



# Certified Kubernetes Administrator (CKA) Crash Course

Curriculum February 2025



# About the Trainer

Social media handles and web pages



**bmuschko@automatedascent.com**



**benjaminmuschko**



**bmuschko**



**automatedascent.com**

# Exam Objectives and Overview





# Kubernetes Certification Path

Entry level certifications for beginners, hands-on certification for practitioners



Associate



Developer



Administrator

Entry Level

Practitioner

Practitioner

# Exam Objectives

Demonstrate basic understanding of administrative Kubernetes tasks

- Perform typical responsibilities of a Kubernetes administrator.
- The [certification program](#) allows users to demonstrate their competence as part of a hands-on test.
- The badge is well-regarded by employers as a way to set yourself apart from the competition.
- Supplement with practical experience of the Kubernetes ecosystem.



# Practical Application of Skills

You will have to solve practical problems hands-on

- Test takers need to solve problems on a real Kubernetes cluster. There are no multiple choice questions.
- You will have to use `kubectl`. Managing objects can be performed using the imperative or declarative approach.
- A visual dashboard is not available.



# Exam Domains

High-level topics and their weight that contribute to the total score

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



# Curriculum Details

Each domain is broken up into subtopics (find the latest outline on [GitHub](#))

## **Cluster Architecture, Installation and Configuration – 25%**

- Manage role based access control (RBAC)
- Prepare underlying infrastructure for installing a Kubernetes cluster
- Create and manage Kubernetes clusters using kubeadm
- Manage the lifecycle of Kubernetes clusters
- Implement and configure a highly-available control plane
- Use Helm and Kustomize to install cluster components
- Understand extension interfaces (CNI, CSI, CRI, etc.)
- Understand CRDs, install and configure operators

## **Workloads and Scheduling – 15%**

- Understand application deployments and how to perform rolling update and rollbacks
- Use ConfigMaps and Secrets to configure applications
- Configure workload autoscaling
- Understand the primitives used to create robust, self-healing, application deployments
- Configure Pod admission and scheduling (limits, node affinity, etc.)

## **Servicing and Networking – 20%**

- Understand connectivity between Pods
- Define and enforce Network Policies
- Use ClusterIP, NodePort, LoadBalancer service types and endpoints
- Use the Gateway API to manage Ingress traffic
- Know how to use Ingress controllers and Ingress resources
- Understand and use CoreDNS

## **Storage – 10%**

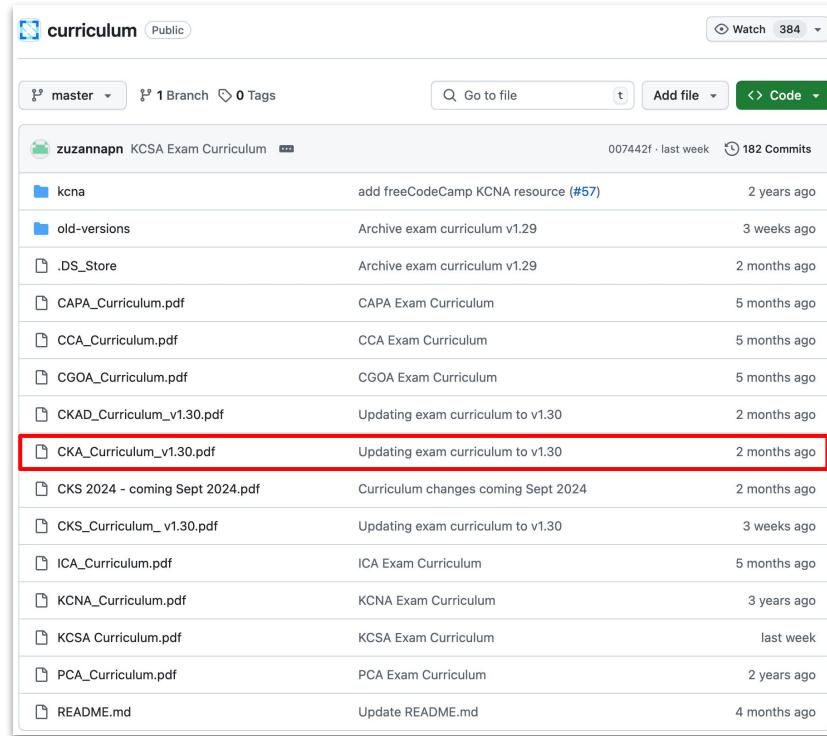
- Implement storage classes and dynamic volume provisioning
- Configure volume types, access modes and reclaim policies
- Manage persistent volumes and persistent volume claims

## **Troubleshooting – 30%**

- Troubleshoot clusters and nodes
- Troubleshoot cluster components
- Monitor cluster and application resource usage
- Manage and evaluate container output streams
- Troubleshoot services and networking

# Kubernetes Version Used in Exam

Suffix in curriculum file name indicates version, changes ~ every 3 months

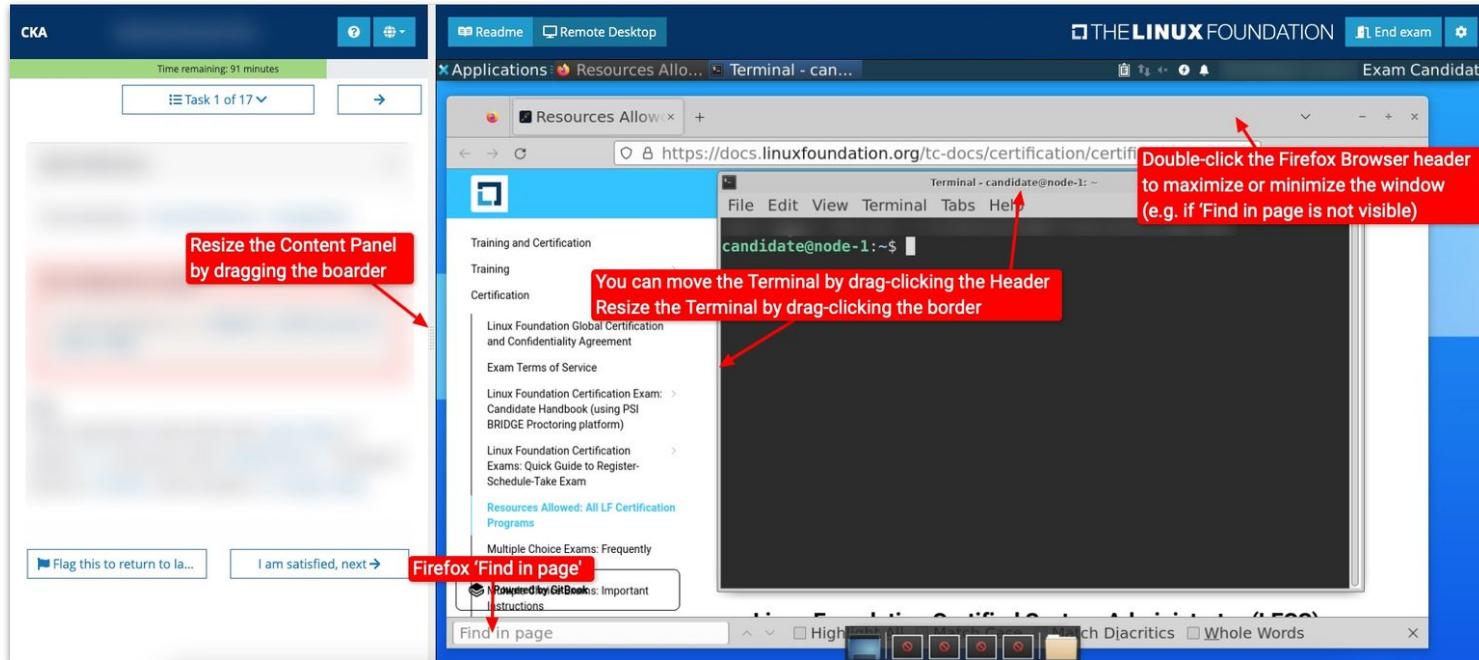


The screenshot shows a GitHub repository named "curriculum" which is public. The repository has 1 branch and 0 tags. It contains 182 commits. The files listed include:

File	Description	Last Commit
kcna	add freeCodeCamp KCNA resource (#57)	2 years ago
old-versions	Archive exam curriculum v1.29	3 weeks ago
.DS_Store	Archive exam curriculum v1.29	2 months ago
CAPA_Curriculum.pdf	CAPA Exam Curriculum	5 months ago
CCA_Curriculum.pdf	CCA Exam Curriculum	5 months ago
CGOA_Curriculum.pdf	CGOA Exam Curriculum	5 months ago
CKAD_Curriculum_v1.30.pdf	Updating exam curriculum to v1.30	2 months ago
<b>CKA_Curriculum_v1.30.pdf</b>	Updating exam curriculum to v1.30	2 months ago
CKS 2024 - coming Sept 2024.pdf	Curriculum changes coming Sept 2024	2 months ago
CKS_Curriculum_v1.30.pdf	Updating exam curriculum to v1.30	3 weeks ago
ICA_Curriculum.pdf	ICA Exam Curriculum	5 months ago
KCNA_Curriculum.pdf	KCNA Exam Curriculum	3 years ago
KCSA_Curriculum.pdf	KCSA Exam Curriculum	last week
PCA_Curriculum.pdf	PCA Exam Curriculum	2 years ago
README.md	Update README.md	4 months ago

# Exam Environment

PSI Remote desktop, single monitor, no bookmarks ([announcement](#))



# Using Kubernetes Documentation

You are allowed to open documentation pages during the exam (see [FAQ](#))

- Docs: <https://kubernetes.io/docs>
- Blog: <https://kubernetes.io/blog>
- Helm: <https://helm.sh/docs>

# Using the Alias for kubectl

Preconfigured in exam environment

```
$ alias k=kubectl
```

```
$ k version
```

```
...
```

Create an alias for the `kubectl` command line tool in your local environment

Use the shortcut to refer to `kubectl`



# Using Auto-Completion for kubectl

Preconfigured in exam environment

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



```
$ kubectl create
```



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

# Working in a Context and Namespace

It's required to solve problems in the correct cluster and namespace

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

```
$ kubectl config use-context <context-of-question>
```

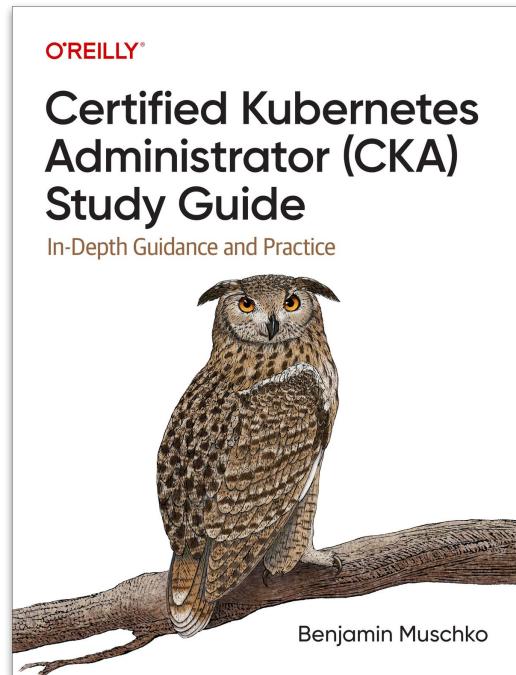
Switches to context  
in kubeconfig file

Sets a context entry  
for the namespace in  
the kubeconfig file



# Companion Study Guide with Practice Questions

Available on Amazon and the O'Reilly learning platform



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>



# Practice Exam on O'Reilly Learning Platform

Interactive labs with automatic scenario and Kubernetes setup

The screenshot shows a web-based learning environment. At the top, there's a navigation bar with 'O'REILLY' and links for 'Topics', 'Start Learning', and 'Featured'. A search bar says 'Search 50,000+ courses, events, titles, and more'. Below the navigation, the main content area has a title 'CKA Prep: Installing a Cluster' and a subtitle 'Step 1 of 3 Initializing the Control Plane Node'. The content includes instructions: 'Start by initializing the control plane on the machine `controlplane`. You are already logged into the machine at the beginning of the lab.' It then asks to initialize the control plane using the `kubeadm init` command, with a note that two command line options are required: `--pod-network-cidr: Use 10.244.0.0/16` for Classless Inter-Domain Routing (CIDR) and `--install-kubernetes-version: 1.29.2`. It also says to check the nodes after running the command. At the bottom, there are buttons for 'Show Solution' and 'Continue'.



## Online access on O'Reilly learning platform

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



# 30-Day Trial Access to the O'Reilly Learning Platform

Check out the content first before committing

The screenshot shows a web browser window for learning.oreilly.com. The page title is "Activate Your Membership". It features fields for "First name", "Last name", "Email", "Create a password", and a "Promo code" field containing "MUSCHKO24". Below these is a dropdown menu set to "Country: USA". At the bottom are two buttons: a red "Activate Account" button and a smaller "Already have an account? Sign In" link. 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

There's no need to buy at full price!

**Cost \$395 | Online Exam**

**REGISTER FOR EXAM**

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

Disclosure: This is an affiliate code.



# Preparing for the Exam

Practice is the key to cracking the exam

- Run through practice exams as often as you can
- Read through online documentation start to end
- Know your tools (especially vim, bash, YAML)

# Don't Stress

The voucher allows for taking the exam twice

- Pick the time you are most comfortable with, get enough sleep
- The exam environment may take a while to get accustomed to
- Take it easy on your first attempt, but give it your best effort
- If you fail the first attempt, write down the areas of knowledge you were less proficient with. Focus on those for the second attempt.

# Cluster Architecture, Installation, and Configuration





# Topics We'll Cover

## Cluster Management

- Installing a cluster from scratch
- Upgrading a cluster version
- Etcd disaster mitigation

## Cluster Architecture

- Highly-available (HA) clusters
- Infrastructure extension interfaces

## Application Stack Management

- Role-based access control (RBAC)
- Using Helm and Kustomize
- Installing and interacting with CRDs and Operators



# Cluster Management

Installing and upgrading the cluster, etcd disaster recovery



# What is Kubeadm?

Tool for creating and maintaining Kubernetes clusters

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



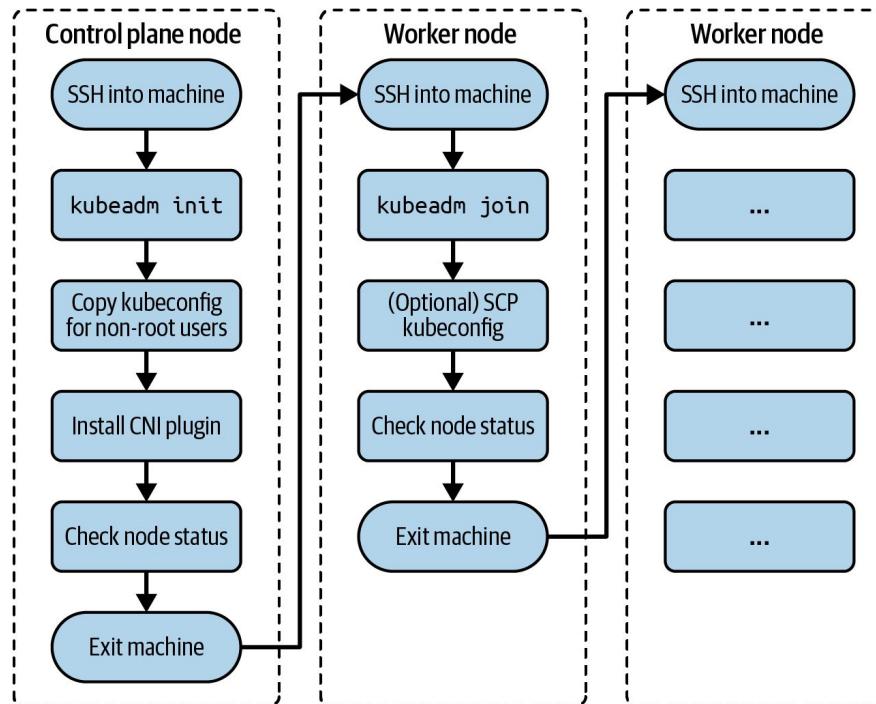
# Installing a Cluster

Start with control plane, join worker nodes

- Initialize the control plane on node using `kubeadm init`.
- Install a [Pod network add-on \(CNI\)](#).
- Join worker nodes using `kubeadm join`.
- Follow the [detailed installation instructions](#) in the documentation.



# Cluster Installation Process





## Exercise

Creating a cluster using  
kubeadm



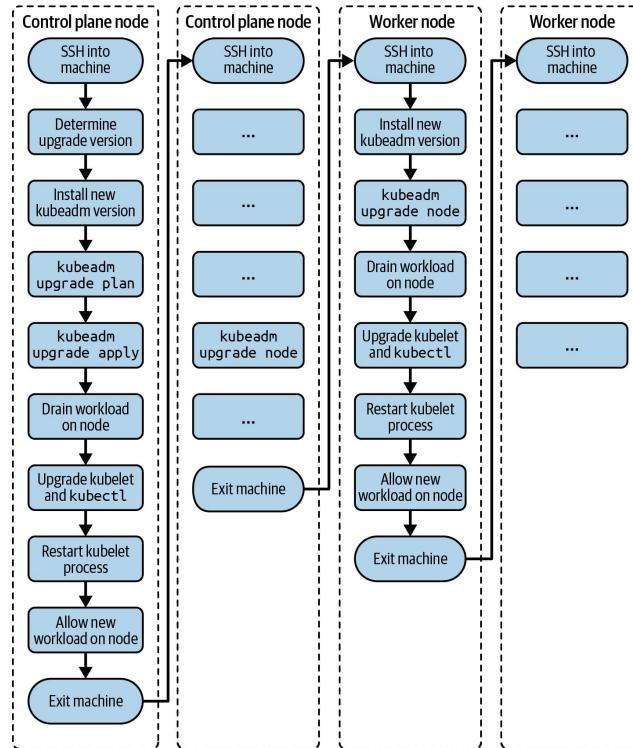


# Upgrading a Cluster Version

Upgrading should be done in version increments

- Determine which version to upgrade to.
- Upgrade control plane node(s).
- Upgrade worker nodes.
- Verify the status of the cluster.
- Follow the [detailed installation instructions](#) in the documentation.

