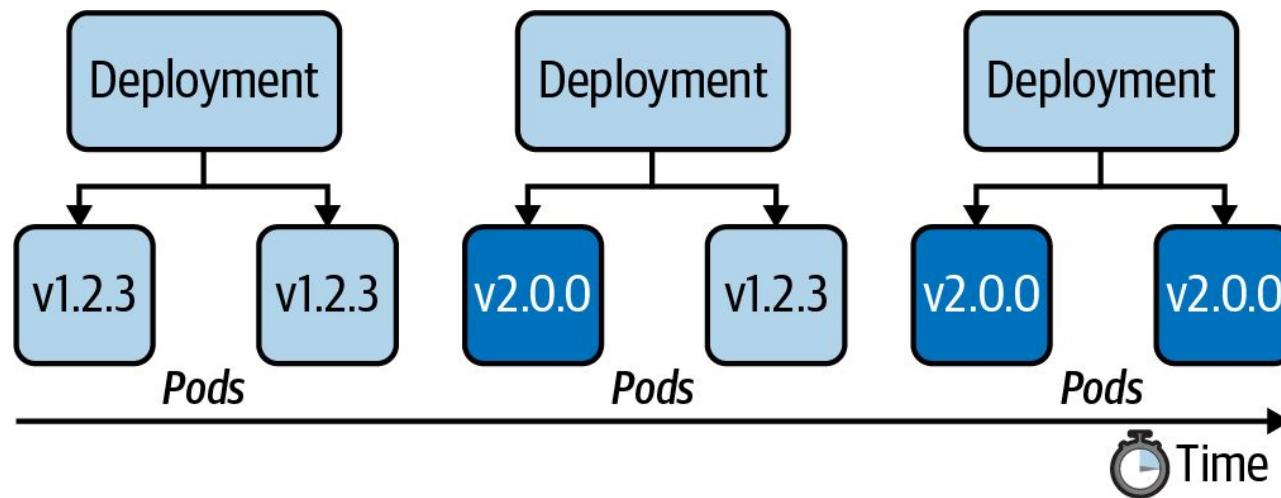
```
$ kubectl describe replicaset.apps/my-deploy-8448c488b5
Controlled By:  Deployment/my-deploy
```

```
$ 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
```

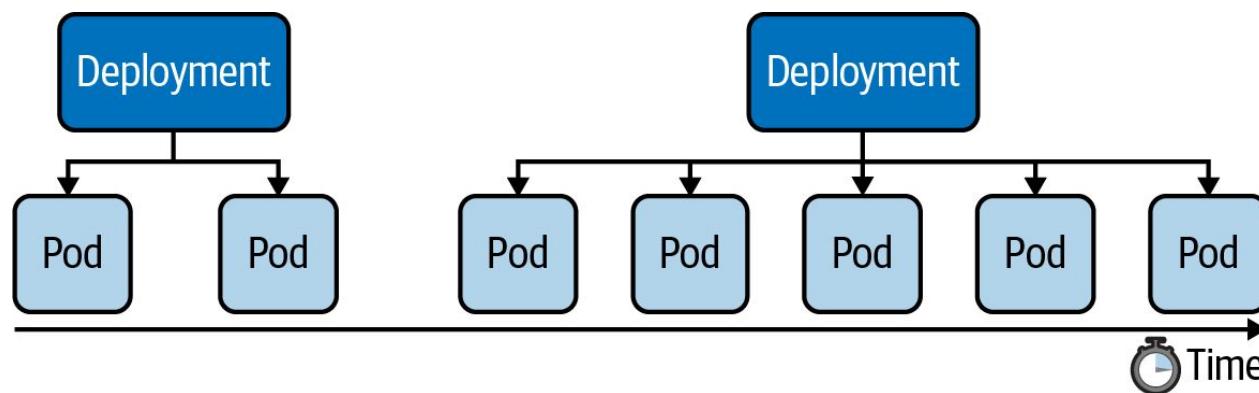
```
spec:  
  replicas: 1
```

- *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*

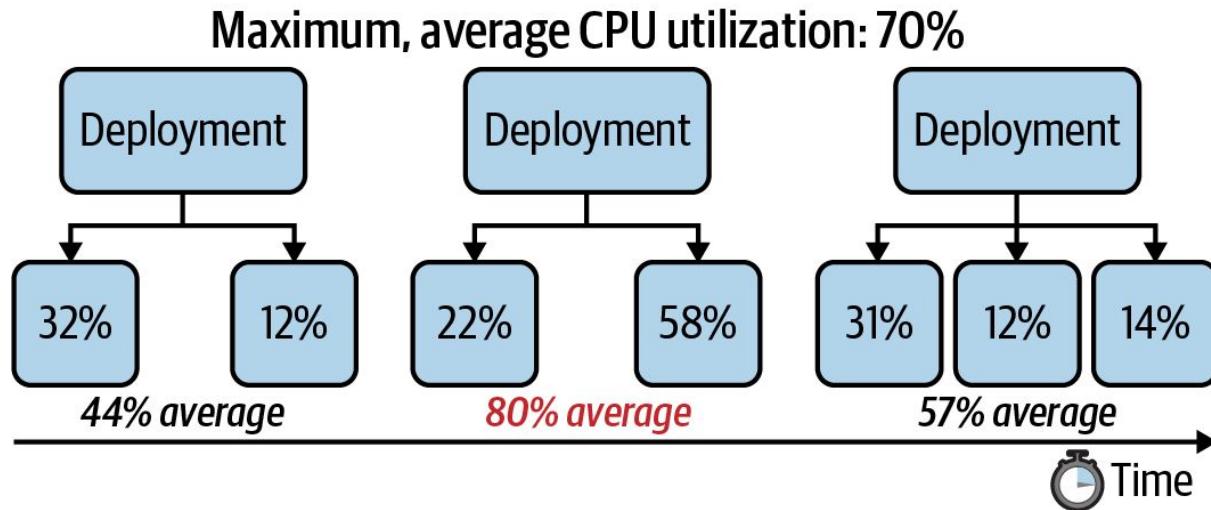
```
$ kubectl scale deployment my-deploy --replicas=5
deployment.apps/my-deploy scaled

$ 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



# Defining Configuration Data

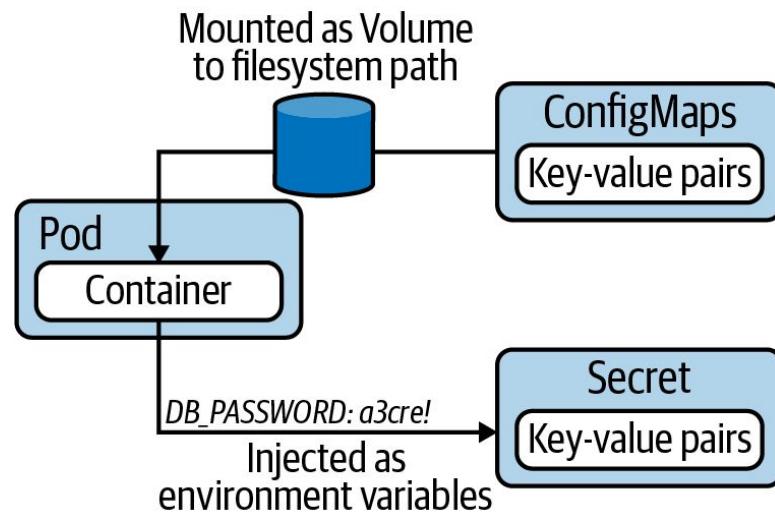
*Control runtime behavior with centralized information*

- 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
```