# Cluster Upgrade Process





# Exercise

Upgrading a cluster  
version using kubeadm

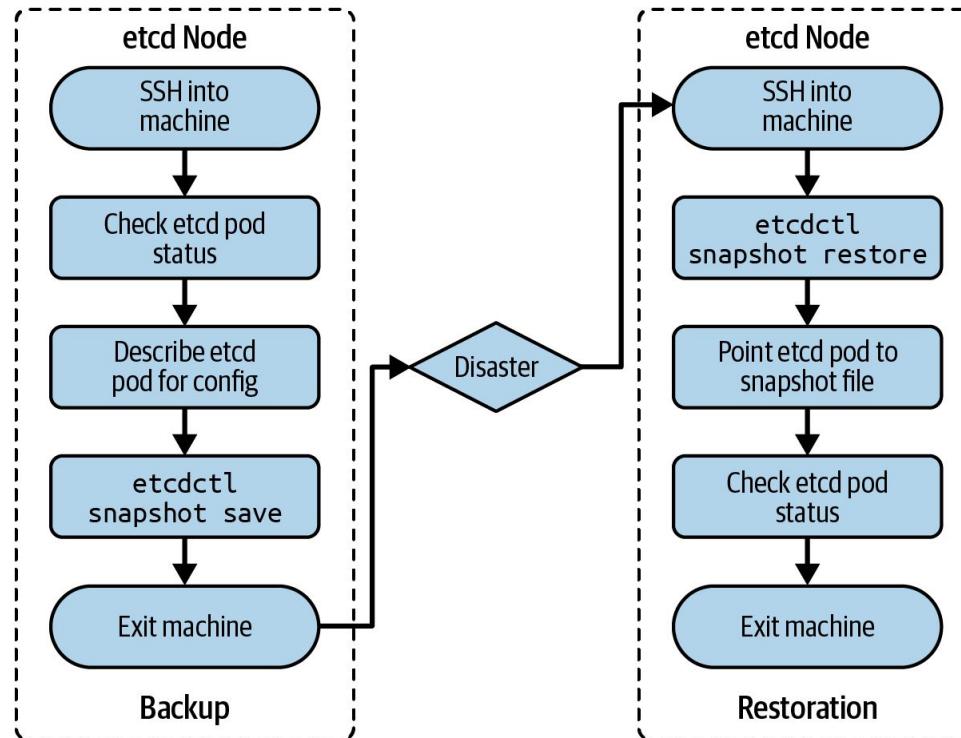


# Backing Up and Restoring etcd

Get etcdctl and etcdutil 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 (deprecated) or with `etcdutil snapshot restore`. The option `--data-dir` is mandatory. Modify the `volumes.hostPath.path` in `/etc/kubernetes/manifests/etcd.yaml` to point to directory.
- Follow the [detailed installation instructions](#) in the documentation.

# Backup and Restoration Process



# Exercise

Backing up and restoring  
etcd



# Exam Essentials

What to focus on for the exam

- The exam puts a strong emphasis on cluster management processes.
- Ensure that you practice processes for installing a cluster, upgrading a cluster version, and backing up and restoring etcd.
- Try to tackle exam questions that exercise those processes to later as they will take a significant amount of time to run.



# Cluster Architecture

High availability (HA) clusters and infrastructure extension interfaces



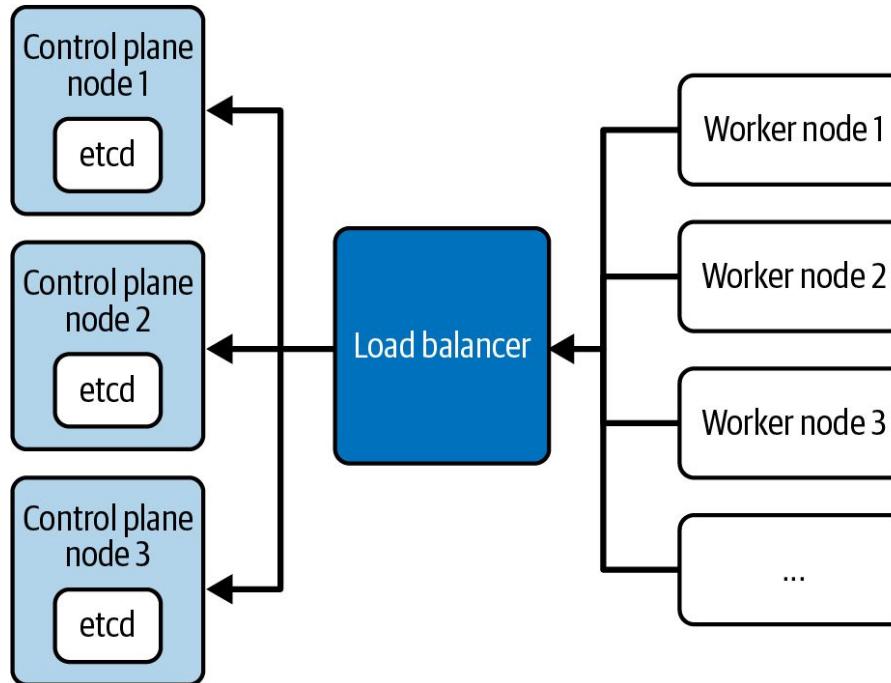
# Managing a Highly Available Cluster

Clusters with a single control plane node can cause outages

- The single control plane node is a single point of failure.
- Any ReplicaSet running on a worker node cannot re-create a Pod due to the inability to talk back to the scheduler running on a control plane node.
- The cluster cannot be accessed externally anymore (e.g., via `kubectl`), as the API server cannot be reached.
- High-availability (HA) clusters help with scalability and redundancy and consist of more than a single control plane node set up with `kubeadm`. Follow the [detailed installation instructions](#) in the documentation.

# Stacked etcd Control Plane Nodes

Etcd is colocated with control plane node



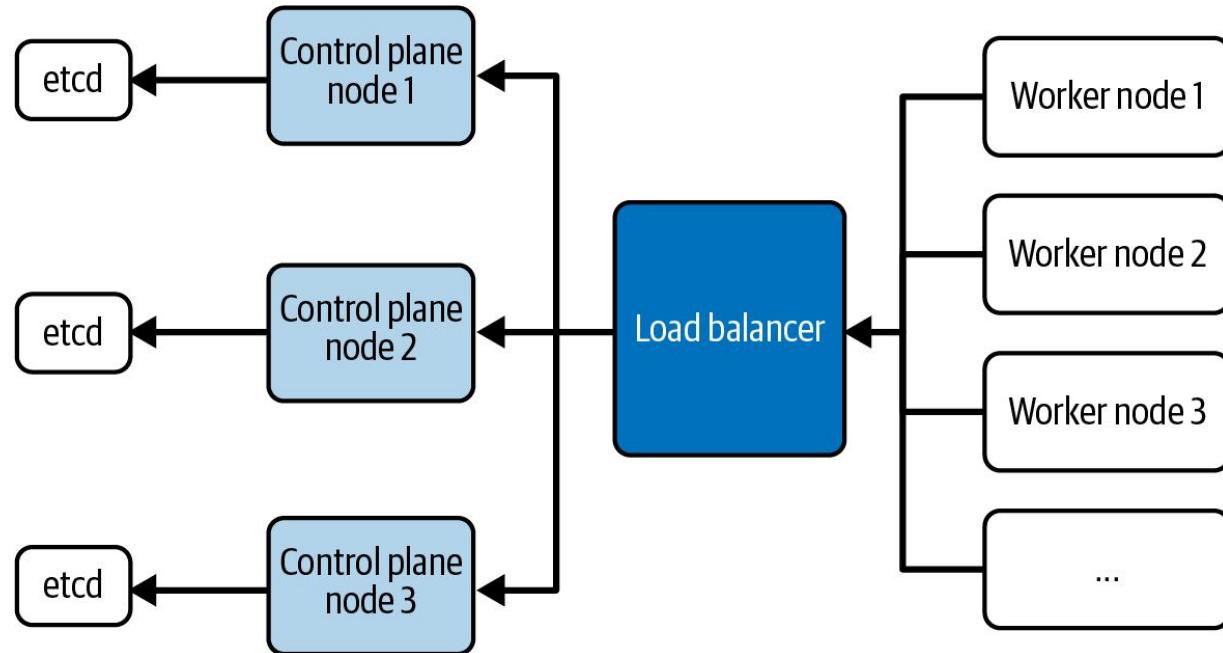
# Stacked etcd Control Plane Nodes

Practical implementation details

- Requires 3 or more machines dedicated to run control plane nodes.
- It runs the risk of losing etcd when the co-located control plane node becomes unavailable.
- This topology requires less infrastructure.

# External etcd Cluster

Etcd runs on dedicated machines



# External etcd Cluster

Decide based on requirements for redundancy, and cost

- Decouples etcd from other control plane functionality and therefore has less of an impact on redundancy when a control plane node or etcd host is lost.
- This topology requires twice as many hosts as the stacked etcd topology, but at a minimum 3 or more machines dedicated to run control plane nodes.

# Which Topology to Choose?

Decide based on requirements for redundancy, and cost

- The *stacked etcd topology* requires less infrastructure and therefore has a lower cost. You will lose the colocated etcd instance if the control plane node goes down.
- The *external etcd node topology* provides a higher degree of redundancy but for a higher cost.
- The [documentation](#) provides more details.

# Exam Essentials

What to focus on for the exam

- Given the complexity of setting up a HA Kubernetes cluster, it's unlikely that you will have to demonstrate its creation or management hands-on.
- Understand the pros and cons of HA clusters, and how to set it up in theory. If you want to create such an architecture in practice, create virtual machines to get a feeling for it.

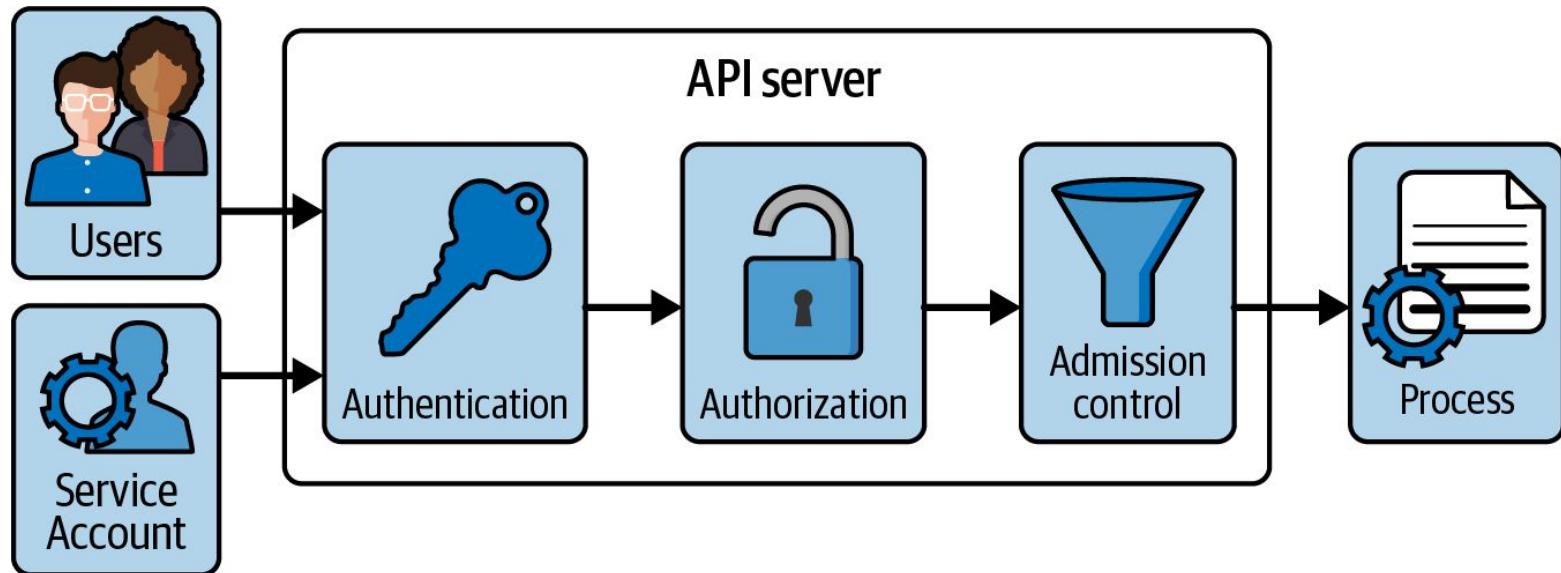


# Authentication and Authorization

Access control for the Kubernetes API

# Processing a Request to the API Server

Any client request to the API server needs to pass three phases



# API Request Processing Phases

Every request has to pass those phases in the following order

- **Authentication:** Validates the identity of the caller using a supported authentication strategy, e.g. client certificates or bearer tokens.
- **Authorization:** Determines if the identity provided in the first stage can access the verb and HTTP path request. This is usually implemented with RBAC.
- **Admission Controller:** Verifies if the request is well-formed and potentially needs to be modified before the request is processed. Ensures that the resource included in the request is valid (could also be implemented as part of admission control).

# Authentication via Credentials in Kubeconfig

The kubeconfig file at `$HOME/.kube/config` is evaluated by `kubectl`

```
apiVersion: v1
kind: Config
clusters:
- cluster:
  certificate-authority: /Users/bmuschko/.minikube/ca.crt
  extensions:
  - extension:
    last-update: Thu, 04 May 2023 08:48:09 MDT
    provider: minikube.sigs.k8s.io
    version: v1.30.1
  name: cluster_info
  server: https://127.0.0.1:58936
  name: minikube
contexts:
- context:
  cluster: minikube
  user: bmuschko
  name: bmuschko
current-context: bmuschko
users:
- name: bmuschko
  user:
    client-certificate-data: REDACTED
    client-key-data: REDACTED
```

API server endpoint for connecting with cluster

Groups access parameters under a convenient name

Client certificate for user

# Managing Kubeconfig and Current Context

Interaction with `kubeconfig` configuration

```
$ kubectl config view ←  
apiVersion: v1  
kind: Config  
current-context: bmuschko  
...
```

Renders the contents of  
the kubeconfig file

```
$ kubectl config current-context ←  
bmuschko
```

Shows the  
currently-selected context

```
$ kubectl config use-context johndoe ←  
Switched to context "johndoe".
```

Switches to the context  
defined in the kubeconfig

```
$ kubectl config current-context  
johndoe
```

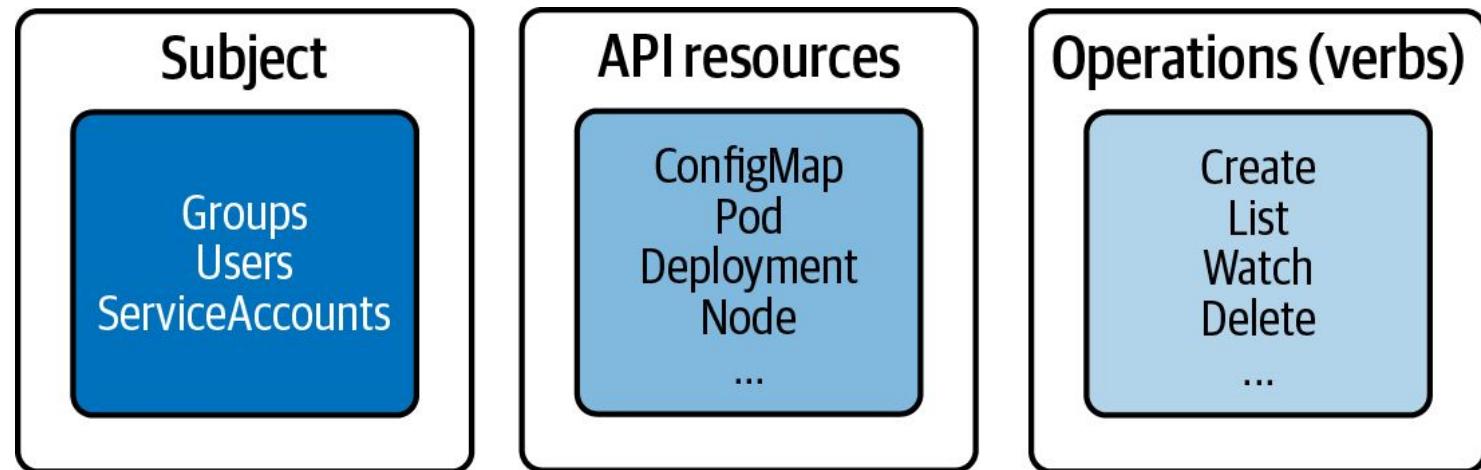
# What is RBAC?

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.”

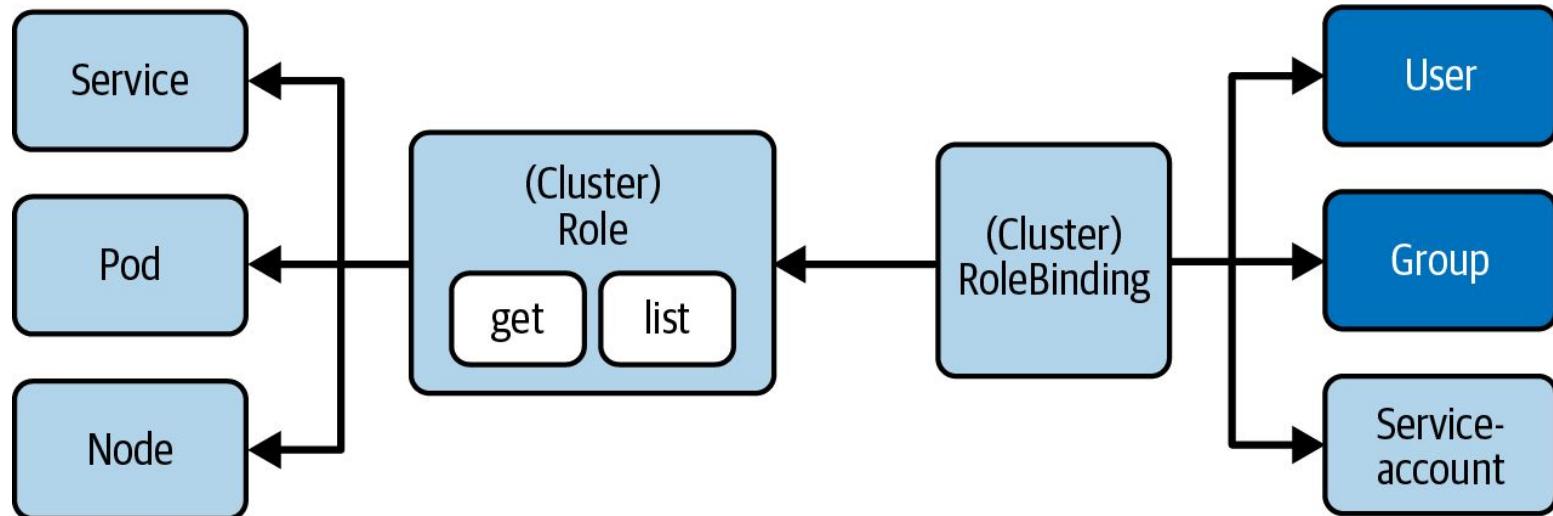
# RBAC High-Level Overview

Three key elements for understanding concept



# Involved RBAC Primitives

Restrict access to certain operations of API resources for subjects





# Namespace and Cluster-Wide Permissions

Defining permission scopes

- The *Role* maps API resources to verbs for a single namespace.
- The *RoleBinding* maps a reference of a Role to one or many subjects for a single namespace.
- To apply permissions across all namespaces of a cluster or cluster-wide resources, use the corresponding API primitives *Cluster* and *ClusterRoleBinding*. The configuration options of the manifests are the same.