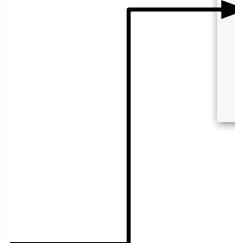


# Consuming as Volume

*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

Creating and  
Mounting a Secret



---

# Understanding Health Probing

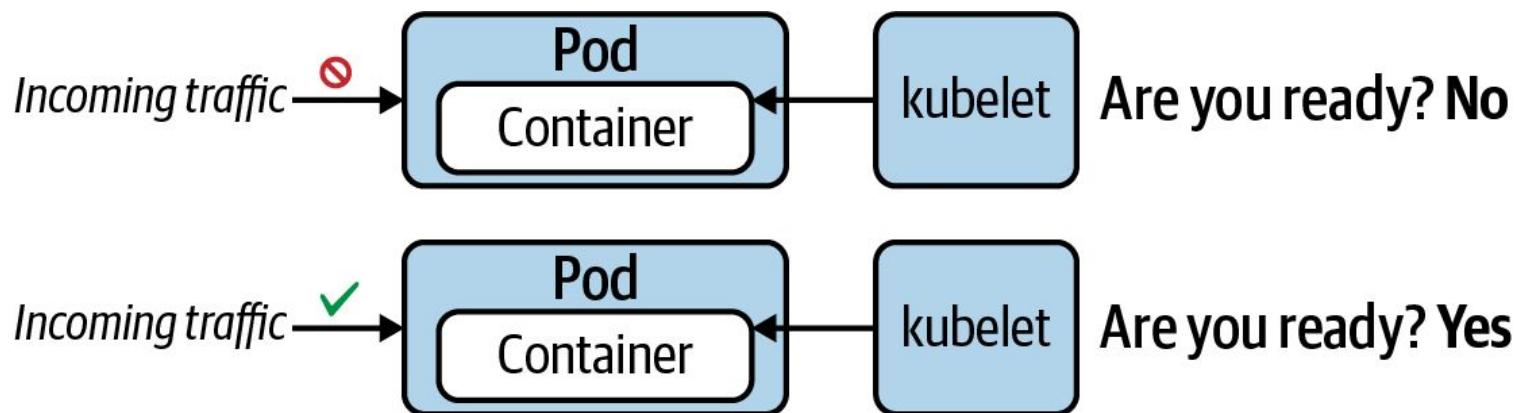
*Runtime issues can arise, but you check for them*

- 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.



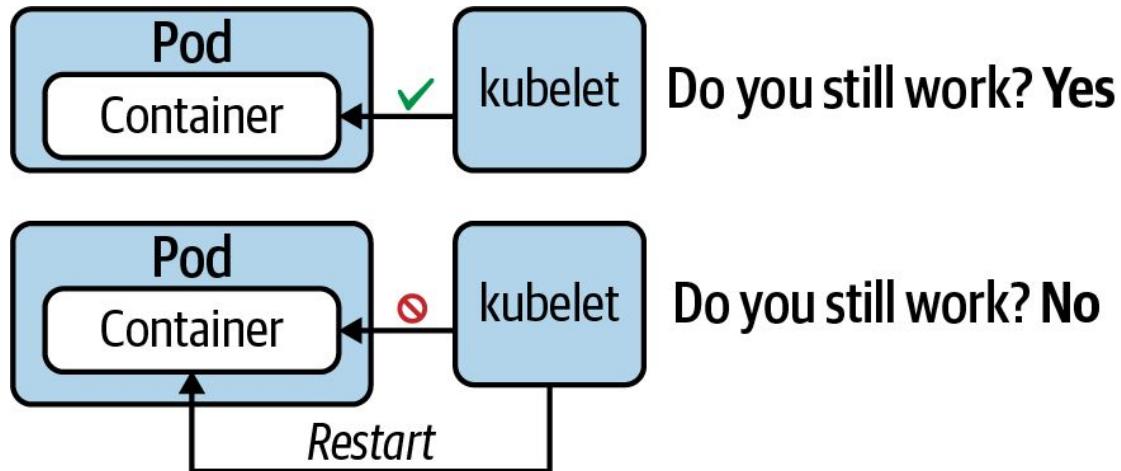
# Readiness Probe

*“Is application ready to serve requests?”*



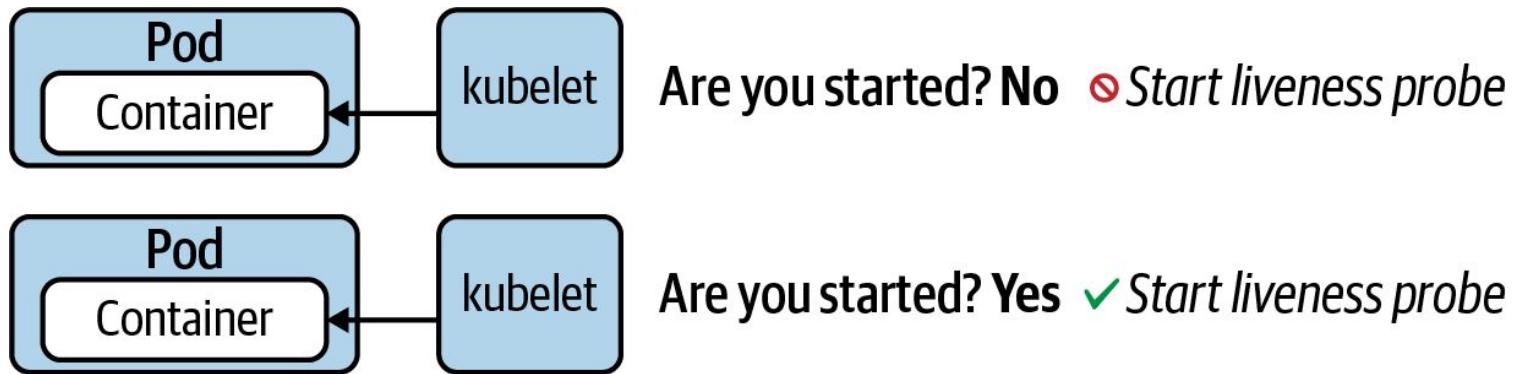
# Liveness Probe

*“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

Configuring Health  
Probes for a Pod



# 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

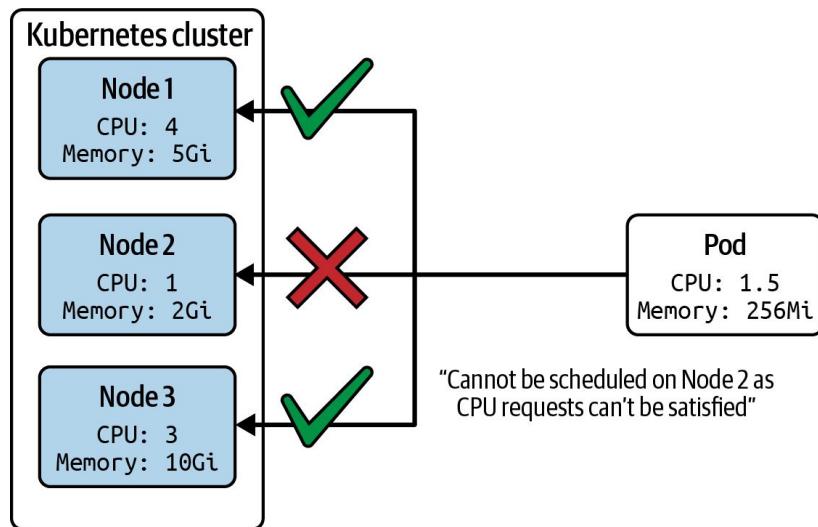
*Definition and runtime effects*

- 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



---

# Node Affinity & Taints/Tolerations

*Concepts with different purposes, but can go hand in hand*

- **Node Affinity:** Attract Pods to a node as soft or hard requirement.
- **Taint:** Allow a node to repel a set of Pods.
- **Tolerations:** Applied to Pods to allow scheduling them to nodes with a specific taint.