# Creating a Role with Imperative Approach

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 be spelled out explicitly (in  
this case apps/Deployment )

# Getting Role Details

Maps objects of a Kubernetes primitive 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 with Imperative Approach

Maps 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: johndoe
```

← 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
```

# What is a Service Account?

Non-human request to Kubernetes API from a process

- Processes running outside of Kubernetes or processes running inside of a container may need to interact with the Kubernetes API.
- A Service Account allows for authenticating with the API server through an authentication token.
- **Example:** “I have a CI/CD process that retrieves Pod information by making calls to the Kubernetes API.”



# Service Account and RBAC

## Assigning targeted permissions

- Authorization is controlled through RBAC and assigned to the Service Account.
- If not assigned explicitly, a Pod uses the `default` Service Account. The `default` Service Account has the same permissions as an unauthenticated user.
- Only assign the RBAC permissions to a Service Account that it needs to fulfill the operation(s) it is trying to perform.

# Service Account as Subject

Pod assigns the Service Account by name



# Using the kubernetes Service

Convenient way to get API server endpoint from within the cluster

```
$ kubectl get service kubernetes
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	13h

```
$ kubectl run api-call --image=alpine:3.16.2 -it --rm <
```

```
--restart=Never -- wget https://kubernetes.default.svc.cluster.local/<  
api/v1/namespaces/k97/pods
```

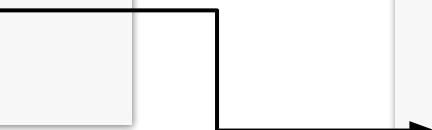
API endpoint for  
listing all Pods in  
namespace k97

# Service Account YAML Manifest

In a Pod manifest, refer to the Service Account by name

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

```
apiVersion: v1
kind: Pod
metadata:
  name: list-pods
  namespace: k97
spec:
  serviceAccountName: sa-api
  ...
```





# Accessing Service Account Authentication Token

Automounted at specific Volume mount path

```
$ kubectl describe pod list-pods -n k97
...
Containers:
  pods:
    ...
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from  ↪
        kube-api-access-xnkwd (ro)
      ...
$ kubectl exec -it list-pods -n k97 -- /bin/sh
# cat /var/run/secrets/kubernetes.io/serviceaccount/token
eyJhbGciOiJSUzI1N...
# exit
```

# 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 "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)" https://kubernetes.default.svc.cluster.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



# Exam Essentials

What to focus on for the exam

- All clients will make a call to the API server which will take care of authentication and authorization.
- Role-Based Access Control (RBAC) defines the permissions for permitted operations on specific API resources.
- RBAC kicks in every time the API server receives a request.



# Exam Essentials

What to focus on for the exam

- A Role and RoleBinding define permissions for objects in a namespace. ClusterRole and ClusterRoleBinding define permissions across all namespaces.
- Process that need access to the Kubernetes API use a Service Account. The Service Account is subject that can be tied in with RBAC.



# Helm

Deploying an application defined by a set of YAML manifests

# 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 (e.g., for running Grafana or PostgreSQL).



# 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>

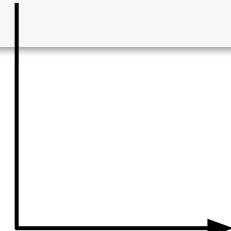
...

CHART

11.0.6

VERSION APP

2.361.2



The screenshot shows the Jenkins chart page on artifacthub.io. It features the Bitnami logo and a brief description of Jenkins. Below the description are several buttons: INSTALL, TEMPLATES, DEFAULT VALUES, VALUES SCHEMA, and CHANGELOG. At the bottom, it displays the application version (2.361.2) and chart version (11.0.6), along with download links.



# 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.
```

# Exercise

Installing an existing Helm chart from the central chart repository



# Standard Chart Structure

Predefined directory structure with mandatory `Chart.yaml` and `values.yaml`

```
$ tree
.
├── Chart.yaml
└── templates
    ├── web-app-pod-template.yaml
    └── web-app-service-template.yaml
└── values.yaml
```

# Chart File

Describes the meta information of the chart (e.g., name and version)

*Chart.yaml*

```
apiVersion: 1.0.0
name: web-app
version: 2.5.4
```



# Values File

Key-value pairs used at runtime to replace placeholders in the YAML manifests

*values.yaml*

```
db_host: mysql-service
db_user: root
db_password: password
service_port: 3000
```

Additional [built-in objects](#) are available for use.

# Template File

YAML manifest that can define placeholders using double curly braces

*web-app-pod-template.yaml*

```
apiVersion: v1
kind: Pod
metadata:
  name: web-app
spec:
  containers:
    - image: bmuschko/web-app:1.0.1
      name: web-app
      env:
        - name: DB_HOST
          value: {{ .Values.db_host }}
        - name: DB_USER
          value: {{ .Values.db_user }}
        - name: DB_PASSWORD
          value: {{ .Values.db_password }}
...
...
```



# Previewing Final Templates

Replaces placeholders with actual values from `values.yaml`

```
$ helm template .  
---  
# Source: Web Application/templates/web-app-pod-template.yaml  
apiVersion: v1  
kind: Pod  
metadata:  
  name: web-app  
spec:  
  containers:  
    - image: bmuschko/web-app:1.0.1  
      name: web-app  
      env:  
        - name: DB_HOST  
          value: mysql-service  
        - name: DB_USER  
          value: root  
        - name: DB_PASSWORD  
          value: password  
  ...
```



# Installing a Chart from Local Files

Helpful for trying out the deployment before publishing the chart file

```
$ helm install web-app .
```

```
NAME: web-app
LAST DEPLOYED: Wed Apr 19 11:58:34 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

```
$ kubectl get pods,services
```

NAME	READY	STATUS	RESTARTS	AGE
pod/web-app	1/1	Running	0	3m24s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT (S)	AGE
service/web-app-service	NodePort	10.109.44.239	<none>	3000:30456/TCP	3m24s

# Overriding Values

End user can tweak runtime behavior to personal requirements

```
$ helm install --namespace custom --create-namespace --set service_port=9090 web-app .  
NAME: web-app  
LAST DEPLOYED: Wed Apr 19 11:58:34 2023  
NAMESPACE: custom  
STATUS: deployed  
REVISION: 1  
TEST SUITE: None
```



Providing a custom namespace (defaults to default namespace)

Override values in values.yaml

# Bundling the Template Files into a Chart Archive File

Archive file is usually published to a chart repository for consumption

```
$ helm package .
```

Successfully packaged chart and saved it to:

```
/Users/bmuschko/helm/web-app-2.5.4.tgz
```

```
$ ls
```

```
web-app-2.5.4.tgz
```

# Exercise

Implementing, packaging,  
and installing a custom  
Helm chart



# Exam Essentials

What to focus on for the exam

- Helm is an open source tool for installing a set of YAML manifests (chart) from a central repository. The tool uses templating functionality for providing custom, runtime values.
- Practice discovering, and installing existing charts using the Helm executable.
- You are not required to understand how to build and publish your own charts. I'd recommend trying it out yourself though so you can understand the workflow.



# Kustomize

Template-free customization of application configuration

# What is Kustomize?

Customizing application manifests

- Writing manifests for application stacks is straightforward, however, deploying them to different Kubernetes runtime environments with varying configuration gets tricky.
- Instead of copying and modifying manifests per environment, you can use Kustomize to merge configuration change.





# How Does Kustomize Work?

Operating without templates

- Kustomize does not require using a template engine or changing the original manifests.
- It applies changes upon execution.
- Kustomize can be used as a subcommand/flag of `kubectl` or as a separate [executable](#).





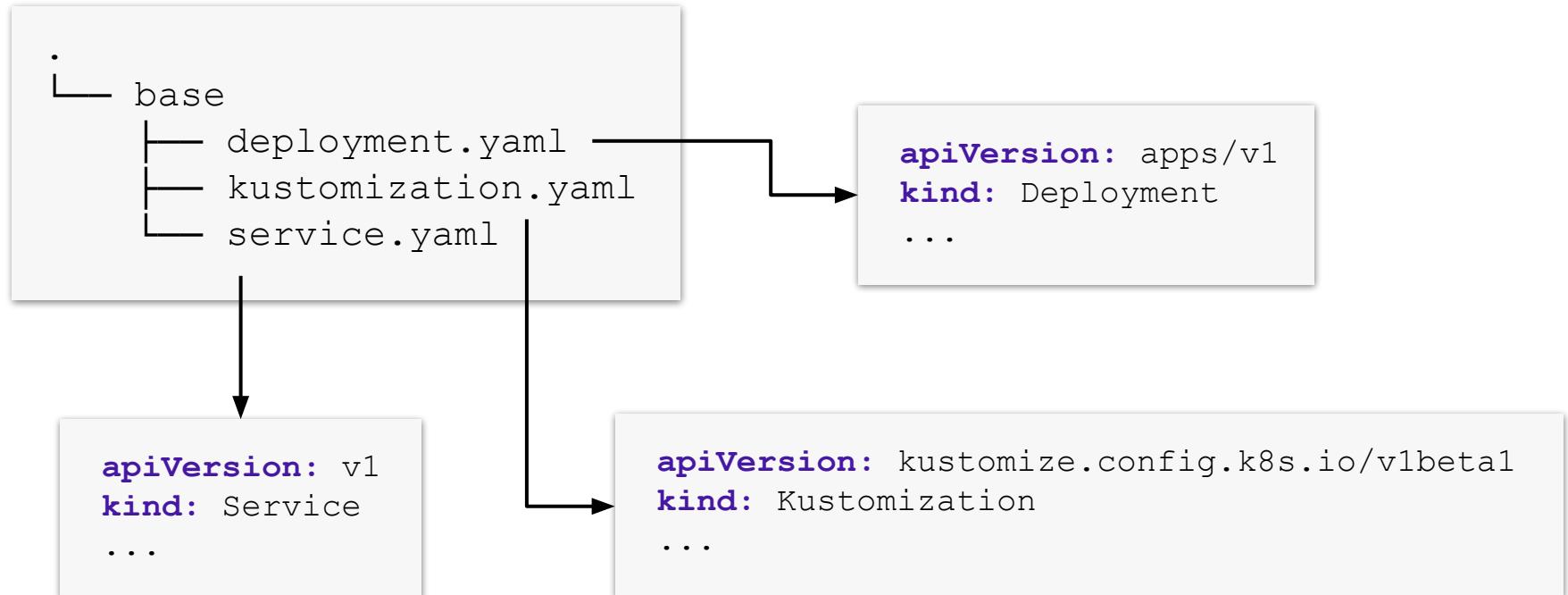
# Kustomize Use Cases

Convenient manifest management

- Generating manifests from other sources. For example, creating a ConfigMap and populating its key-value pairs from a properties file.
- Adding common configuration across multiple manifests. For example, adding a namespace and a set of labels for a Deployment and a Service.
- Composing and customizing a collection of manifests. For example, setting resource boundaries for multiple Deployments.

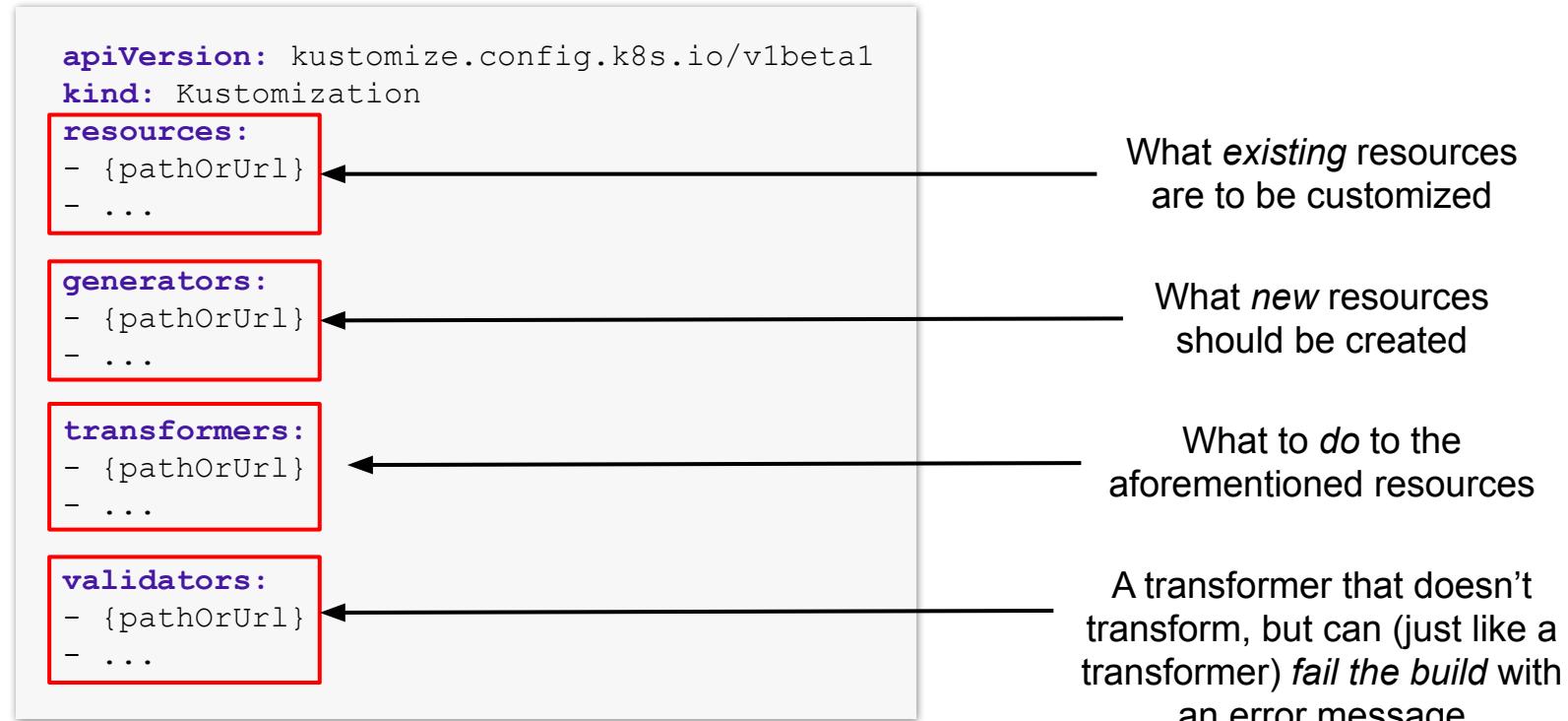
# Basic Kustomize Setup

Regular YAML manifests + a `kustomization.yaml` file



# Kustomization File Structure

Generate or transform other Kubernetes Resource Model (KRM) objects



# Example Kustomization File

Merges labels into a set of resources

*kustomization.yaml*

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
```

```
labels:
  layer: backend
```

Defines labels for all  
processed resources

```
resources:
  - deployment.yaml
  - service.yaml
```

Declares resources  
to be modified



# Generating a Kustomize File

Imperative command to create `kustomization.yaml` file

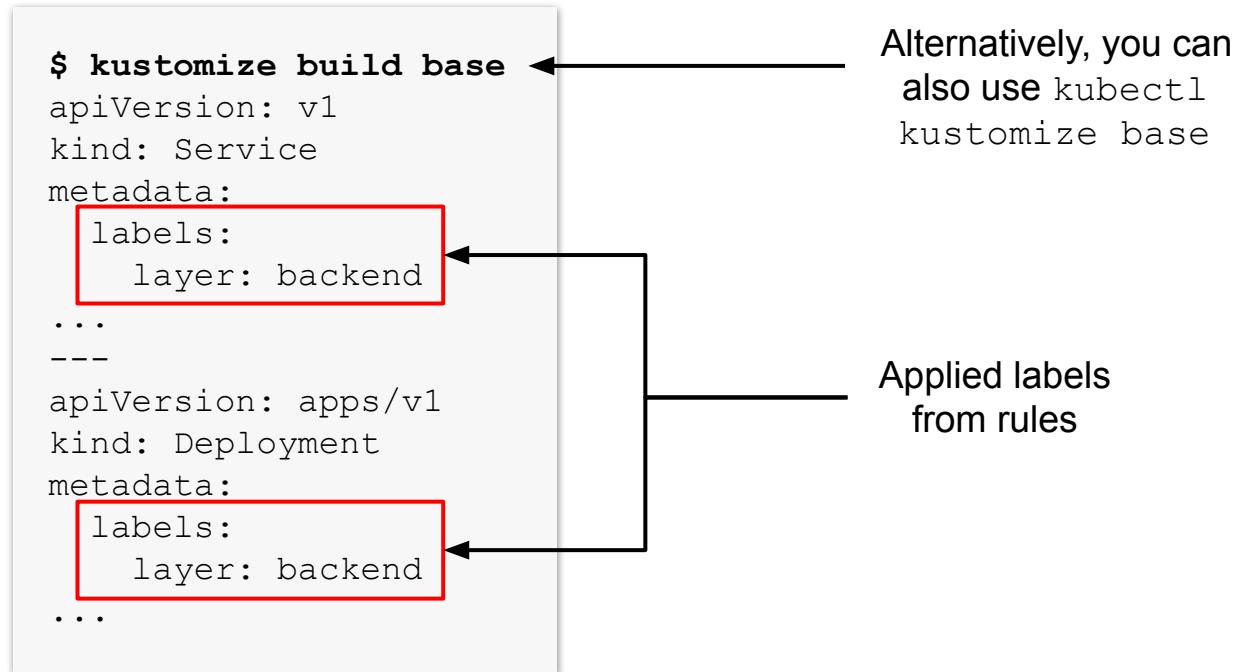
```
$ cd base
$ ls .
deployment.yaml service.yaml

$ kustomize create --resources=deployment.yaml,service.yaml <
--labels=layer:backend

$ ls .
deployment.yaml kustomization.yaml service.yaml
```

# Building and Previewing a Set of Resources

Applies the rules in `kustomization.yaml` and renders the result to the output





# Managing the Resources

Use the typical commands provided by `kubectl`

```
$ kubectl apply -k base
```

```
service/backend-service configured
deployment.apps/backend-deployment created
```

```
$ kubectl get deployments,services
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/backend-deployment	1/1	1	1	7s

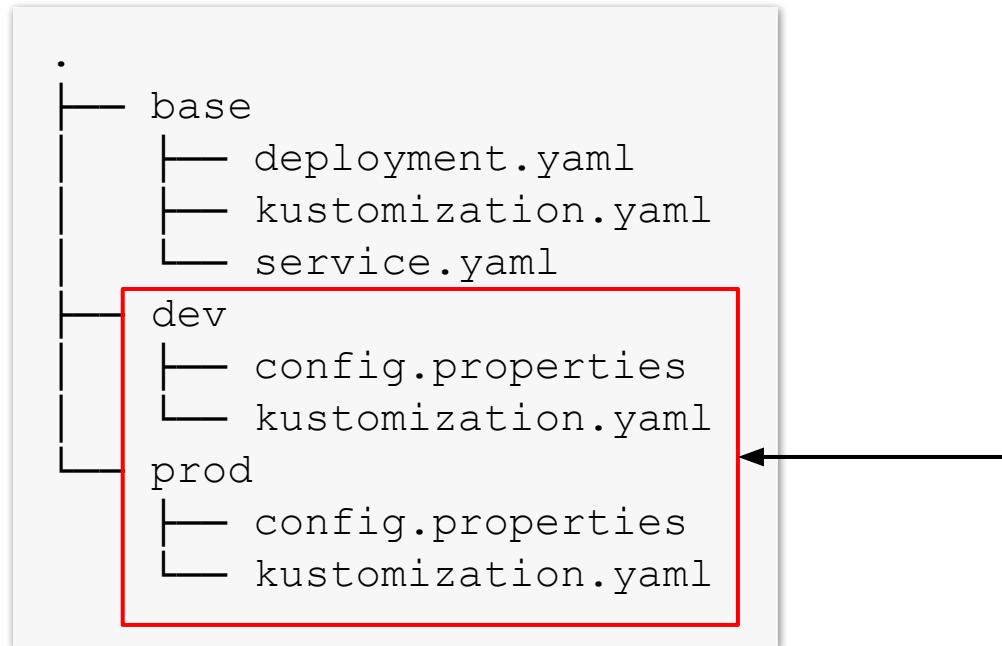
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	...
service/backend-service	ClusterIP	10.107.228.58	<none>	...

```
$ kubectl delete -k base
```

```
service "backend-service" deleted
deployment.apps "backend-deployment" deleted
```

# Environment Kustomize Setup

Subdirectories contain environment-specific configuration



Create a dedicated  
directory + configuration  
per environment

# Example Production Environment

Base configuration is merged with the configuration of an environment

*prod/kustomization.yaml*

```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
- ../base/
namespace: prod
namePrefix: prod-
configMapGenerator:
- name: backend-configmap
  env: config.properties
replicas:
- name: backend-deployment
  count: 10
```

# Merging Environment Configuration

Base configuration + environment configuration

```
$ kustomize build prod
apiVersion: v1
data:
  DB_USERNAME: prod-user
kind: ConfigMap
metadata:
  name: prod-backend-configmap-7gbb88gbhc
  namespace: prod
---
apiVersion: v1
kind: Service
metadata:
  labels:
    layer: backend
  name: prod-backend-service
  namespace: prod
...
...
```

# Managing the Resources

Command is the same as for managing a the base configuration

```
$ kubectl apply -k prod
configmap/prod-backend-configmap-7gbb88gbhc created
service/prod-backend-service created
deployment.apps/prod-backend-deployment created
```

# Kustomize Benefits

A good option for operators managing their own manifests

- **Native to kubectl:** Kustomize does not necessarily require you to install additional tooling.
- **DRY approach:** Define a base configuration just once with the ability to reuse it. Environment-specific configuration will be added as an overlay.

# Kustomize Benefits

A good option for operators managing their own manifests

- **Easy to learn:** Does not require learning another template engine. You'll use standard YAML manifests to define resources.
- **Scalable variants:** New configuration e.g. environments can be easily added.



## Exercise

Generating a Secret and  
patching a Deployment  
with Kustomize



# Exam Essentials

What to focus on for the exam

- Kustomize is an alternative to Helm without the need to publish and use a binary artifact.
- Kustomize is better suited for operators that deeply understand manifests to deploy and manage application stacks or cluster components.



# Exam Essentials

What to focus on for the exam

- Understand the Kustomize-specific commands available to `kubectl`.
- Gain a broad overview over operations available to a `kustomization.yaml` file.
- Come up with some scenarios that you suitable for using Kustomize and exercise them.
- For an interesting comparison between Helm and Kustomize, view this [video](#).



# Extension Interfaces

Enhancing the cluster node functionality

# What are Extension Interfaces?

Kubernetes exposes APIs to extend the core functionality

- You can envision Kubernetes as a platform with core functionality that can be enhanced by APIs.
- Some extension interfaces are mandatory and need to be provided with an implementation for Kubernetes to work properly.
- For example, you need to provide a Container Network Interface (CNI) plugin so that Pods can be assigned virtual IP address for communication purposes when created.



# What are Extension Interfaces?

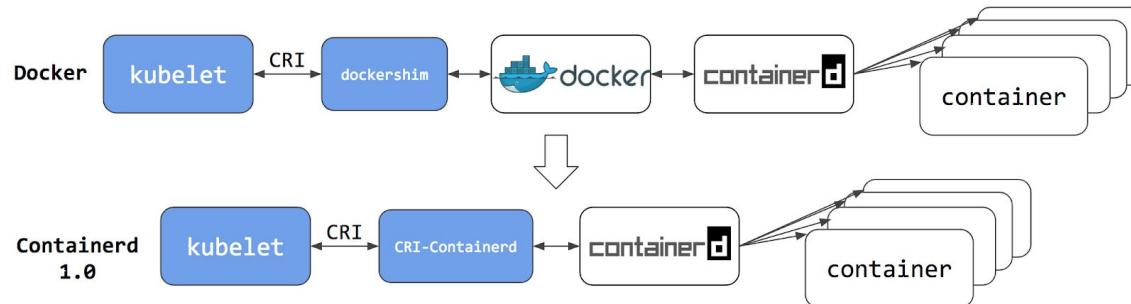
Kubernetes exposes APIs to extend the core functionality

- Other extension interfaces are optional and can enhance Kubernetes if needed.
- For example, the Container Storage Interface (CSI) can support proprietary storage solutions.

# Container Runtime Interface (CRI)

Responsible for running and managing containers

Versions of Kubernetes before 1.24 simply shipped with Docker Engine which communicated through [dockershim](#). The container runtime implementation has been made configurable in Kubernetes 1.24. [Containerd](#) became the default implementation of this interface.

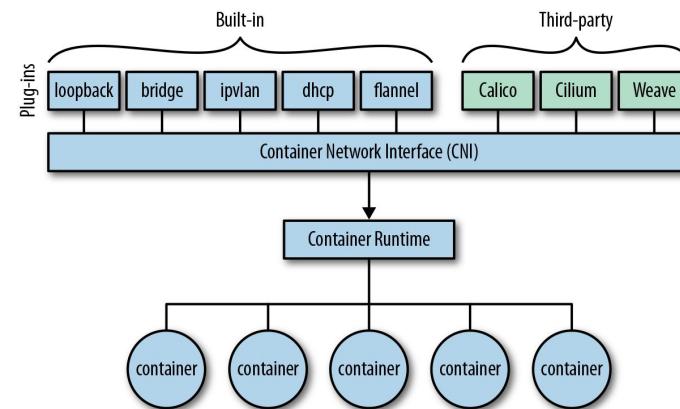


Source: [Kubernetes blog](#)

# Container Network Interface (CNI)

Responsible for managing network interfaces for the containers

Implementations of this interface configure the network, provision IP addresses, and maintain connectivity with multiple hosts – [Calico](#), [Cilium](#), and other network overlays use CNI.

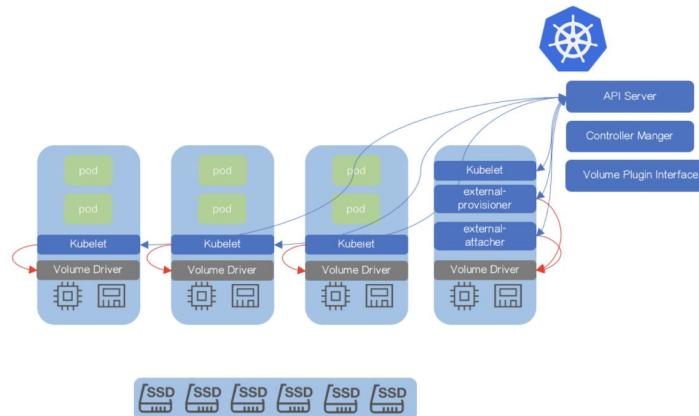


Source: O'Reilly "[Container Networking](#)" Report

# Container Storage Interface (CSI)

Manages container volumes attached to running containers

Can be persistent or ephemeral storage that is attached to running containers. Most cloud providers implement CSI for their storage solutions, e.g. [AWS EFS](#).



Source: [Kubernetes blog](#)



# Exam Essentials

What to focus on for the exam

- Understand that Kubernetes can be extended by APIs, so-called extension interfaces.
- Common extension interfaces include CRI, CNI, CSI. You are not expected to configure an implementation for all of those interfaces.
- Configuring a CRI and CNI for an on-prem Kubernetes cluster is mandatory to make it operational. You will find the necessary steps in the Kubernetes documentation.



# Custom Resource Definitions (CRDs)

Extending the Kubernetes API

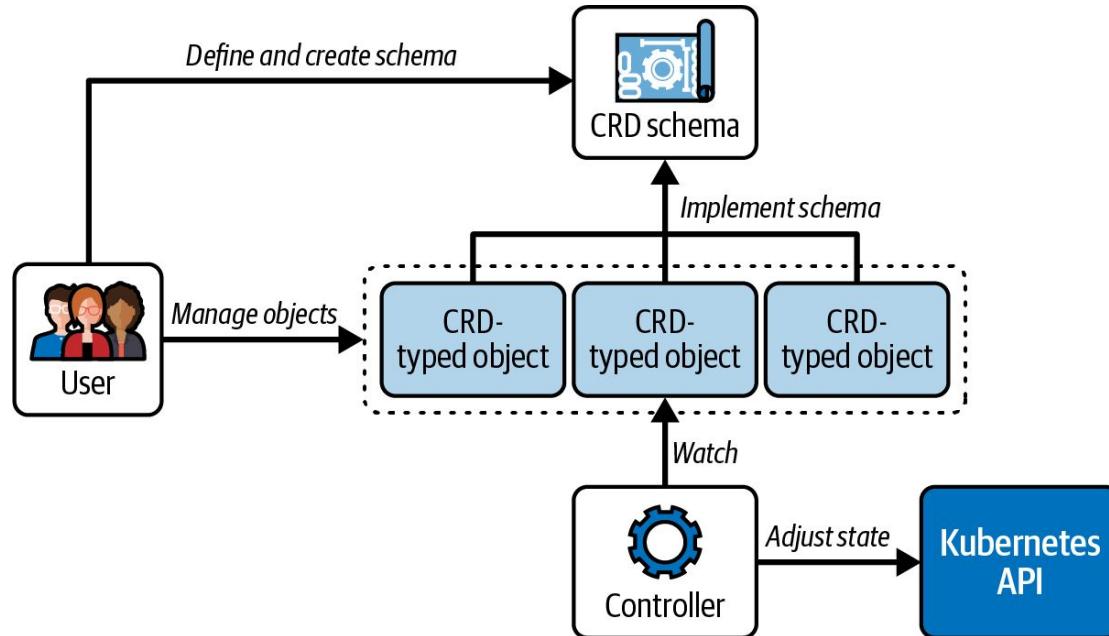
# What is a Custom Resource Definition (CRD)?

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 CRD allows you to define, create, and persist one or many custom objects and expose them with the Kubernetes API.
- CRDs are combined with *controllers* to make them actually useful. Controllers interact with the API server and implement the custom logic. This is called the *Operator pattern*.

# The Operator Pattern

Consists of CRDs and Controllers



# Example CRD

Smoke testing against a Service after deployment

- **Assumption:** An web-based application stack is deployed via a Deployment with one or more than one replica. The Service object routes network traffic to the Pods.
- **Desired functionality:** After the deployment happened, we want to run a quick smoke test against the Service's DNS name to ensure that application is operational. The result of the smoke test will be sent to an external service for the purpose of rendering it on the UI dashboard.
- **Goal:** Implement the Operator pattern (CRD and controller) for the use case.

# CRD YAML Manifest

Defines the schema for the custom object

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: smoketests.stable.bmuschko.com
spec:
  group: stable.bmuschko.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                service:
                  type: string
              ...
  scope: Namespaced
  names:
    plural: smoketests
    singular: smoketest
    kind: SmokeTest
    shortNames:
      - st
```

Combination of <plural>.<group>

Versions supported by CRD

Attributes for custom type

Identifiers for custom type

# Custom Resource Object YAML Manifest

Instantiation of CRD kind

```
apiVersion: stable.bmuschko.com/v1
kind: SmokeTest
metadata:
  name: backend-smoke-test
spec:
  service: backend
  path: /health
  timeout: 600
  retries: 3
```

← Group and Version of custom kind

Kind defined by CRD

Attributes and values that make  
custom kind configurable



# Creating the CRD and Custom Object

CRD object needs to be created before custom objects

```
$ kubectl create -f loadtest-resource.yaml
customresourcedefinition.apiextensions.k8s.io/smoketests.stable.bmuschko.com
created

$ kubectl create -f loadtest.yaml
smoketest.stable.bmuschko.com/backend-smoke-test created
```



# Discovering CRDs

Once installed, you can list all available CRDs

```
$ kubectl api-resources --api-group=stable.bmuschko.com
NAME      SHORTNAMES      APIVERSION      NAMESPACED      KIND
smoketests  st            stable.bmuschko.com/v1  true          SmokeTest
```

```
$ kubectl get crd
NAME                  CREATED AT
smoketests.stable.bmuschko.com  2023-05-04T14:49:40Z
```

# Controller Implementation

The controller acts upon the state of CRDs

- You can interact with the custom resource object using CRUD (create/read/update/delete) operations; however, no smoke test logic will actually be initiated.
- A controller acts as a reconciliation process by inspecting the state through the API server.
- Controllers can use one of the Kubernetes [client libraries](#), written in Go or Python, to access custom resources.

# Smoke Test Controller

The actual logic for smoke tests lives in the controller

- The controller checks for the state of `SmokeTest` objects by making a call to the relevant Kubernetes API endpoints.
- It can inspect the state of the objects and initiate the execution of a HTTP call to the Service.
- After the operation has finished, it sends the result to the external service to capture smoke test outcomes over time for monitoring, dashboarding, and notification purposes.

## Exercise

Defining and interacting  
with a CRD



# Exam Essentials

What to focus on for the exam

- You are not expected to implement a CRD schema yourself. All you need to know is how to discover and use them. Controller implementations are out-of-scope.
- More specifically, learn how to use the `kubectl get crds` command to discover installed CRDs, and how to create objects from a CRD schema.
- If you want to go further, install an open source CRD, e.g. the [external secrets](#) operator, and inspect its schema.

# Workloads and Scheduling



# Topics We'll Cover

## Deployments and ReplicaSets

- Managing sets of Pods
- Rolling out changes to replicas

## Scaling of Workload

- Manual scaling of workload
- Automatic scaling of workload with the HorizontalPodAutoscaler

## Scheduling and Workload Admission

- Aspects that influence scheduling of workload, e.g. Node Affinity, Taints and Tolerations
- Admission control of workload with LimitRange, and ResourceQuota

## Configuration Data

- Creating ConfigMaps and Secrets
- Use case-driven consumption options from a Pod



# Deployments

Scaling workload, deployment strategies



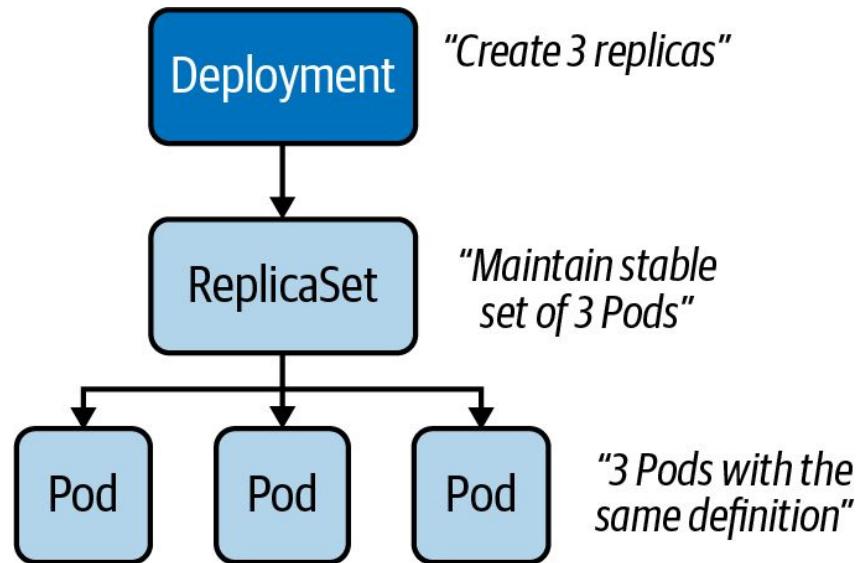
# What is a Deployment?