# Kubernetes Scheduler

*kube-scheduler is the default scheduler for Kubernetes*

```
$ kubectl get pods -n kube-system
NAME                  READY   STATUS    RESTARTS   AGE
kube-scheduler-minikube 1/1     Running   2          76d
...
```

[Detailed information on node selection](#)



# What Node Does Pod Run On?

*Once Pod is scheduled, the node is assigned automatically*

```
$ kubectl get nodes
NAME        STATUS    ROLES      AGE     VERSION
minikube   Ready     master    204d    v1.19.2

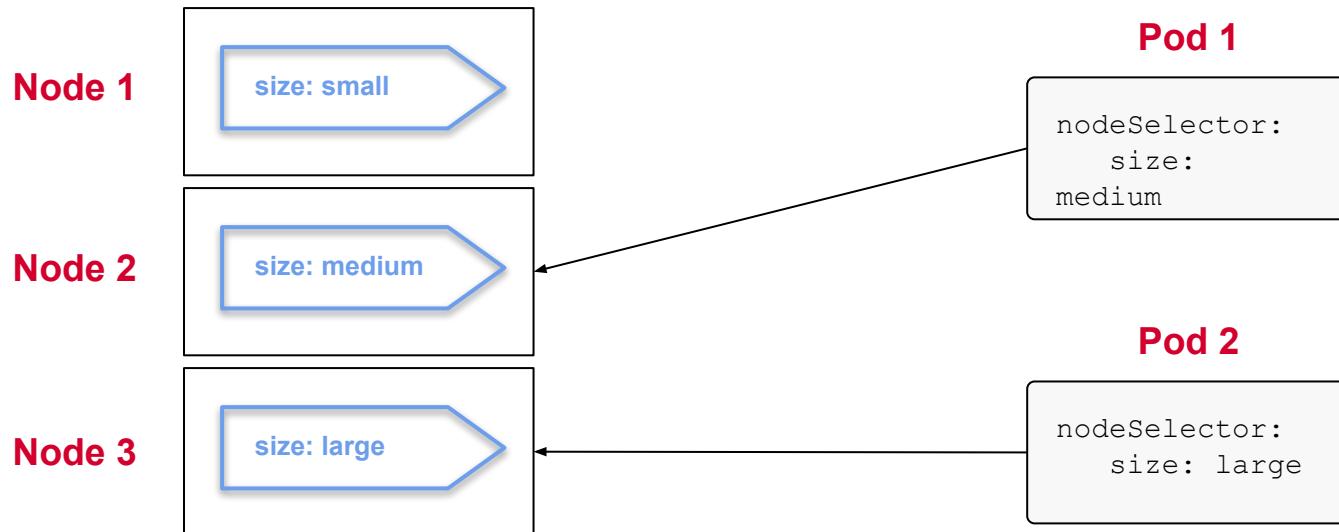
$ kubectl get pods -o=wide
NAME    READY    STATUS    RESTARTS   AGE     IP          NODE      ...
app     1/1     Running   0          22h    10.0.0.102  minikube  ...

$ kubectl get pod app -o yaml | grep nodeName:
nodeName: minikube
```



# Node Selection Constraint

*Define a label selector in Pod's spec that matches node label*



# Node Selection Constraint

*Add labels to nodes*

```
$ kubectl label nodes minikube size=medium
node/minikube labeled

$ kubectl get nodes --show-labels
NAME      STATUS   ROLES    AGE     VERSION   LABELS
...
minikube-m02   Ready    master   204d    v1.19.2   ...size=medium
minikube-m03   Ready    master   204d    v1.19.2   ...
minikube-m04   Ready    master   204d    v1.19.2   ...
```



# Node Selection Constraint

*Define the `nodeSelector` attribute upon Pod creation*

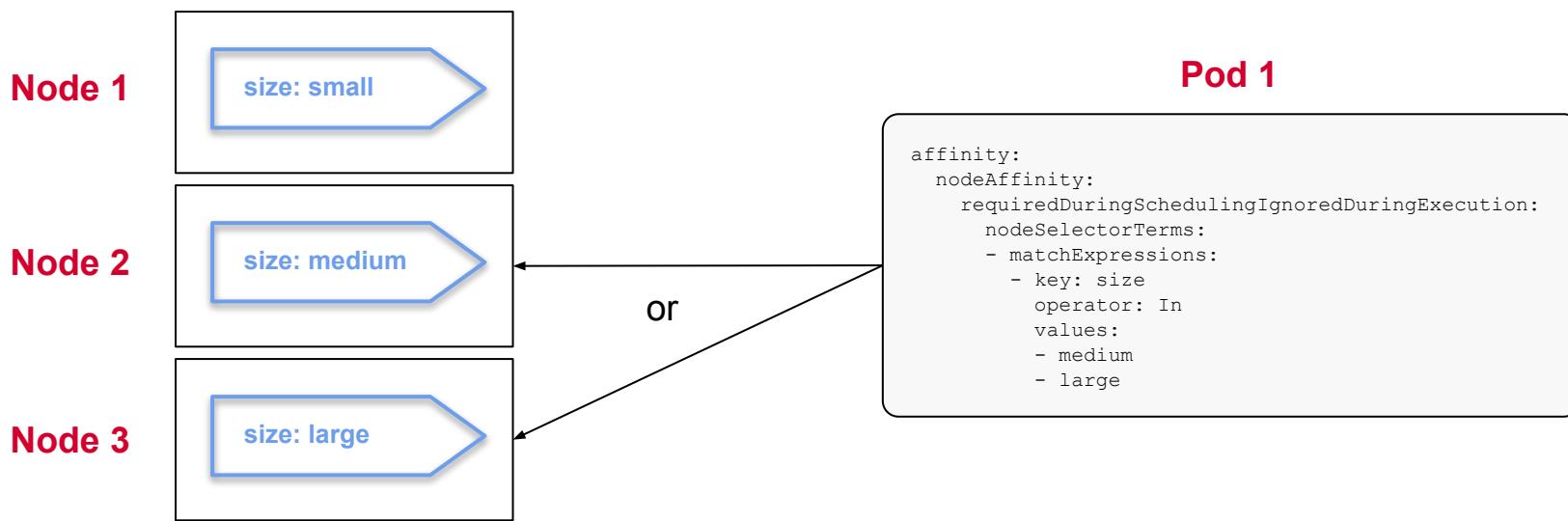
```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - image: nginx
      name: nginx
  restartPolicy: Never
  nodeSelector:
    size: medium
```

*Can't have multiple keys with the same value as the underlying data structure is a map*



# Node Affinity

*Similar to nodeSelector but more flexible and powerful*



# Setting a Pod's Node Affinity

*Requires a lot of configuration in various shapes and forms*

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - image: nginx
      name: nginx
  restartPolicy: Never
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: size
                operator: In
                values:
                  - medium
```

*Available operators:*  
*In, NotIn, Exists,*  
*DoesNotExist, Gt, Lt*



# Node Affinity Types

*Currently two types, potentially more in the future*

Type	Description
requiredDuringSchedulingIgnoredDuringExecution	Rules that must be met for a Pod to be scheduled onto a node.
preferredDuringSchedulingIgnoredDuringExecution	Rules that specify preferences that the scheduler will try to enforce but will not guarantee.

*\*IgnoredDuringExecution means that changes to the affinity of a running Pod does not have an effect*



---

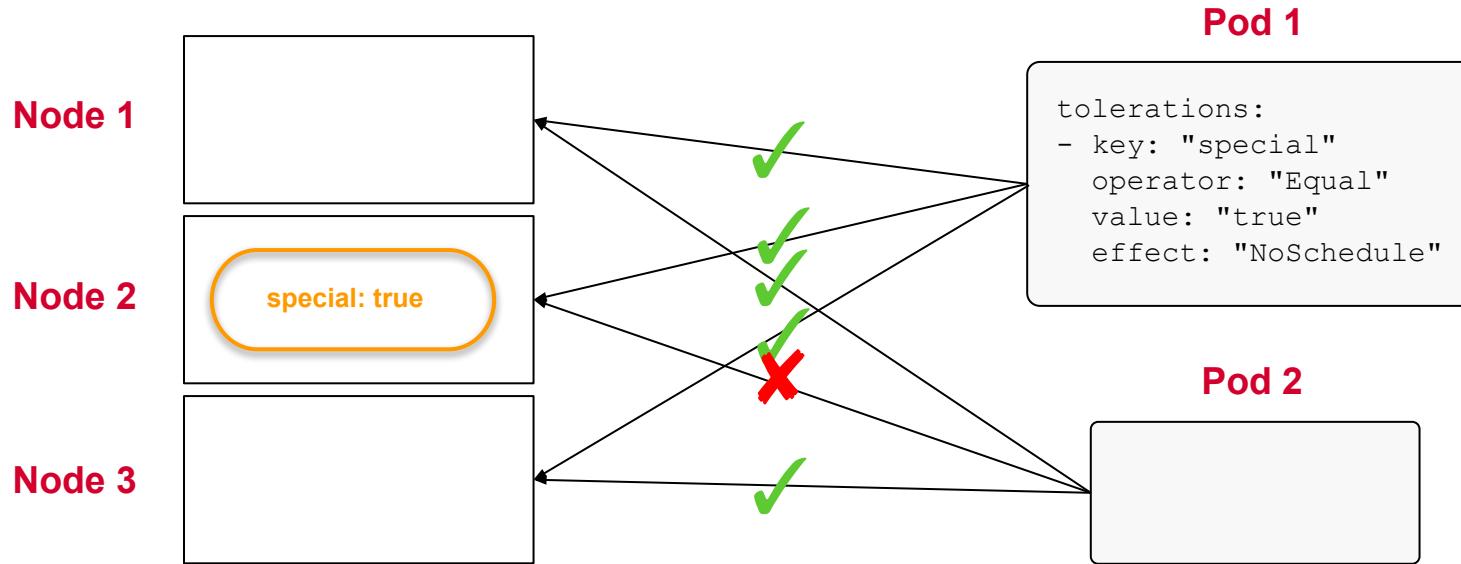
# EXERCISE

Scheduling a Pod  
on Specific Nodes



# Taints and Tolerations

*A Pod that doesn't have specific toleration is repelled*



# Setting a Node Taint

*Add taint to nodes*

```
$ kubectl taint nodes minikube-m02 special=true:NoSchedule
node/minikube-m02 tainted
$ kubectl get nodes minikube-m02 -o yaml | grep -C 3 taints:
...
spec:
  taints:
  - effect: NoSchedule
    key: special
    value: "true"
```

*key=value:effect*



# Taint Effects

*Needs to be provided to node and Pod*

Effect	Description
NoSchedule	Unless a Pod has matching toleration, it won't be scheduled on the node.
PreferNoSchedule	Try not to place a Pod that does not tolerate the taint on the node, but it is not required.
NoExecute	Evict Pod from node if already running on it. No future scheduling on the node.



# Setting a Pod's Toleration

*Requires a lot of configuration in various shapes and forms*

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - image: nginx
    name: nginx
  restartPolicy: Never
  tolerations:
  - key: "special"
    operator: "Equal"
    value: "true"
    effect: "NoSchedule"
```

*Available operators:*  
*Equal, Exists*



---

# EXERCISE

Configuring a Node  
to Only Accept  
Specific Pods



---

# Q & A



---

# BREAK



# Services & Networking

Inter-Pod Communication, Service Types, Ingress,  
CoreDNS, CNI plugins

# What is a Service?

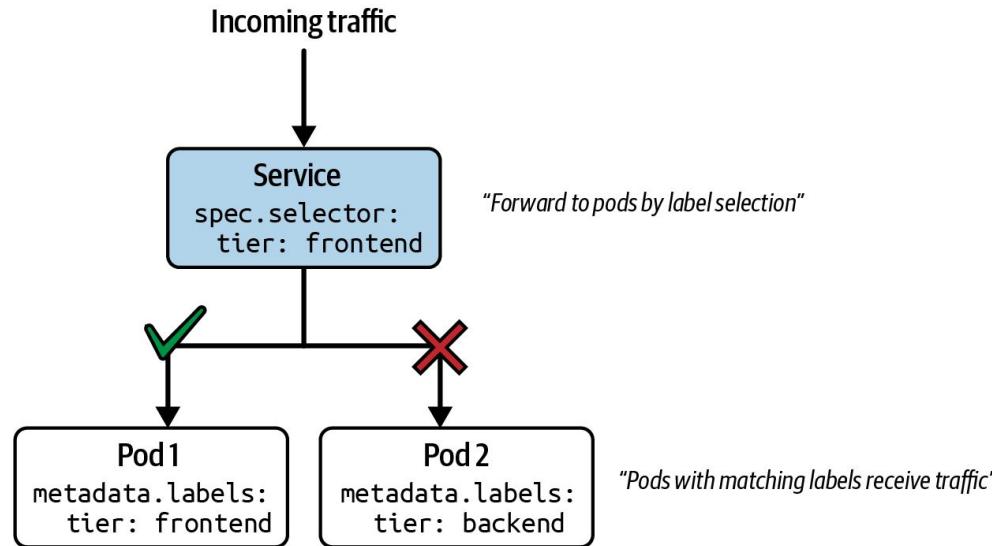
*Discoverable networking capability for Pods*

- 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

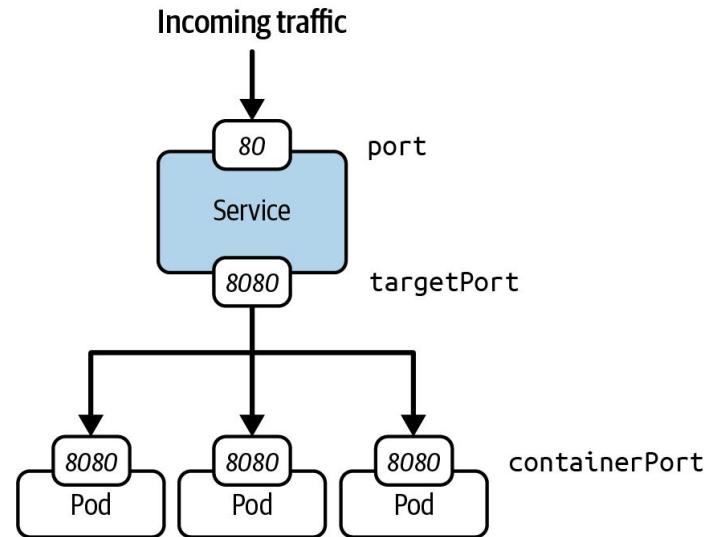
*Defines containers and their images running in them*

```
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
```