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 with Imperative Approach

Creates objects for the Deployment, ReplicaSet, and Pod(s)

```
$ 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 Template Attributes

Replica configuration is defined under `spec.template`

```
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
```

*Deployment*

Template spec attributes  
are the same as for a  
Pod spec



*Pod*

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - image: nginx:1.14.2
      name: nginx
```

# Deployment YAML Manifest

Creates an error if label selector and template labels do not match

```
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

# Listing 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

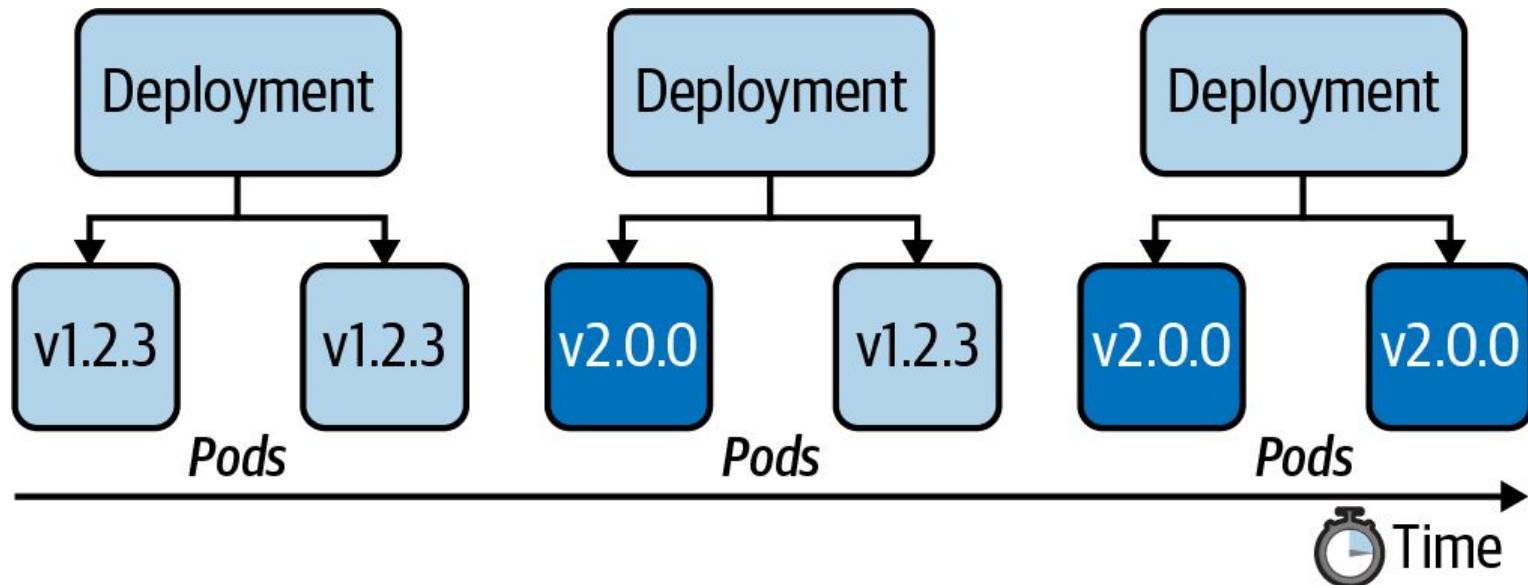
```
$ kubectl describe deployment.apps/my-deploy
Replicas:           1 desired | 1 updated | 1 total | 1 available |
                  0 unavailable
NewReplicaSet:    my-deploy-8448c488b5 (1/1 replicas created)
```

```
$ 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

# Setting a Change Cause for a Revision

Tracked by annotation with key `kubernetes.io/change-cause`

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

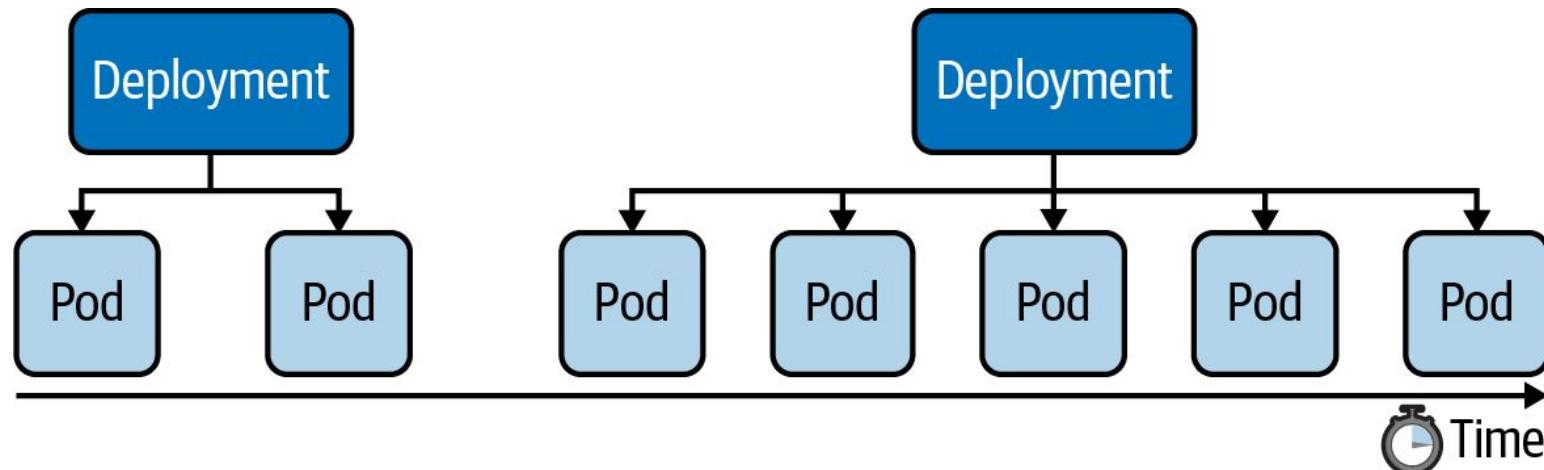
```
$ kubectl annotate deployment my-deploy
kubernetes.io/change-cause="image updated to 1.16.1"
deployment.apps/my-deploy annotated
```

```
$ kubectl rollout history deployment my-deploy
deployment.apps/my-deploy
REVISION  CHANGE-CAUSE
2          <none>
3          image updated to 1.16.1
```

What changed  
with this revision?

# Manually Scaling a Deployment

"We measured performance under load. Scale up to exactly x number of Pods."



# Changing the Number of Replicas

Use scale command or change spec.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

## Exercise

Performing rolling updates  
and manually scaling a  
Deployment





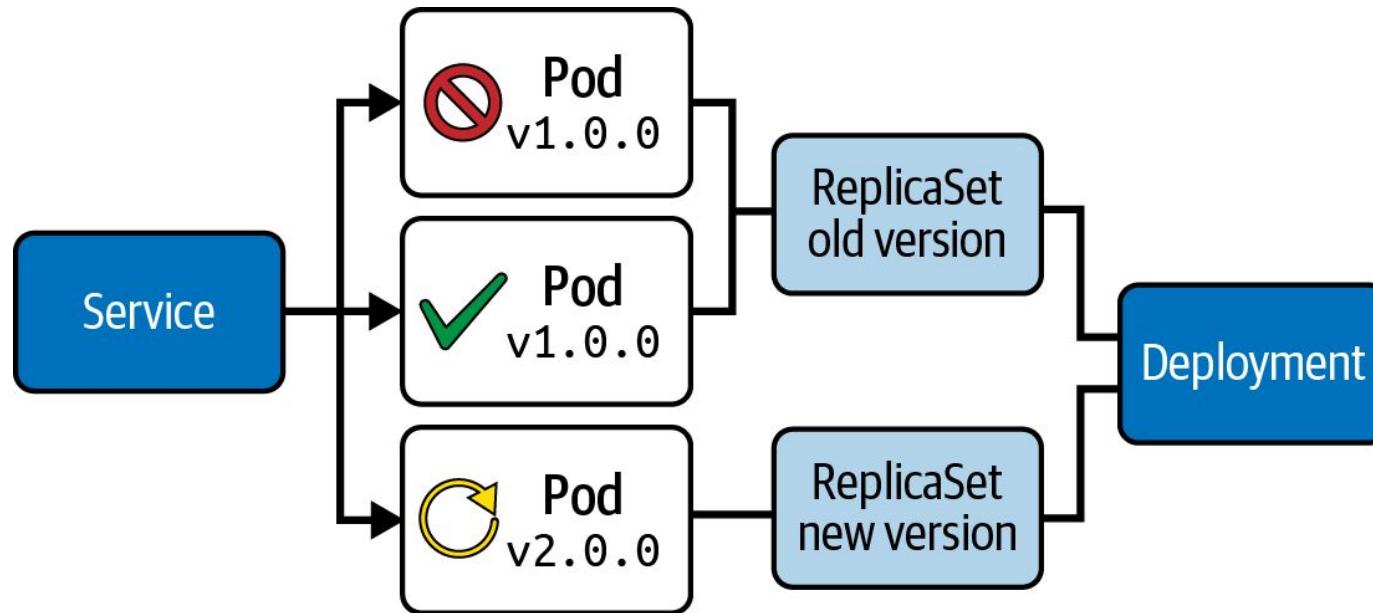
# Common Deployment Strategies

Choosing the right deployment procedure depends on the needs

- **Rolling Update:** Release a new version on a rolling update fashion, one after the other.
- **Recreate:** Terminate the old version and release the new one.
- **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.

# Rolling Update Deployment Strategy

Secondary ReplicaSet is created with new version of application



# Rolling Update Deployment Strategy

Runtime behavior can be fine-tuned

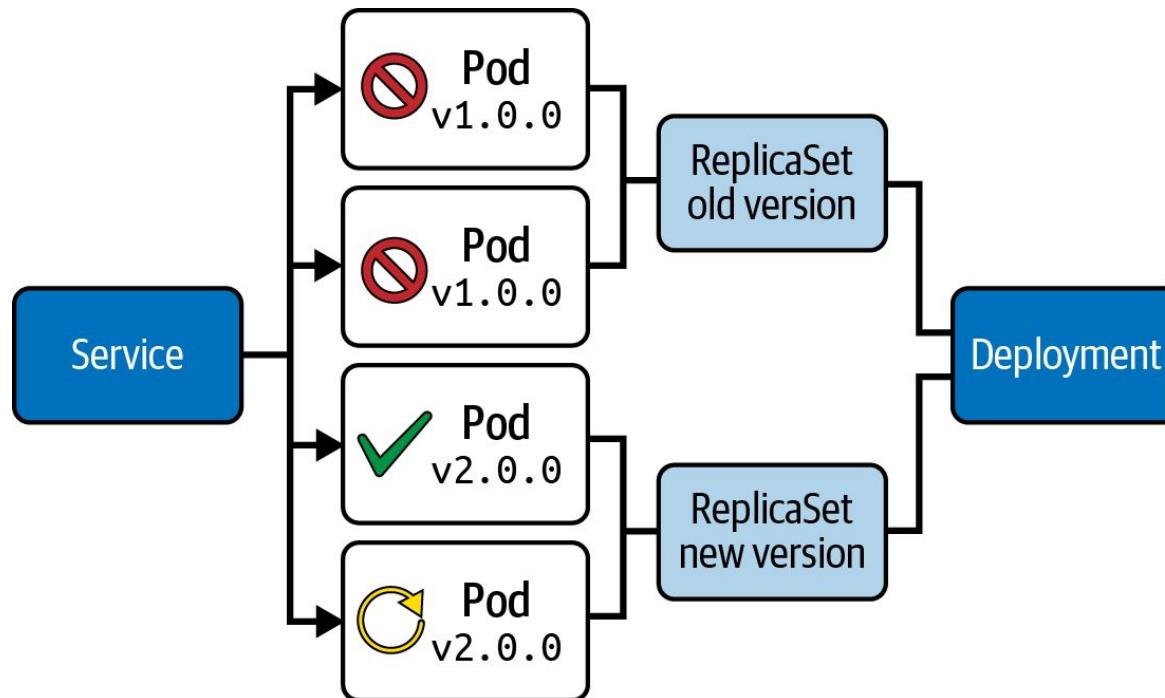
```
spec:  
  replicas: 3  
  strategy:  
    type: RollingUpdate  
    rollingUpdate:  
      maxSurge: 2  
      maxUnavailable: 0
```

Determines how update  
should be performed

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

# Recreate Deployment Strategy

Terminates all the running instances then recreate them



# Recreate Deployment Strategy

No configuration options available

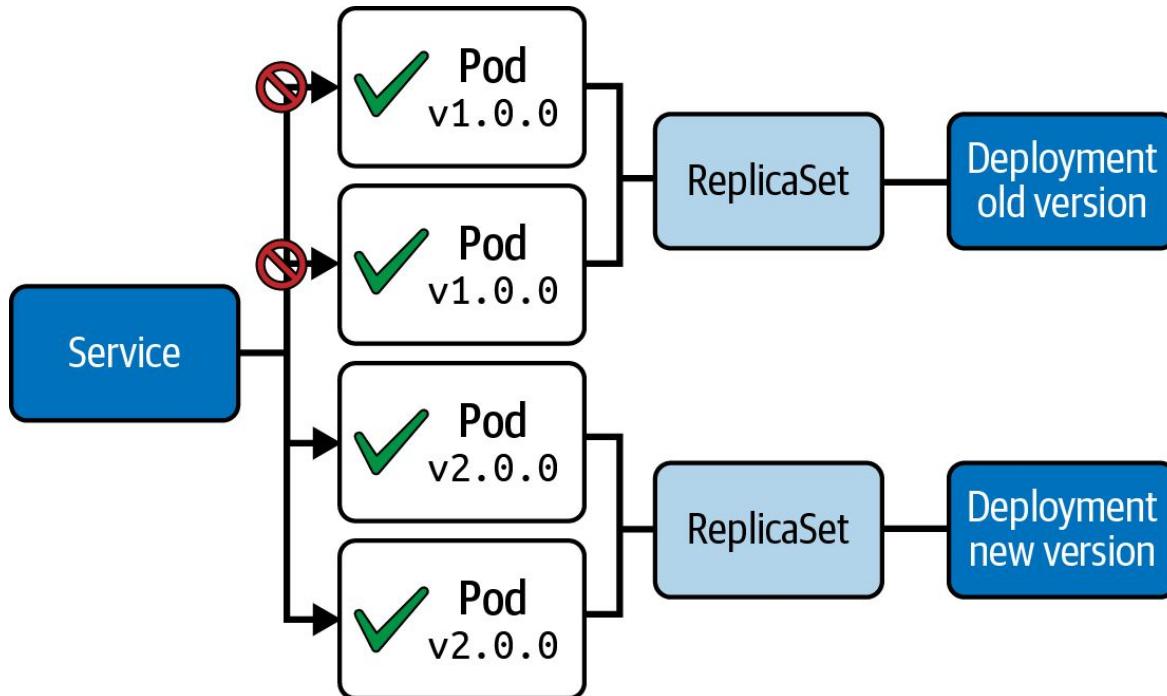
```
spec:  
  replicas: 3  
  strategy:  
    type: Recreate
```

← Set the strategy explicitly

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

# Blue/Green Deployment Strategy

Deployment object with new version is created alongside



# Blue/Green Deployment Strategy

Requires a duplicate set of objects

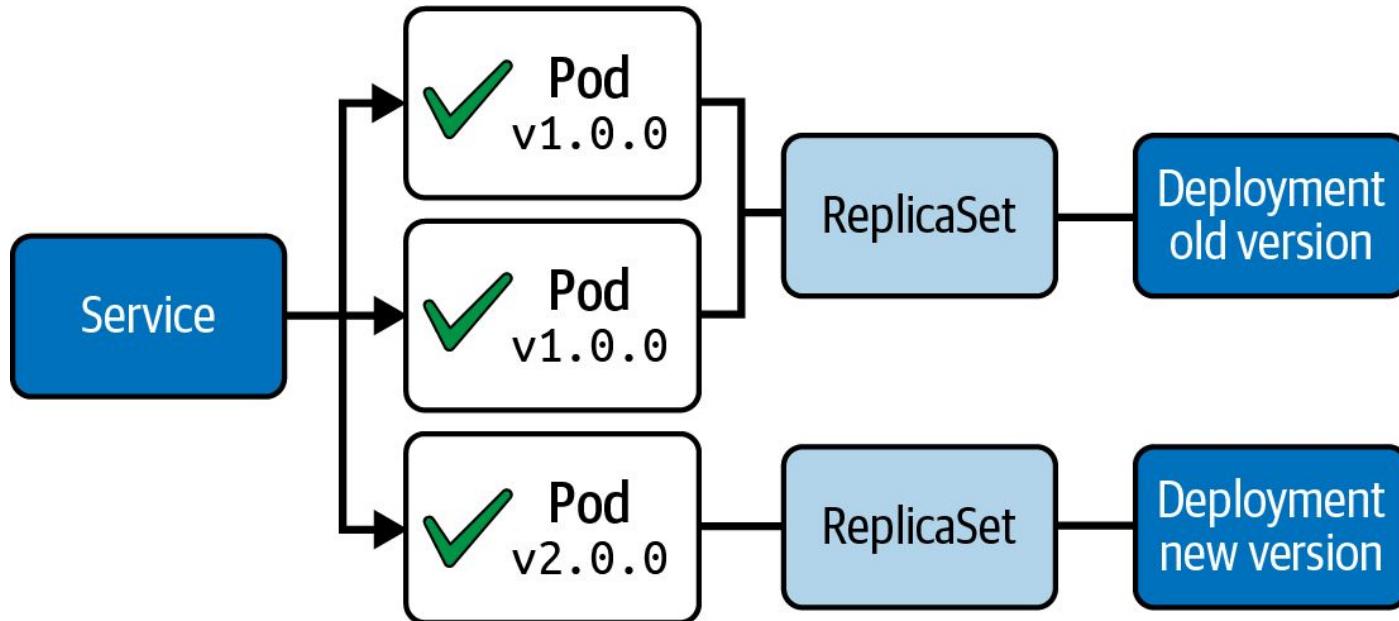
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blue
spec:
  selector:
    matchLabels:
      app: my-deploy
  version: 1.0.0
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green
spec:
  selector:
    matchLabels:
      app: my-deploy
  version: 2.0.0
```

- **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



# Canary Deployment Strategy

The new Deployment initially starts with a smaller number of replicas

```
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

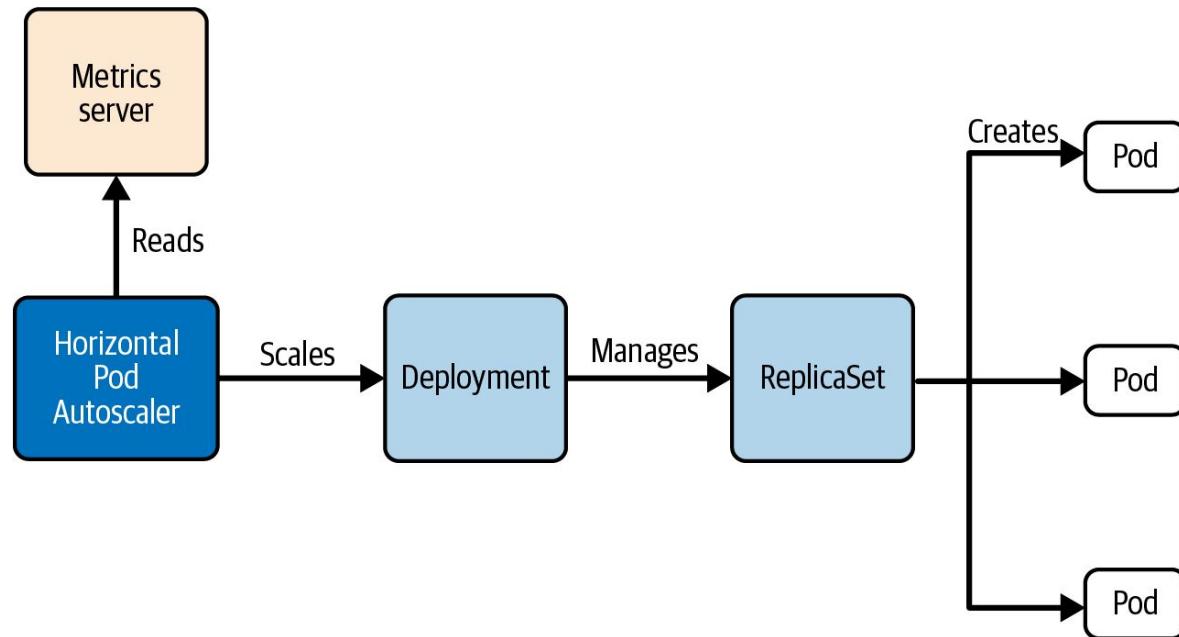
# Types of Autoscalers

Scaling 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. A Pod should also set the resource requests and limits.