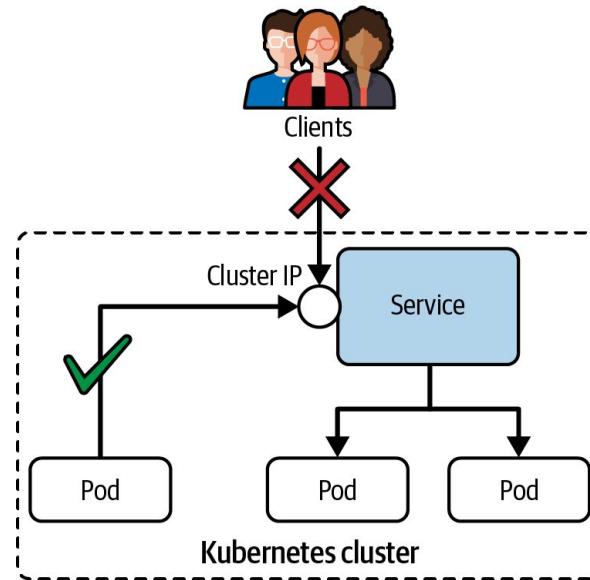
# Port Mapping

*“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

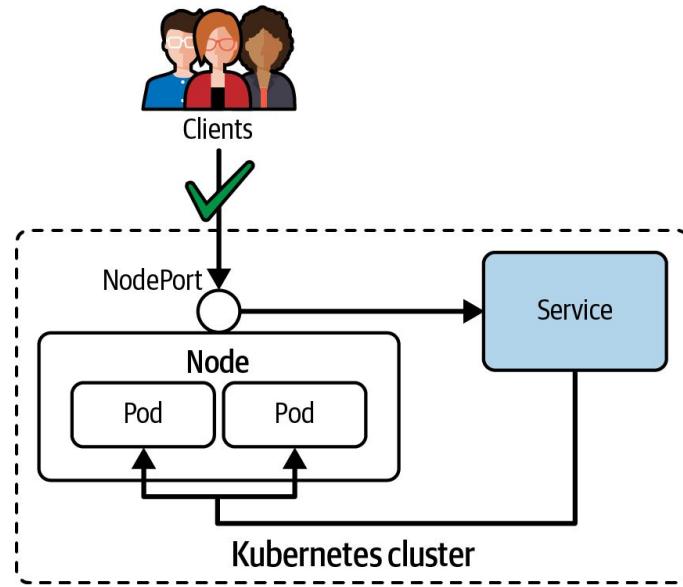
NAME            TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/echoserver  ClusterIP  10.96.254.0 <none>        5005/TCP  8s

$ 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

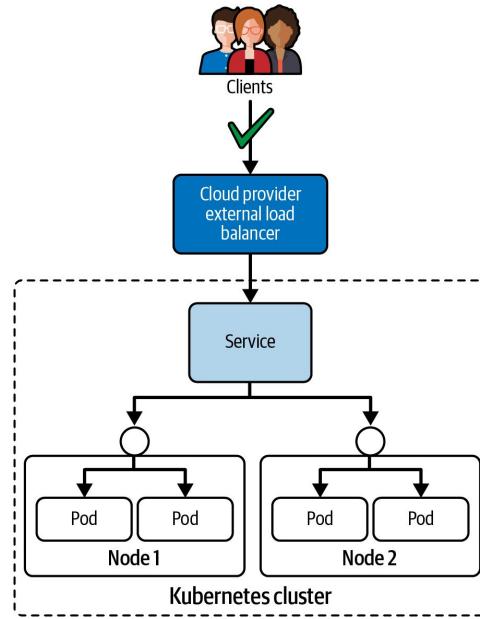
$ kubectl get nodes -o<
  jsonpath='{ $.items[*].status.addresses[?(@.type=="InternalIP")].address }'
192.168.64.15

$ 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.
...
```



---

# EXERCISE

Routing traffic to  
Pods from Inside  
and Outside of a  
Cluster



# What is an Ingress?

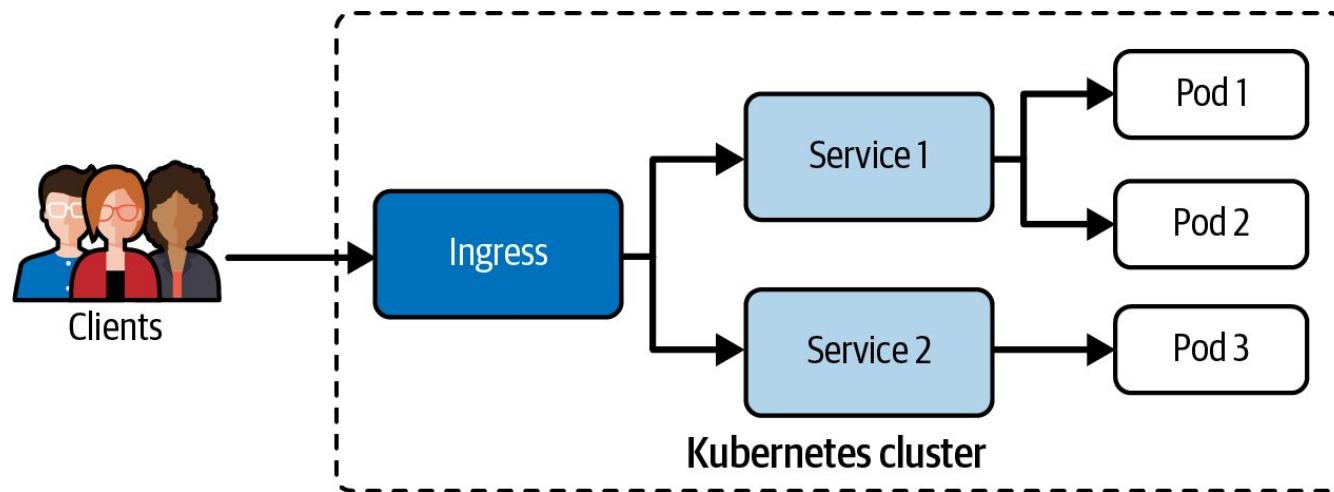
*Route HTTP(S) traffic from the outside of cluster to Service*

- 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.
...
...
```