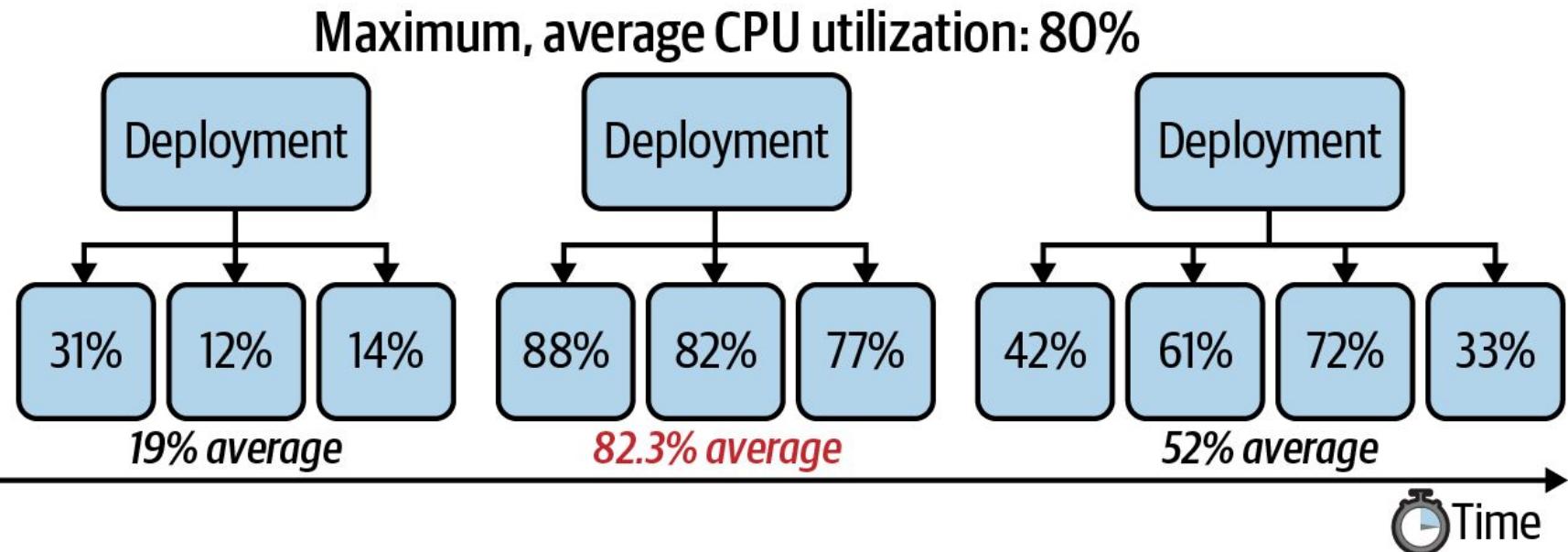
# Horizontal Pod Autoscaler (HPA)

Kubernetes primitive that reaches into metrics server



# Autoscaling a Deployment by CPU

"Auto-scale based on average CPU utilization across all replicas."



# Horizontal Pod Autoscaler YAML Manifest

Creates an error if label selector and template labels do not match

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: app-cache
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: app-cache
  minReplicas: 3
  maxReplicas: 5
  metrics:
    - type: Resource
      resource:
        name: cpu
      target:
        type: Utilization
        averageUtilization: 80
```

The scaling target,  
a Deployment

One or many resource types that  
the thresholds should apply to

# Inspecting the Horizontal Pod Autoscaler

Use scale command or change spec.replicas attribute in live object

```
$ kubectl get hpa
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
app-cache	Deployment/app-cache	199/500Mi, 0%/80%	3	5	3	2m14s



Renders current resource utilization  
and compares them with thresholds



## Exercise

Creating a Horizontal Pod  
Autoscaler for a  
Deployment



# Exam Essentials

What to focus on for the exam

- A Deployment provides replication and scaling capabilities for a set of Pods, also called replicas. It uses label selection to match with the Pod template.
- Practice how to manually scale the number of replicas and how to use a Horizontal Pod Autoscaler to scale automatically.
- Rolling out a new revision can be achieved by updating the attribute values of the Pod template. Rolling back to a previous version can be performed by using the `kubectl rollout undo` command.



# Resource Management

Resource requests and limits, resource quotas, limit ranges

# Managing Resources for Objects and Namespaces

It's best practice to make a 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.
- A LimitRange is a policy that constrains the resource allocations for a single object (e.g., for a Pod or PersistentVolumeClaim).

# 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.

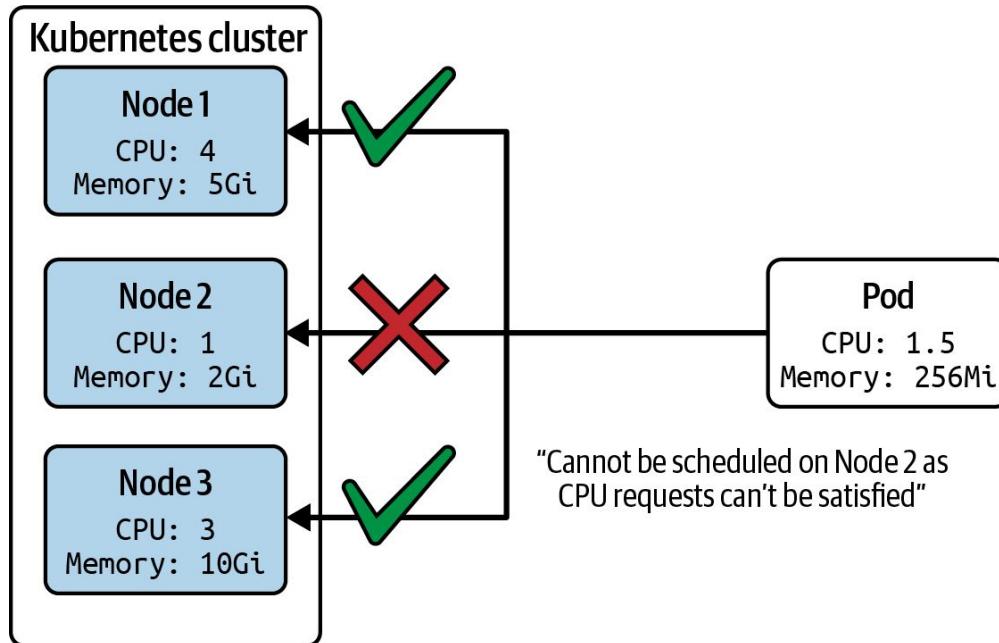
# Container Resource Requests

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



# Container Resource Requests Options

Configuration options and example values

YAML Attribute	Description	Example Value
<code>spec.containers[].resources.requests.cpu</code>	CPU resource type	500m (five hundred millicpu)
<code>spec.containers[].resources.requests.memory</code>	Memory resource type	64Mi (2^26 bytes)
<code>spec.containers[].resources.requests.hugepages-&lt;size&gt;</code>	Huge page resource type	hugepages-2Mi: 60Mi
<code>spec.containers[].resources.requests.ephemeral-storage</code>	Ephemeral storage resource type	4Gi

# Container Resource Requests YAML Manifest

Minimum amount of resources defined 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.

# Container Resource Limits Options

Configuration options and example values

YAML Attribute	Description	Example Value
<code>spec.containers[].resources.limits.cpu</code>	CPU resource type	500m (five hundred millicpu)
<code>spec.containers[].resources.limits.memory</code>	Memory resource type	64Mi (2^26 bytes)
<code>spec.containers[].resources.limits.hugepages-&lt;size&gt;</code>	Huge page resource type	hugepages-2Mi: 60Mi
<code>spec.containers[].resources.limits.ephemeral-storage</code>	Ephemeral storage resource type	4Gi

# Container 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%)    ...
```

# Container Resource Requests/Limits YAML Manifest

It is considered best practice to define requests and limits for every container

```
spec:  
  containers:  
    - name: business-app  
      resources:  
        requests:  
          memory: "256Mi"  
          cpu: "1"  
        limits:  
          memory: "512Mi"  
          cpu: "2"
```

## Exercise

Defining container  
resource requests and  
limits



# What is a ResourceQuota?

Defines resource constraints per namespace

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

# ResourceQuota YAML Manifest

Limits # of objects and aggregate container 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 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: 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

Renders 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 resource quota  
for a namespace



# What is a LimitRange?

Constrains or defaults the resource allocations for an object type

- Enforces minimum and maximum compute resources usage per Pod or container in a namespace.
- Enforces minimum and maximum storage request per PersistentVolumeClaim in a namespace.
- Enforces a ratio between request and limit for a resource in a namespace.
- Set default request/limit for compute resources in a namespace and automatically injects them to containers at runtime.

# LimitRange YAML Manifest

Sets defaults, minimum and maximum CPU allocation for containers

```
apiVersion: v1
kind: LimitRange
metadata:
  name: cpu-resource-constraint
spec:
  limits:
    - default:
        cpu: 500m
      defaultRequest:
        cpu: 500m
    max:
      cpu: "1"
    min:
      cpu: 100m
  type: Container
```

Defines default limits

Defines default requests

Minimum and maximum resource range

# Automatically Uses Container Defaults for Pod

Container doesn't provide any resource requests/limits

```
$ kubectl create -f pod.yaml
pod/rate-limiter created

$ kubectl describe pod rate-limiter
...
Containers:
  business-app:
    ...
    Limits:
      cpu: 500m
    Requests:
      cpu: 500m
    ...
  
```



# Rejected Pod Creation for Exceeded Limits

Requests a higher maximum for CPU resources than defined by LimitRange

```
$ kubectl create -f pod.yaml
Error from server (Forbidden): error when creating "pod.yaml": pods "rate-limiter" is forbidden: maximum cpu usage per Container is 1,
but limit is 2
```

## Exercise

Creating a Pod conforming  
with LimitRange in  
namespace



# Exam Essentials

What to focus on for the exam

- Learn how to define minimum and maximum resources for a container. Have an understanding of those resource definitions on Pod scheduling and runtime behavior.
- A resource quota that limit aggregate resource consumption or limit the quantity of objects (e.g., number of Pods or Secrets) that can be created. Experience the effects by instantiating object for the namespaces defined by the resource quota.
- A limit range constraints or defaults the resource allocations for an object type. Know how to list and inspect them to judge their effect on object creation.



# Pod Scheduling

Node selector, node affinity, taints and tolerations

# Assigning Pods to a Node

The scheduler uses a scoring system

- The scheduler cluster component is responsible for placing a Pod on a node best suited by following placement rules. The component runs periodically.
- Pods and their container can define requirements that help with determining the node. If the requirements cannot be met by any of the nodes, then the Pod stays unscheduled until the scheduler checks again.
- Feasible nodes that can fulfill the requirements are scored. The scheduler picks the node with the highest score and places the Pod on it.
- Scheduling decisions include resource requirements, affinity and anti-affinity specifications, and many more.

# Discovering the Scheduler Cluster Component

Runs in the `kube-system` namespace of a control plane node

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

# Determining the Node a Pod runs on

Definition and runtime effects

```
$ kubectl get nodes
NAME           STATUS   ROLES      AGE    VERSION
controlplane   Ready    control-plane   4m23s  v1.30.1
node01        Ready    <none>     4m     v1.30.1

$ kubectl get pods -o=wide
NAME    READY   STATUS    RESTARTS   AGE    IP          NODE    ...
app    1/1     Running   0          22h   10.0.0.102  node01  ...
...  
$ kubectl get pod app -o yaml | grep nodeName:
nodeName: node01
```

# Requirements for Pod Scheduling

Soft and hard requirements for scheduling a Pod on nodes

- The scheduler does a reasonably good job of assigning a Pod to a feasible node.
- Under certain conditions, you want to *restrict* which node a Pod can run on or just define a *preferred* selection criteria. This is usually expressed by label selection.

# Pod Scheduling Options

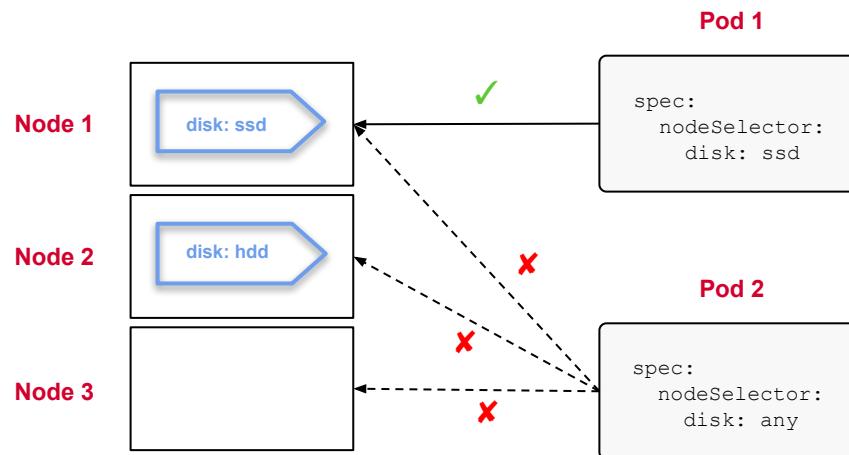
Concepts can be combined if needed

- **Node selector**: A hard requirement to determine which node the Pod needs to run on.
- **Node affinity and anti-affinity**: A more flexible requirement for defining hard or soft requirements for node selection.
- **Pod topology spread constraints**: Defines how to spread Pods across different topologies, e.g. regions and zones.

# Node Selector

Defines a hard requirement for scheduling a Pod on specific nodes

- Example use case: Ensuring that a Pod ends up on a node with an SSD Volume attached to it.
- Label one or many nodes with a label, and select the label from the Pod via `spec.nodeSelector`.



# Labeling of a Node

Definition and runtime effects

```
$ kubectl label nodes node01 disk=ssd
```

node/node01 labeled

```
$ kubectl get nodes --show-labels
```

NAME	STATUS	ROLES	AGE	VERSION	LABELS
controlplane	Ready	control-plane	4m23s	v1.30.1	...,
node01	Ready	<none>	4m	v1.30.1	..., disk=ssd

# Setting a Pod's Node Selector

Assignment of a single label key-value pair

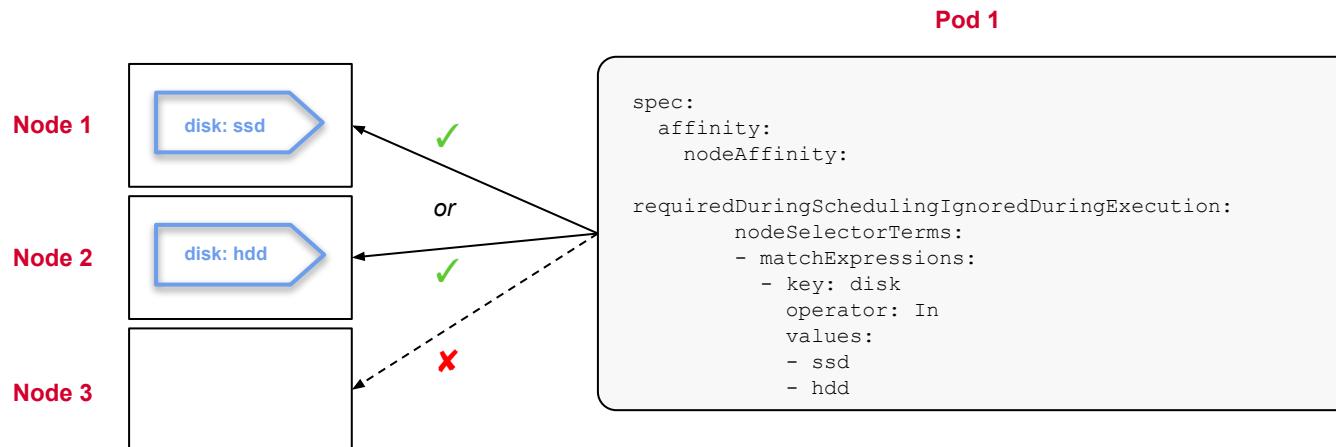
```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  nodeSelector:
    disk: ssd
  containers:
  - name: nginx
    image: nginx:1.27.1
```

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

# Node Affinity

Defines flexible, expressive requirements for scheduling a Pod on specific nodes

- Example use case: Ensuring that a Pod ends up on a node with a SSD or HDD Volume attached to it.
- Label one or many nodes with a label, and select the label from the Pod via `spec.affinity.nodeAffinity`



# Setting a Pod's Node Affinity

Assignment of one or more expressions

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: disk
                operator: In
                values:
                  - ssd
                  - hdd
  containers:
    - name: nginx
      image: nginx:1.27.1
```

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

# Node Affinity Types

Currently two types available, 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

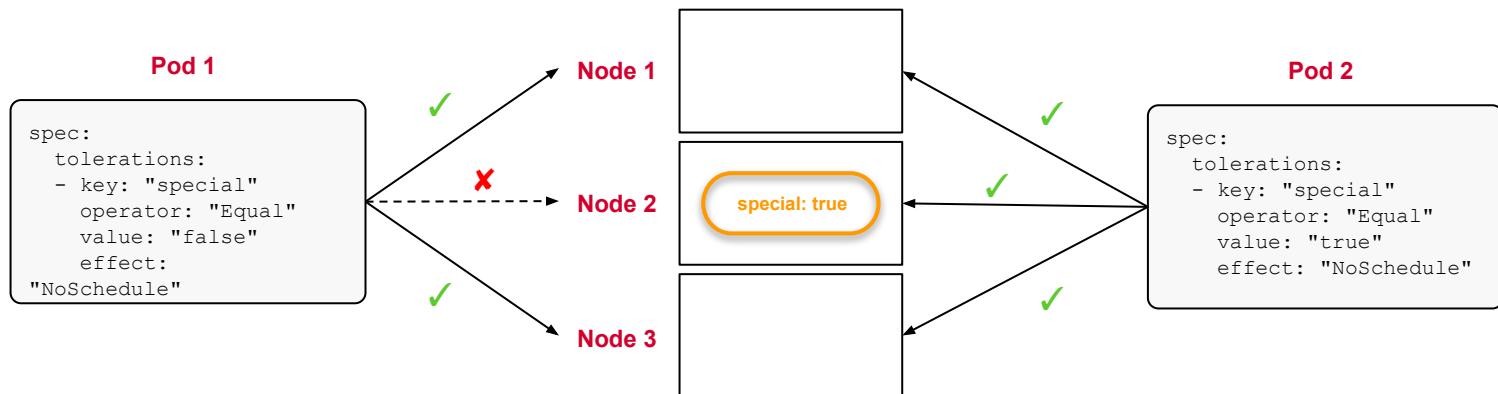
Scheduling a Pod on a  
node using node selector  
and node affinity



# Taints and Tolerations

Allows a node to repel a set of Pods (opposite of node affinity)

- Example use case: Cluster with specialized hardware should not schedule Pods on nodes that don't require more costly runtime environments.
- Add a taint to one or many nodes, and allow scheduling a Pod by adding toleration(s) via `spec.tolerations`.