---

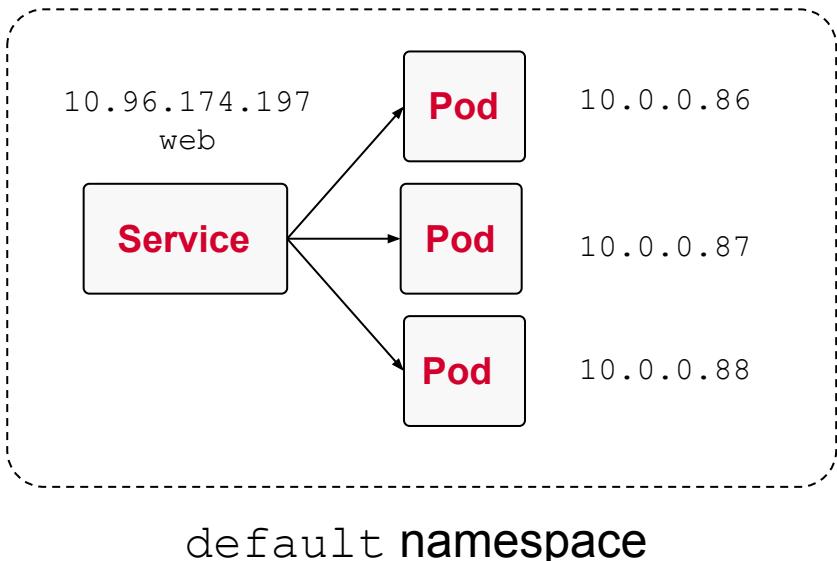
# EXERCISE

Defining and Using  
an Ingress



# DNS for Services

*Kubernetes DNS service creates record for Service*



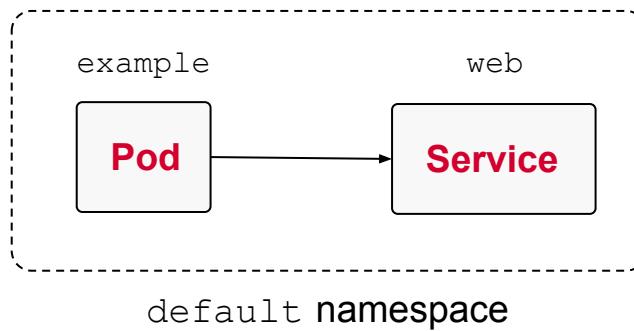
**Kubernetes DNS Service**

Hostname	IP Address
web	10.96.174.197



# Resolving a Service by DNS

*Resolve by hostname within the same namespace*

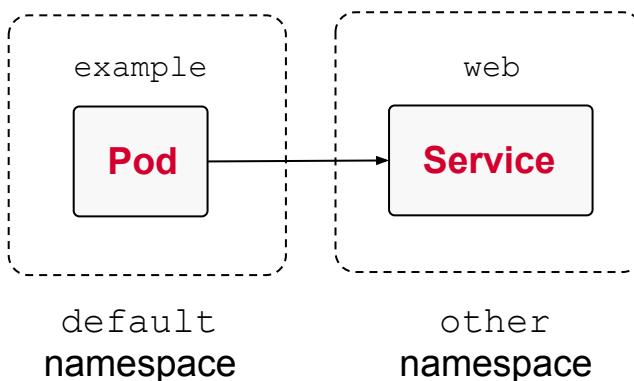


```
$ curl http://web  
Hello World
```



# Resolving a Service by DNS

*Resolve by namespace, type, root domain from another namespace*



```
$ curl http://web.other
```

Hello World

```
$ curl http://web.other.svc
```

Hello World

```
$ curl http://web.other.svc.cluster.local
```

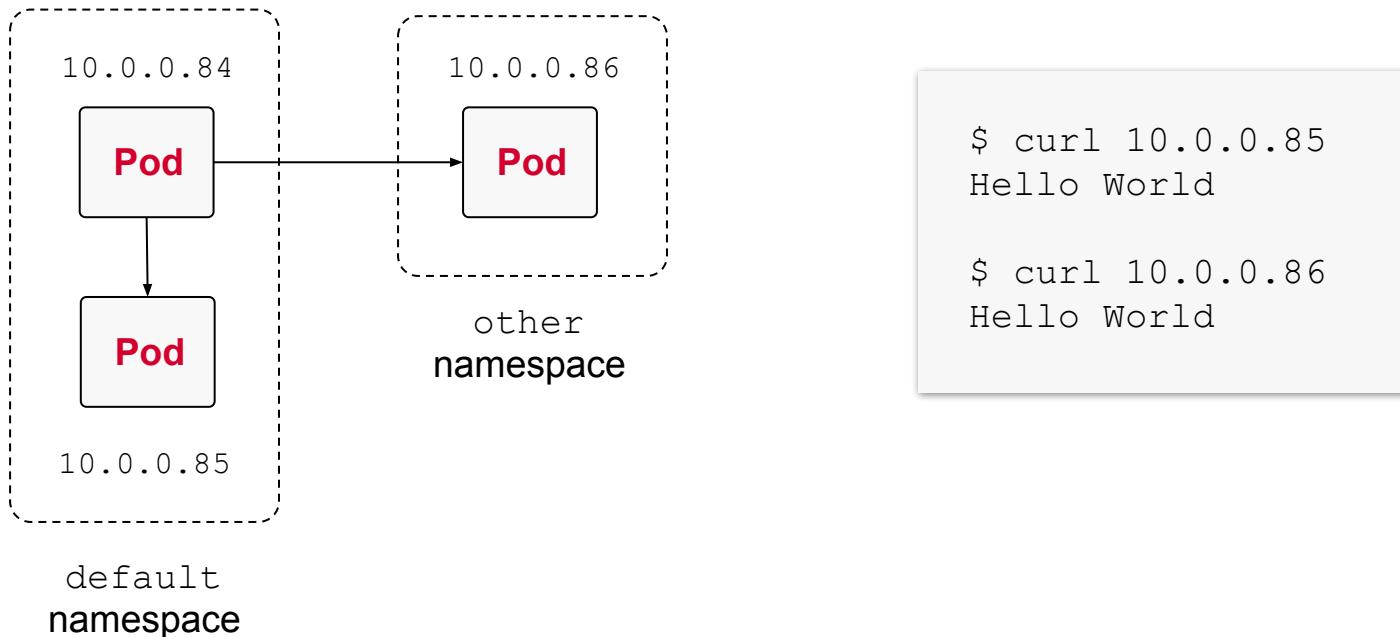
Hello World

Root Domain



# Resolving a Pod by DNS

*DNS records are not created by default, resolve by IP address*



# Object Representation CoreDNS

*Recommended Kubernetes DNS server implementation*

```
$ kubectl get pods -n kube-system
NAME                  READY   STATUS    RESTARTS   AGE
coredns-f9fd979d6-skk6w   1/1     Running   2          67d

$ kubectl get services -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   195d
```



# Configuring CoreDNS

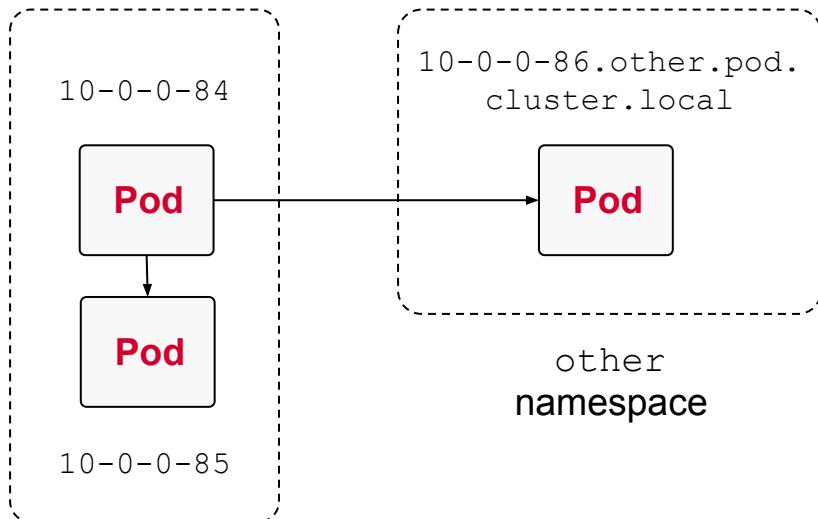
*Sets root domain and enables DNS for Pods*

```
$ kubectl describe configmap coredns -n kube-system
Data
=====
Corefile:
-----
.:53 {
    ...
    kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        fallthrough in-addr.arpa ip6.arpa
        ttl 30
    }
    ...
}
```



# Resolving a Pod by DNS

*DNS records are not created by default, resolve by IP address*



default  
namespace

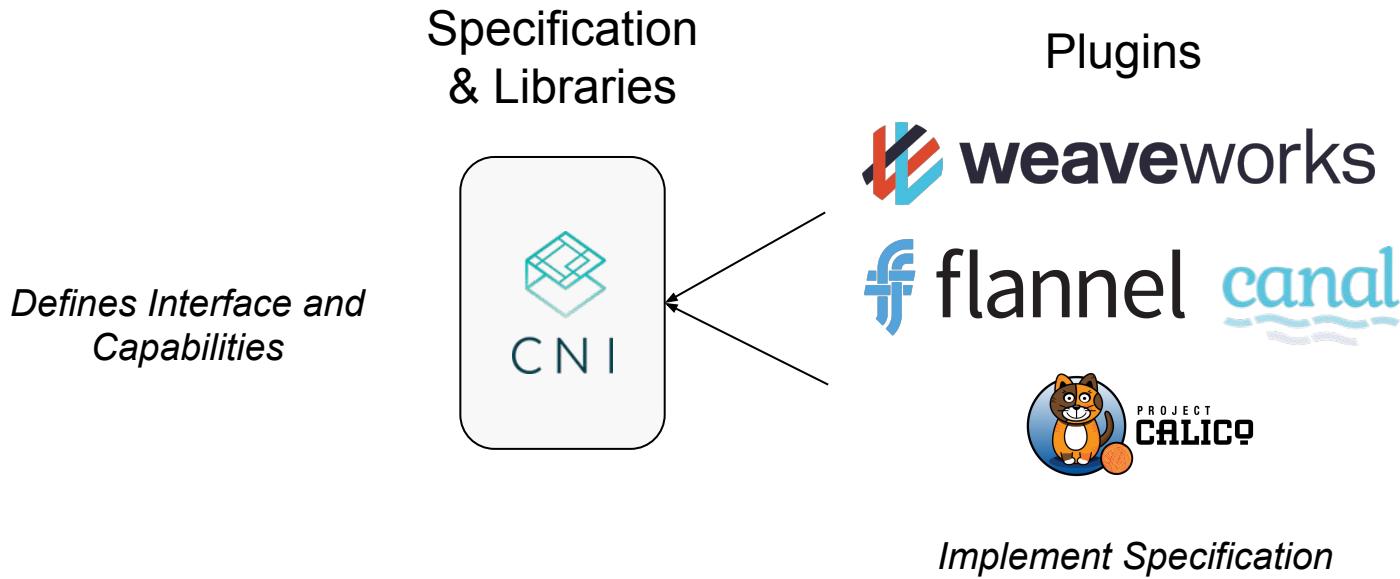
```
$ curl 10-0-0-85
Hello World

$ curl
10-0-0-86.other.pod.cluster
.local
Hello World
```



# Understanding CNI

*Kubernetes uses CNI for Pod networking*



# Choosing a CNI Plugin

*Direct installation instructions only available for Weave Net*

**List of plugins:**

<https://kubernetes.io/docs/concepts/cluster-administration/addons/>

**Installation instructions for Weave Net:**

<https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/#steps-for-the-first-control-plane-node>



---

# Q & A



---

# BREAK

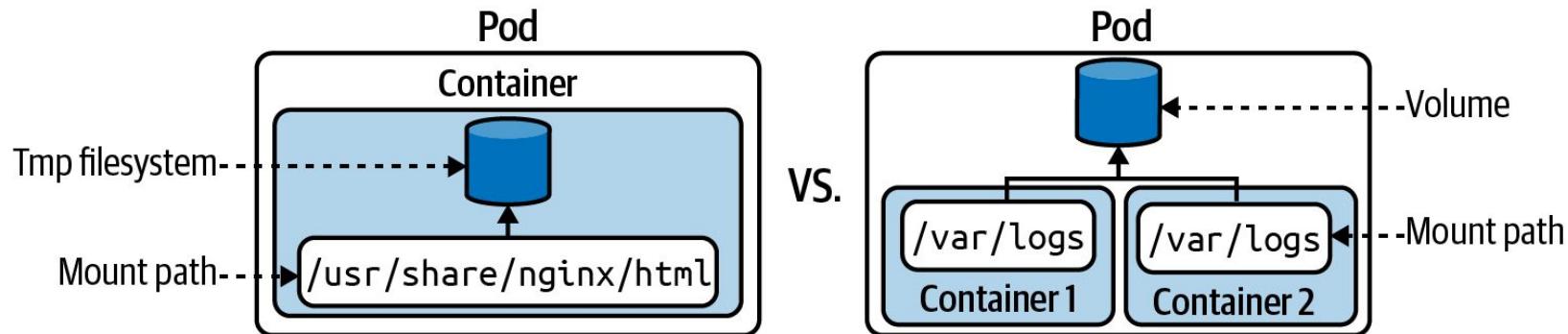


# Storage

Volumes, Volume Configuration Options,  
PersistentVolumes with Static & Dynamic Binding

# 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 or  
Dynamic Binding



# Troubleshooting

Cluster/Node Logging, Monitoring Applications,  
Identifying and Fixing Application, Cluster, and  
Networking Issues

---

# 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

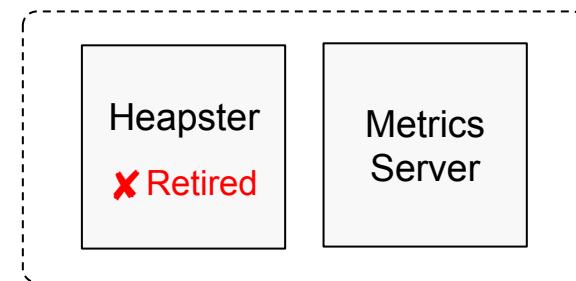
*Relevant to the exam: metrics server*

## Commercial Products



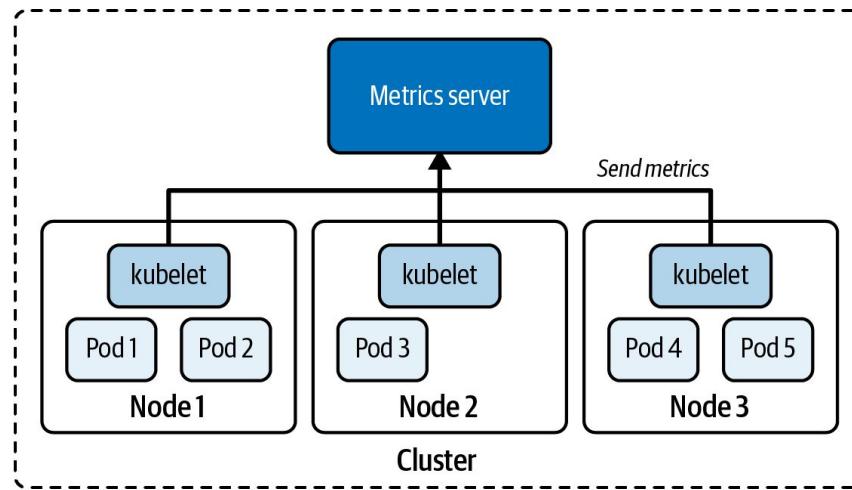
VS.