# Adding a Taint to a Node

A taint follows the format `key=value:effect`

```
$ kubectl taint nodes node01 special=true:NoSchedule  
node/node01 tainted  
  
$ kubectl get nodes node01 -o yaml | grep -C 3 taints:  
...  
spec:  
  taints:  
    - effect: NoSchedule  
      key: special  
      value: "true"
```

# 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(s)

Every Pod can define one or more tolerations even with the same key

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  tolerations:
    - key: "special"
      operator: "Equal"
      value: "true"
      effect: "NoSchedule"
  containers:
    - name: nginx
      image: nginx:1.27.1
```

Available operators:  
Equal, Exists

## Exercise

Configuring a node to only accept specific Pods



# Exam Essentials

What to focus on for the exam

- Kubernetes offers a set of mechanism to control which node a Pod can be scheduled on. For exam, you will need to understand the node selector, node affinity, and taints and tolerations.
- The node selector is simplest form of directly assign a Pod to a specific node by label selection. Node affinity enhances the same concept by allowing more flexible rules.
- Taints and tolerations implement the opposite effect of node affinity. A node can repeal a Pod from being scheduled on it if the Pod doesn't provide a matching toleration.



# ConfigMaps and Secrets

Defining and consuming configuration data

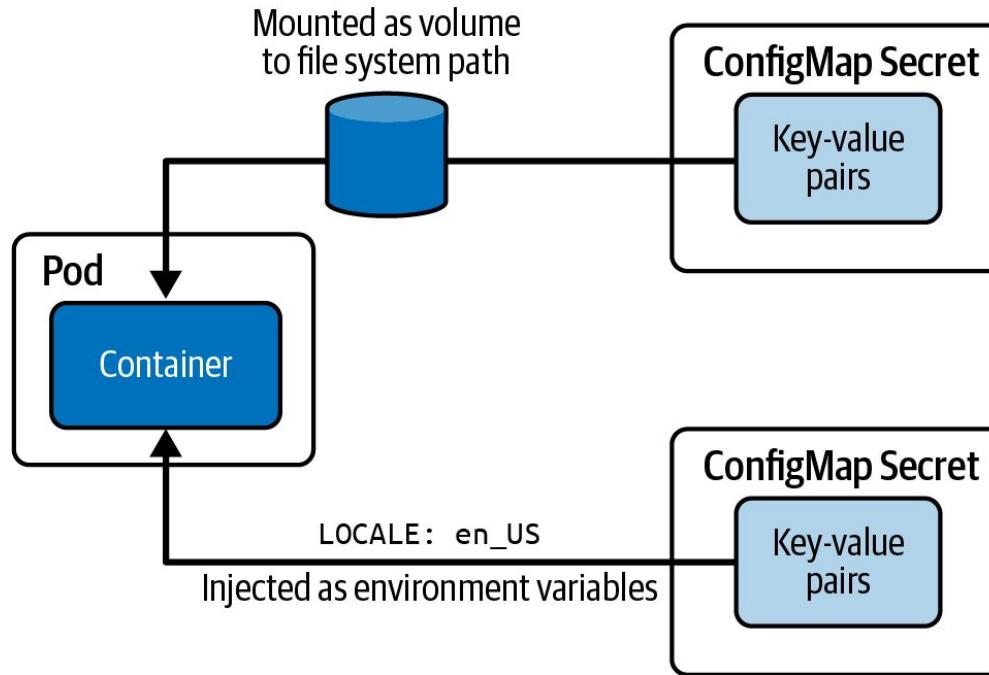
# Defining Configuration Data

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!**
- ConfigMap and Secret objects are stored in etcd in unencrypted form by default. Encryption of data in etcd is configurable.

# Consuming Configuration Data

Applicable to ConfigMap and Secret



# Creating a ConfigMap with Imperative Approach

Key-value pairs can be parsed from 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
```

# Source Options for Data Parsed by a ConfigMap

Configuration options and example values

Option	Description	Example Value
--from-literal	Literal values, which are key-value pairs as plain text.	--from-literal=locale=en_US
--from-env-file	A file that contains key-value pairs and expects them to be environment variables.	--from-env-file=config.env
--from-file	Huge page resource type.	--from-file=app-config.json
--from-file	A directory with one or many files.	--from-file=config-dir

# Creating a ConfigMap from Literal Values

Imperative creation by pointing to plain-text key-value pairs

```
$ kubectl create configmap db-config  
--from-literal=DB_HOST=mysql-service  
--from-literal=DB_USER=backend  
configmap/db-config created
```

Parameter can  
be repeated  
multiple times



# ConfigMap YAML Manifest

Key-value pair definition under data attribute

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  DB_HOST: mysql-service
  DB_USER: backend
```

Follows typical  
naming conventions  
for environment  
variables



# Consuming a ConfigMap as Environment Variables

Controlling runtime behavior with the help of simple values

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - image: bmuschko/web-app:1.0.1
      name: backend
      envFrom:
        - configMapRef:
            name: db-config
```

```
$ kubectl exec -it backend -- env
DB_HOST=mysql-service
DB_USER=backend
```

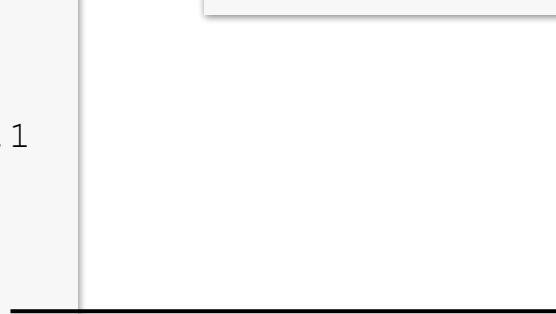


# Reassigning Environment Variable Keys

Keys can be renamed or reformatted to make them suitable for consumption

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - image: bmuschko/web-app:1.0.1
      name: backend
      env:
        - name: DATABASE_URL
          valueFrom:
            configMapKeyRef:
              name: backend-config
              key: database_url
```

```
$ kubectl exec -it backend -- env
DATABASE_URL=jdbc:postgresql://localhost/test
```



# Creating a ConfigMap from a JSON file

Imperative creation by pointing to file

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

db.json

```
{  
  "db": {  
    "host": "mysql-service",  
    "user": "backend"  
  }  
}
```

# ConfigMap YAML Manifest

Key-value pair definition under data attribute

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  db.json: |
    {
      "db": {
        "host": "mysql-service",
        "user": "backend"
      }
    }
```

The file name  
becomes the key, the  
contents of the file  
become the value

# Consuming a ConfigMap as a Volume

A good choice for processing a machine-readable configuration file

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - name: backend
      image:
        bmuschko/web-app:1.0.1
      volumeMounts:
        - name: db-config-volume
          mountPath: /etc/config
  volumes:
    - name: db-config-volume
      configMap:
        name: db-config
```



```
$ kubectl exec -it backend -- /bin/sh
# ls -l /etc/config
db.json
# cat /etc/config/db.json
{
  "db": {
    "host": "mysql-service",
    "user": "backend"
  }
}
```



# Consuming Changed ConfigMap Data

Containers will not refresh consumed data upon a change

- The key-value pairs of a ConfigMap can be changed for a live object.
- Changed ConfigMap key-value pairs consumed by containers will not be reloaded automatically for an already-running Pod.
- The application code needs to implement logic to either reload the configuration data periodically or on-demand.

## Exercise

Creating and consuming a  
ConfigMap





# Creating a Secret with Imperative Approach

Parsed values are base64-encoded automatically

```
$ kubectl create secret generic db-creds --from-literal=pwd=s3cre!
secret/db-creds created

$ kubectl create secret docker-registry my-secret <
  --from-file=.dockerconfigjson=~/docker/config.json
secret/my-secret created

$ kubectl create secret tls accounting-secret --cert=accounting.crt <
  --key=accounting.key
secret/accounting-secret created
```

# Imperative CLI Options for Creating a Secret

You will need to provide one of the options

Option	Description	Type
generic	Creates a secret from a file, directory, or literal value.	Opaque
docker-registry	Creates a secret for use with a Docker registry.	kubernetes.io/dockercfg
tls	Creates a TLS secret.	kubernetes.io/tls

# Source Options for Data Parsed by a Generic Secret

Configuration options and example values

Option	Description	Example Value
--from-literal	Literal values, which are key-value pairs as plain text.	--from-literal=password=secret
--from-env-file	A file that contains key-value pairs and expects them to be environment variables.	--from-env-file=config.env
--from-file	Huge page resource type.	--from-file=id_rsa=~/ssh/id_rsa
--from-file	A directory with one or many files.	--from-file=config-dir

# Specialized Secret Types

Can be set with `--type` CLI option, or the `type` attribute in manifest ([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
bootstrap.kubernetes.io/token	Node bootstrap token data

# Creating a Secret from Literal Values

Imperative command will base64-encode values automatically

```
$ kubectl create secret<br>generic<br>db-creds<br>--from-literal=pwd=s3cre!<br>secret/db-creds created
```

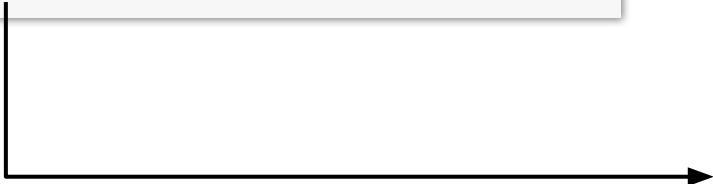
← Plain-text value

# Secret YAML Manifest

Assigned values need to be provided in base64-encoded form

```
$ echo -n 's3cre!' | base64  
czNjcmUh
```

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



# Consuming a Secret as Environment Variables

Accessed values are base64-decoded

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - image: bmuschko/web-app:1.0.1
      name: backend
      envFrom:
        - secretRef:
            name: mysecret
```

```
$ kubectl exec -it backend -- env
pwd=s3cre!
```



# Creating a Secret from a SSH Private Key File

Imperative creation by pointing to file

```
$ kubectl create secret generic secret-ssh-auth  
  --type=kubernetes.io/ssh-auth --from-file=ssh-privatekey  
secret/secret-ssh-auth created
```

ssh-privatekey

```
-----BEGIN RSA PRIVATE KEY-----  
Proc-Type: 4,ENCRYPTED  
DEK-Info:  
AES-128-CBC,8734C9153079F2E8497C8075289EBBF1  
...  
-----END RSA PRIVATE KEY-----
```

# Consuming a Secret as a Volume

A good choice for processing a machine-readable configuration file

```
apiVersion: v1
kind: Pod
metadata:
  name: backend
spec:
  containers:
    - name: backend
      image:
        bmuschnko/web-app:1.0.1
      volumeMounts:
        - name: ssh-volume
          mountPath: /var/app
          readOnly: true
  volumes:
    - name: ssh-volume
      secret:
        secretName:
secret-ssh-auth
```



```
$ kubectl exec -it backend -- /bin/sh
# ls -l /var/app
ssh-privatekey
# cat /var/app/ssh-privatekey
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info:
AES-128-CBC,8734C9153079F2E8497C8075289E
BBF1
...
-----END RSA PRIVATE KEY-----
```

# Plain-Text Secret Values

The `stringData` attribute allows for plain-text values

`secret.yaml`

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

```
kubectl create -f
secret.yaml
```

*Live object*

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-basic-auth
type: kubernetes.io/basic-auth
stringData:
  username: YWRtaW4=
  password: dDBwLVNlY3JldA==
```



# Secrets Are Not Really Secret

Techniques that help with making Secrets more secure

- Secret objects are stored in etcd in unencrypted form by default. Encryption of data in etcd is [configurable](#).
- Define RBAC permissions to only allow developers to create, view, and modify Secret objects in dedicated namespaces. An even stricter policy could only allow administrators to manage Secrets.
- Use Kubernetes-external, third-party Secrets services for managing sensitive data, e.g. AWS Secrets Manager or HashiCorp Vault. You can consume the data with the help of the [External Secrets Operator](#).



## Exercise

Creating and consuming a  
Secret



# Exam Essentials

What to focus on for the exam

- ConfigMaps store plain-text key-value pairs. They are a good fit for simple strings, e.g. connection URLs, and theme names.
- Secrets are meant to represent sensitive data, however, they are not encrypted and therefore not “secret”. The values of the key-value pairs are only base64-encoded.
- ConfigMaps and Secrets can be consumed by Pods as environment variables or Volumes. Choose the appropriate method based on your application’s implementation. Practice both approaches.

# Storage





# Topics We'll Cover

## Volumes Overview

- Types of Volume
- Ephemeral versus persistent Volumes

## Ephemeral Volumes

- Use cases
- Mounting a Volume to a Pod
- Configuration options

## Persistent Volumes

- Static versus dynamic provisioning
- PersistentVolume, PersistentVolumeClaim  
StorageClass
- Configuration options



# Volumes

Ephemeral and persistent storage for Pods



# Managing Data in Containers

Container filesystems are isolated

- Each container manages files in a temporary file system.
- Multiple containers running in the same Pod cannot exchange data as their temporary file system is isolated.
- Data written to the file system is lost when the container is restarted.



# What is a Volume?

A directory, possibly with some data in it accessible to the containers in a Pod

- **Ephemeral Volumes** exist for the lifespan of a Pod. They are useful if you want to share data between multiple containers running in the Pod.
- **Persistent Volumes** preserve data beyond the lifespan of a Pod. They are a good option for applications that require data to exist longer, e.g. in the form of storage for a database-driven application.
- Define the Volume type with `spec.volumes[]` and then reference it in a container with `spec.containers[] .volume.volumeMounts`

# Volume Types

Determines the medium that backs the Volume and its runtime behavior

Type	Description
emptyDir	Empty directory in Pod with read/write access. Only persisted for the lifespan of a Pod.
hostPath	File or directory from the host node's filesystem.
configMap, secret	Provides a way to inject configuration data.
nfs	An existing NFS (Network File System) share. Preserves data after Pod restart.
persistentVolumeClaim	Claims a Persistent Volume.

# Defining 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

You can shell into container and navigate to mount path

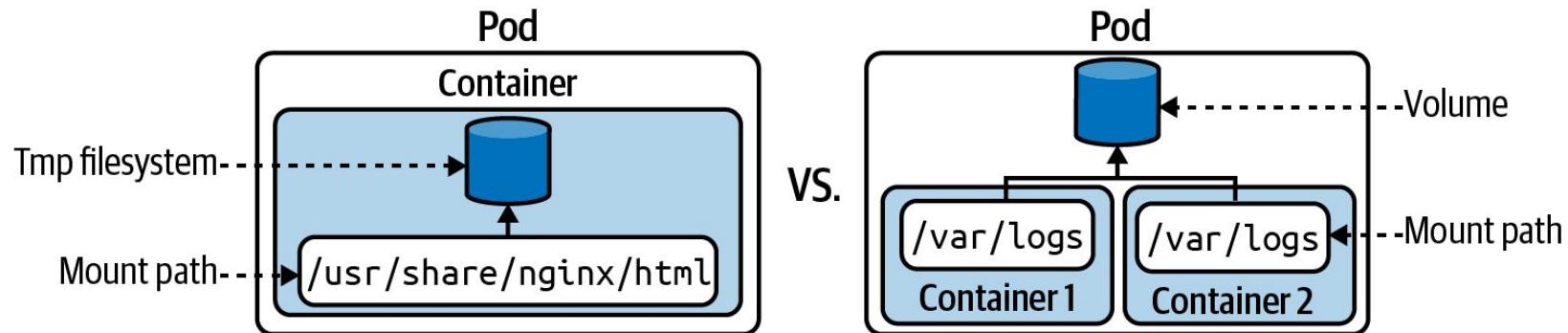
```
$ kubectl create -f pod-with-vol.yaml
pod/my-container created

$ 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

# Ephemeral Volume

Useful for sharing data between containers or for caching data



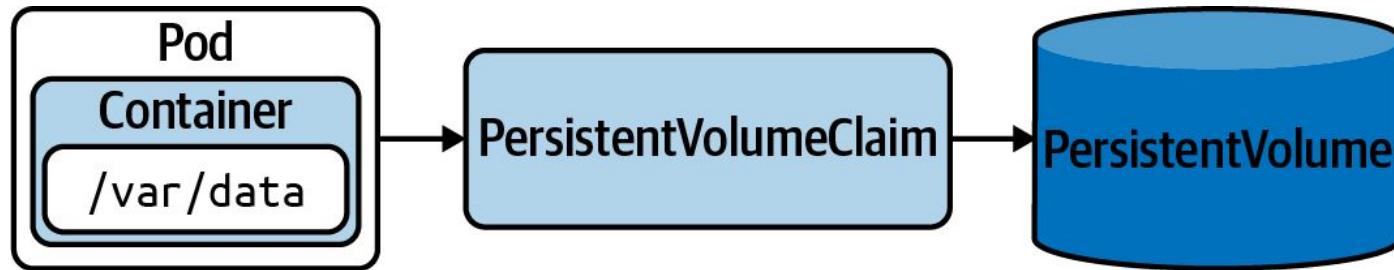
## Exercise

Creating and using an  
ephemeral Volume



# Persistent Volume

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



# Static versus 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

Defines read/write capabilities of Volume

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

# Reclaim Policy

What should happen to PersistentVolume when Claim is deleted?