## Free Solutions



# 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
```



# Accessing Container Logs

*Simply use the `kubectl logs` command*

```
$ 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
```

*Use the command line option `-f`, `--follow` to stream the logs*



# Accessing Container Logs

*Specify container name for multi-container Pods*

```
$ kubectl logs hazelcast -c app
...
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
```

*Use the command line option `-c` or `--container`*



---

# Q & A



---

# BREAK



---

# Official Troubleshooting Docs

*Detailed advice and help during exam*

- Troubleshooting applications
- Troubleshooting cluster



# 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



# 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



# 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 an  
Issue for an  
Application



# Troubleshooting Control Plane

*Check the status of the cluster nodes first - are they running?*

```
$ kubectl get nodes
NAME        STATUS    ROLES      AGE
VERSION
master      Ready     master     198d
v1.22.3
worker-1    Ready     <none>    198d
v1.22.3
worker-2    Ready     <none>    198d
v1.22.3
worker-3    Ready     <none>    198d
v1.22.3
```

✓ Healthy status

*Use the `kubectl cluster-info dump` command for details*



# Troubleshooting Control Plane

*Check the status of control plane Pods - do they indicate issues?*

\$ kubectl get pods -n kube-system					
NAME	READY	STATUS	RESTARTS	AGE	
kube-apiserver-minikube	1/1	Running	49	70d	
kube-controller-manager-minikube	1/1	CashLoopBackoff	2	70d	
kube-proxy-mpgd9	1/1	Running	2	70d	
kube-scheduler-minikube	1/1	Running	2	70d	

✗ Failing controller manager Pod



# Troubleshooting Control Plane

*Check the logs of API server Pod*

```
$ kubectl logs kube-apiserver-minikube -n kube-system
E1231 21:06:22.978390      1 controller.go:116] loading OpenAPI
spec for "v1beta1.metrics.k8s.io" failed with: OpenAPI spec does not
exist
I1231 21:06:22.978460      1 controller.go:129] OpenAPI
AggregationController: action for item v1beta1.metrics.k8s.io: Rate ↴
Limited Requeue.
I1231 21:06:49.825964      1 client.go:360] parsed scheme: ↴
"passthrough"
```

✓ No error messages



---

# EXERCISE

Troubleshooting an  
Issue with the  
Control Plane



# Troubleshooting Worker Nodes

*Check the status of the worker nodes - are they ready?*

```
$ kubectl get nodes
NAME        STATUS    ROLES   AGE     VERSION
master      Ready     master   198d    v1.22.3
worker-1    Ready     <none>  198d    v1.22.3
worker-2    NotReady  <none>  198d    v1.22.3
worker-3    Ready     <none>  198d    v1.22.3
```

✗ Issue with the node `worker-2`



# Troubleshooting Worker Nodes

*Check condition flags of failing worker node*

```
$ kubectl describe node worker-1
...
Conditions:
  Type        Status  LastHeartbeatTime                 LastTransitionTime               Reason                               Message
  ----        -----  ----涯---                   ----涯---                   ----                   -----
  NetworkUnavailable  False   Fri, 25 Dec 2020 10:33:34 -0700   Fri, 25 Dec 2020 10:33:34 -0700   CiliumIsUp
  MemoryPressure     False   Thu, 31 Dec 2020 14:39:42 -0700   Thu, 31 Dec 2020 06:31:36 -0700   KubeletHasSufficientMemory
  DiskPressure       False   Thu, 31 Dec 2020 14:39:42 -0700   Thu, 31 Dec 2020 06:31:36 -0700   KubeletHasNoDiskPressure
  PIDPressure        False   Thu, 31 Dec 2020 14:39:42 -0700   Thu, 31 Dec 2020 06:31:36 -0700   KubeletHasSufficientPID
  Ready              True    Thu, 31 Dec 2020 14:39:42 -0700   Thu, 31 Dec 2020 06:31:36 -0700   KubeletReady
  ...

```

✓ Conditions are healthy

*A node with condition status Unknown may have crashed*



# Troubleshooting Worker Nodes

*Based on conditions check CPU, memory, processes, disk space*

```
$ top
Processes: 568 total, 2 running, 566 sleeping, 2382 threads
15:30:40
Load Avg: 1.96, 1.80, 1.68 CPU usage: 2.49% user, 1.83% sys,
95.66% idle
SharedLibs: 706M resident, 108M data, 196M linkedit.
MemRegions: 192925 total, 6808M resident, 312M private, 5106M
shared. PhysMem: 33G used (4319M wired), 31G unused.
VM: 3461G vsize, 2315M framework vsize, 0(0) swapins, 0(0)
swapouts.
Networks: packets: 6818487/8719M in, 2169040/361M out. Disks:
1305228/17G read, 1354811/32G written.
```

✓ Sufficient memory

```
$ df -h
Filesystem      Size   Used  Avail Capacity iused
ifree %iused Mounted on
/dev/disk1s1s1  1.8Ti  14Gi  1.6Ti    1%  567557
19538461203   0%   /
devfs          187Ki  187Ki  0Bi   100%   648
0 100%  /dev
/dev/disk1s5   1.8Ti  20Ki  1.6Ti    1%     0
19539028760   0%   /System/Volumes/VM
/dev/disk1s3   1.8Ti  282Mi  1.6Ti    1%   799
19539027961   0%   /System/Volumes/Preboot
/dev/disk1s6   1.8Ti  520Ki  1.6Ti    1%   16
19539028744   0%   /System/Volumes/Update
/dev/disk1s2   1.8Ti  172Gi  1.6Ti   10%  1071918
19537956842   0%   /System/Volumes/Data
map auto_home  0Bi   0Bi   0Bi   100%   0
0 100%  /System/Volumes/Data/home
```

✓ Available disk space



# Troubleshooting Worker Nodes

## *Check Kubelet status*

```
$ systemctl status kubelet.service
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset: enabled)
   Drop-In: /etc/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: active (running) since Thu 2021-01-21 22:59:54 UTC; 22min ago
       Docs: https://kubernetes.io/docs/home/
   Main PID: 6171 (kubelet)
     Tasks: 16 (limit: 1151)
    CGroup: /system.slice/kubelet.service
              └─6171 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
                --kubeconfig=/etc/kub
```

✓ Kubelet active



# Troubleshooting Worker Nodes

*View and inspect systemd logs*

```
$ journalctl -u kubelet.service
Jan 22 15:51:25 kube-worker-1 systemd[1]: Started kubelet: The Kubernetes Node Agent.
Jan 22 15:51:25 kube-worker-1 systemd[1]: kubelet.service: Current command vanished from the
unit file, execution of the command list won't be resumed.
Jan 22 15:51:25 kube-worker-1 systemd[1]: Stopping kubelet: The Kubernetes Node Agent...
Jan 22 15:51:25 kube-worker-1 systemd[1]: Stopped kubelet: The Kubernetes Node Agent.
Jan 22 15:51:25 kube-worker-1 systemd[1]: Started kubelet: The Kubernetes Node Agent.
Jan 22 15:51:25 kube-worker-1 kubelet[4330]: F0122 15:51:25.656116    4330 server.go:198] failed
to load Kubelet config file /var/lib/kubelet/config.yaml, error failed to read kubelet config
file "/var/lib/kubelet/config"
...
```

 Failed to read config file



# Troubleshooting Worker Nodes

*Check certificate on node*

```
$ openssl x509 -in /var/lib/kubelet/pki/kubelet.crt -text
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 2 (0x2)
    Signature Algorithm: sha256WithRSAEncryption
Issuer: CN = kube-worker-1-ca@1611330698
    Validity
      Not Before: Jan 22 14:51:38 2021 GMT
      Not After : Jan 22 14:51:38 2022 GMT
    Subject: CN = kube-worker-1@1611330698
...
...
```

✓ Certificate issued by correct CA and not expired



---

# EXERCISE

Troubleshooting an  
Issue with a Worker  
Node



---

# Q & A



# Summary & Wrap Up

Last words of advice...



Thank you