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

# Creating a PersistentVolume

Summarized most important configuration and status

```
$ 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 PersistentVolumeClaim

Will bind to 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 a PersistentVolumeClaim

Binds to PersistentVolume if requirements can be fulfilled

```
$ 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          
```



Binding to the PersistentVolume  
object was successful

# Mounting a PersistentVolumeClaim 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

## Exercise

Creating and using a  
PersistentVolume using  
static provisioning



# Assigning a StorageClass for Dynamic Provisioning

Creates PersistentVolume object automatically via storage class

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

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

# Dynamic Creation of PersistentVolume Object

Adds hash to the name of the object

```
$ kubectl get pv,pvc
```

```
NAME ...
```

```
persistentvolume/db-pvc-b820b919-f7f7-4c74-9212-ef259d421734
```

STORAGECLASS

aws-ebs

```
NAME ...
```

```
persistentvolumeclaim/db-pvc
```

STORAGECLASS

aws-ebs



Created by storage  
class provisioner

## Exercise

Creating and using a  
PersistentVolume using  
dynamic provisioning



# Exam Essentials

What to focus on for the exam

- Understand the need and use cases for ephemeral and persistent volumes.
- Practice the use of the most common Volume types, e.g. `emptyDir`, `hostPath`.
- Internalize the mechanics for creating a persistent volume through static and dynamic binding. The persistent volume does not have to be created manually when using a storage class from the persistent volume claim.

# Servicing and Networking





# Topics We'll Cover

## Services

- Service types and their configuration
- Communication with a service

## Ingresses

- How is it different from a service?
- Traffic routing
- Ingress controller

## Gateway API

- How does the Gateway API compare to the Ingress?
- Gateway API CRDs
- A sample Gateway implementation

## Network Policies

- Defining firewall rules for Pod-to-Pod communication
- Network policy controller



# Services

Providing a stable network endpoint to Pods



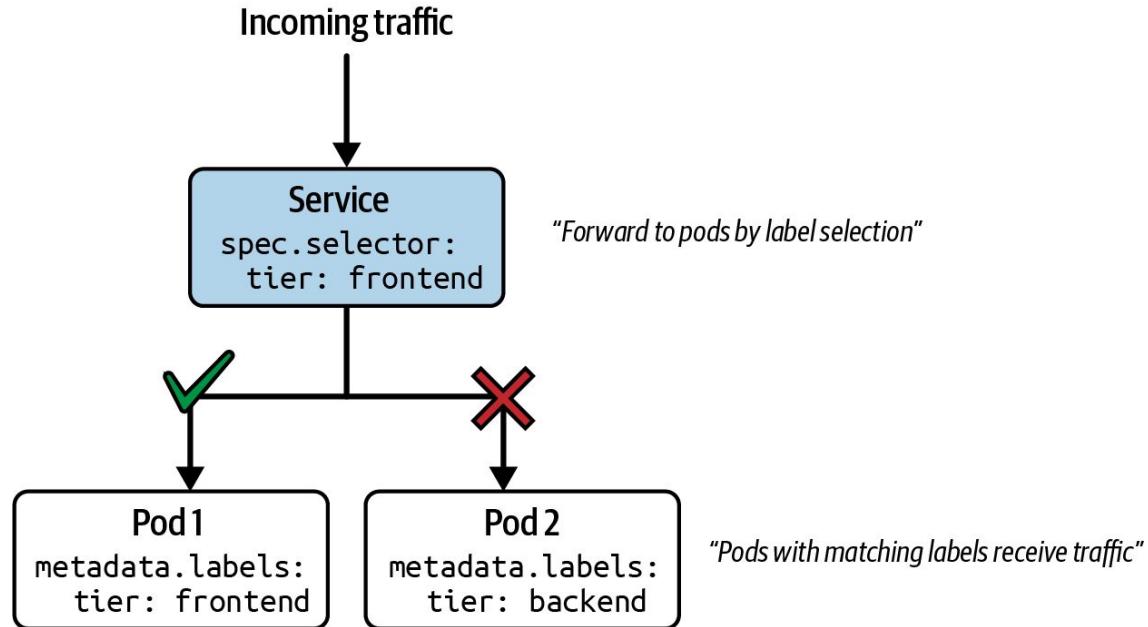
# What is a Service?

Stable endpoint for routing traffic to a set of Pods

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

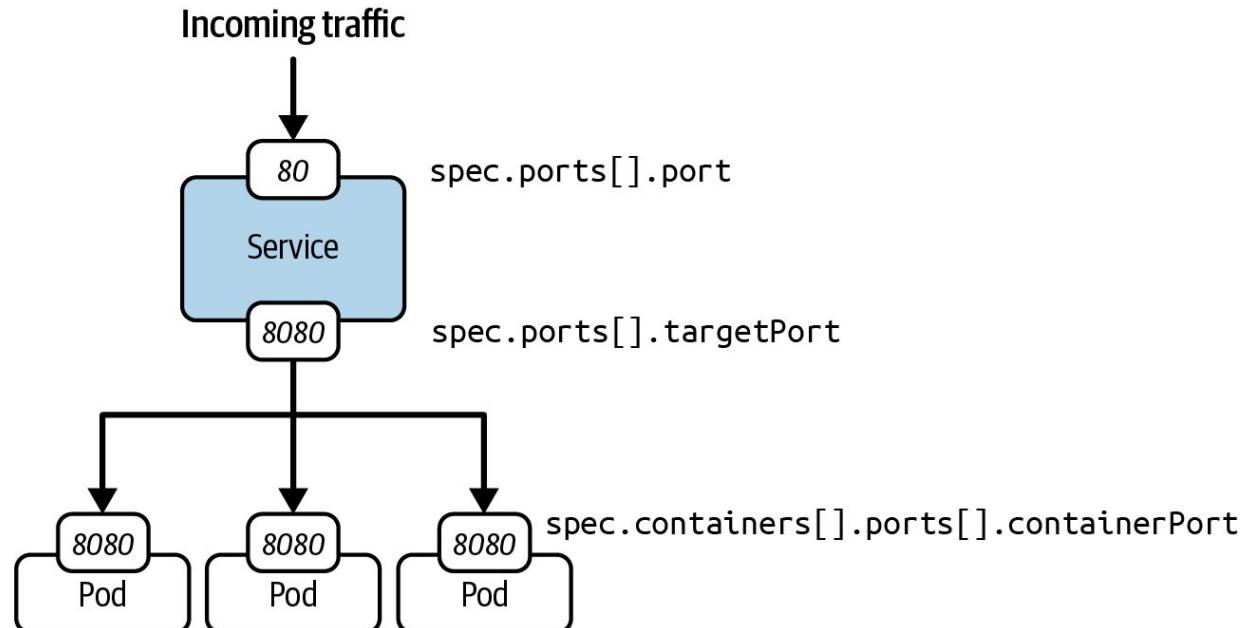
# Request Routing

"How does a Service decide which Pod to forward the request to?"



# Port Mapping

"How to map the Service port to the container port in a Pod?"



# Creating a Service with Imperative Approach

Needs to provide the service type and ports

```
$ kubectl run echoserver  
  --image=k8s.gcr.io/echoserver:1.10  
  --port=8080 ← Exposed container port  
pod/echoserver created
```

```
$ kubectl create  
  service clusterip  
  echoserver  
  --tcp=80:8080 ← Incoming and outgoing port  
service/echoserver created
```

```
$ kubectl run echoserver  
  --image=k8s.gcr.io/echoserver:1.10  
  --port=8080  
  --expose ← Shortcut for creating a Pod + Service  
service/echoserver created  
pod/echoserver created
```

# Service YAML Manifest

Needs to provide the service type and ports

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
spec:
  selector:
    app: echoserver
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
```

← Label selection for Pods

← Mapping of incoming to outgoing port

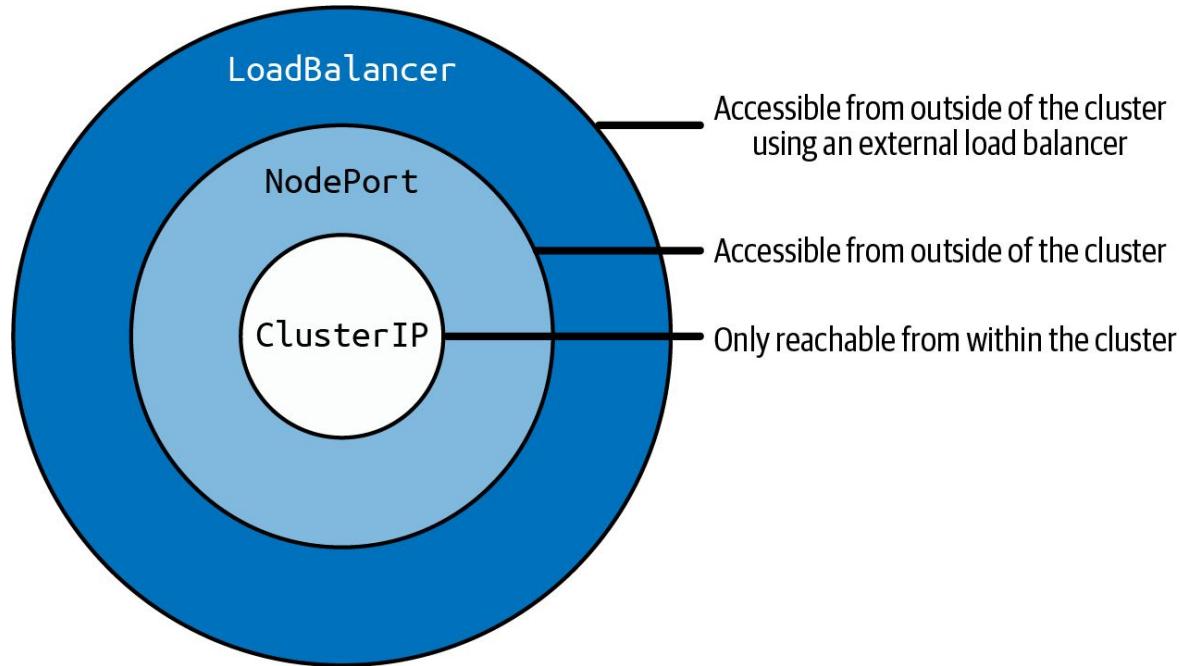
# Service Types

The following table shows a limited list of Service types

Type	Description
ClusterIP	Exposes the service on a cluster-internal IP. Only reachable from Pods within the cluster. Default if value for <code>spec.type</code> has not been assigned in manifest.
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.

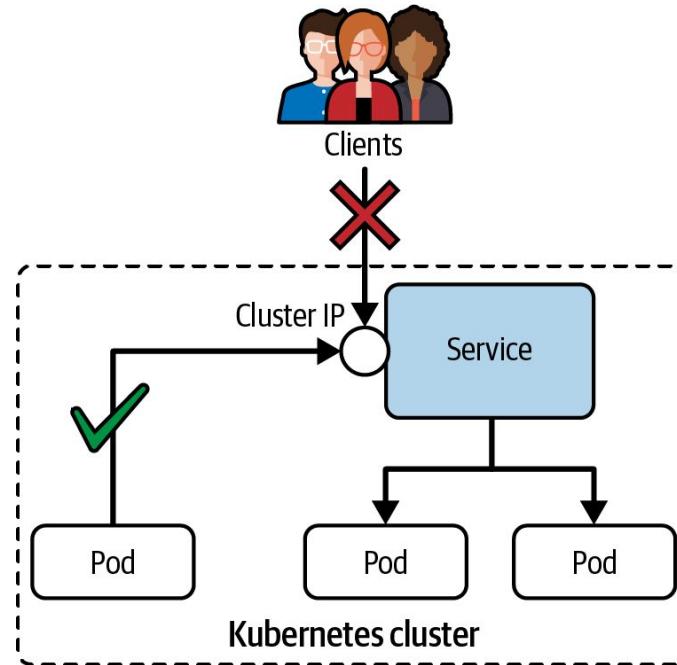
# Service Type Inheritance

Services build on top of each other



# ClusterIP Service Type

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



# ClusterIP Service YAML Manifest at Runtime

ClusterIP service type has been assigned if not already provided

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
spec:
  type: ClusterIP
  clusterIP: 10.96.254.0
  selector:
    app: echoserver
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
```

# 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>        8080/TCP  8s
```

```
$ kubectl run tmp --image=busybox --restart=Never -it --rm <
-- wget 10.96.254.0:8080
Connecting to 10.96.254.0:8080 (10.96.254.0:8080)
...

```

# Discovering the Service by Environment Variables

Application code running in container can get access to Service endpoint

```
$ kubectl exec -it echoserver -- env  
ECHOSERVER_SERVICE_HOST=10.96.254.0  
ECHOSERVER_SERVICE_PORT=8080  
...
```

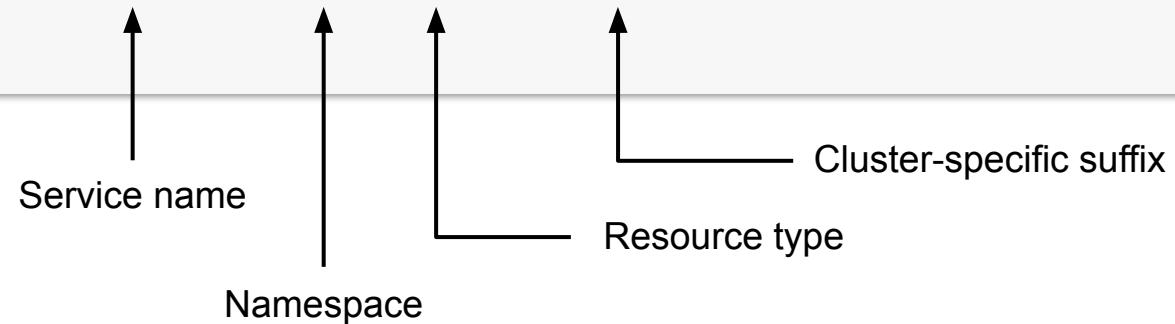
# Discovering the Service by DNS Lookup

Kubernetes' DNS service maps cluster IP address to Service name

```
$ kubectl run tmp --image=busybox --restart=Never -it --rm <  
-- wget echoserver.default.svc.cluster.local:8080
```

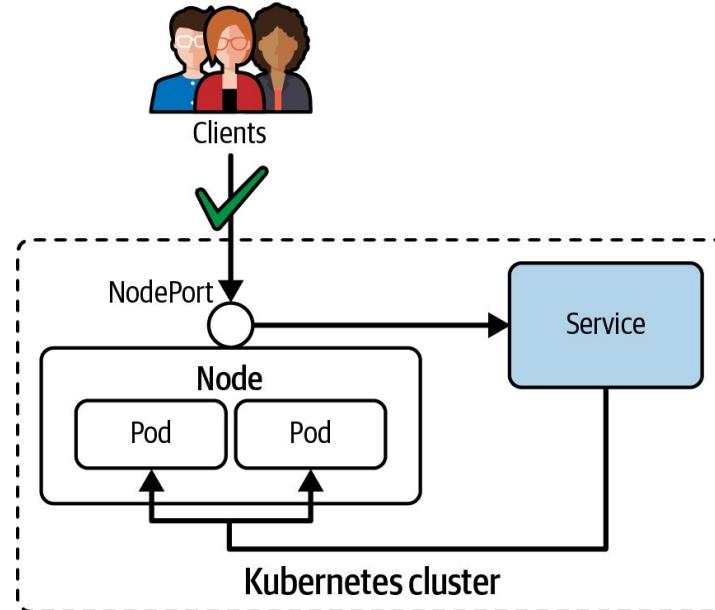
Connecting to echoserver.default.svc.cluster.local:8080 (10.96.254.0:8080)

...



# NodePort Service Type

Can be resolved from outside of the Kubernetes cluster



# NodePort Service YAML Manifest at Runtime

Service type is NodePort, node port has been assigned

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
spec:
  type: NodePort
  clusterIP: 10.96.254.0
  selector:
    app: echoserver
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30158
    protocol: TCP
```

# 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>	8080:30158/TCP	8s

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

192.168.64.15

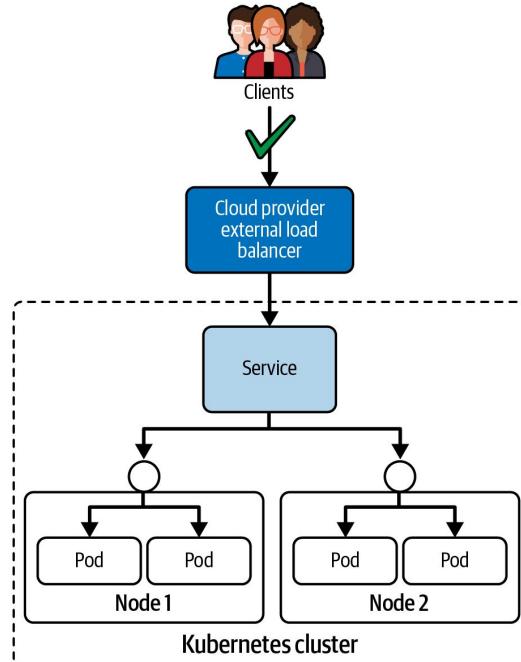
```
$ wget 192.168.64.15:30158
```

Connecting to 192.168.64.15:30158... connected.

...

# LoadBalancer Service Type

Routes traffic from existing, external load balancer to Service



# LoadBalancer Service YAML Manifest at Runtime

Service type is LoadBalancer, external IP address has been assigned

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
spec:
  type: LoadBalancer
  clusterIP: 10.96.254.0
  loadBalancerIP: 10.109.76.157
  selector:
    app: echoserver
  ports:
  - port: 80
    targetPort: 8080
    nodePort: 30158
    protocol: TCP
```



# Accessing a LoadBalancer Service

Use the combination of external 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	8080:30158/TCP	5s

```
$ wget 10.109.76.157:8080
```

Connecting to 10.109.76.157:8080... connected.

...



# Exercise

Routing traffic to Pods  
from inside and outside of  
a cluster



# Exam Essentials

What to focus on for the exam

- Pod-to-Pod communication should not be performed using a Pod's virtual IP address. A restart of the Pod will lease a new virtual IP address.
- A Service offers a stable network interface for implementing microservice architectures within the cluster and exposing network access to outside consumers.
- Have an understanding of the different service types relevant to the exam: ClusterIP, NodePort, and LoadBalancer. Know their network accessibility to clients from within and outside of the cluster. Practice how to call them.



# Ingress

Providing an external network entrypoint to the cluster



# What is an Ingress?

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

- It's not a specific Service type, nor should it be confused with the Service type LoadBalancer.
- Defines rules for mapping a URL context path to one or many Service objects.
- TLS termination (support for HTTPS) needs to be configured explicitly. By default, only HTTP is supported.

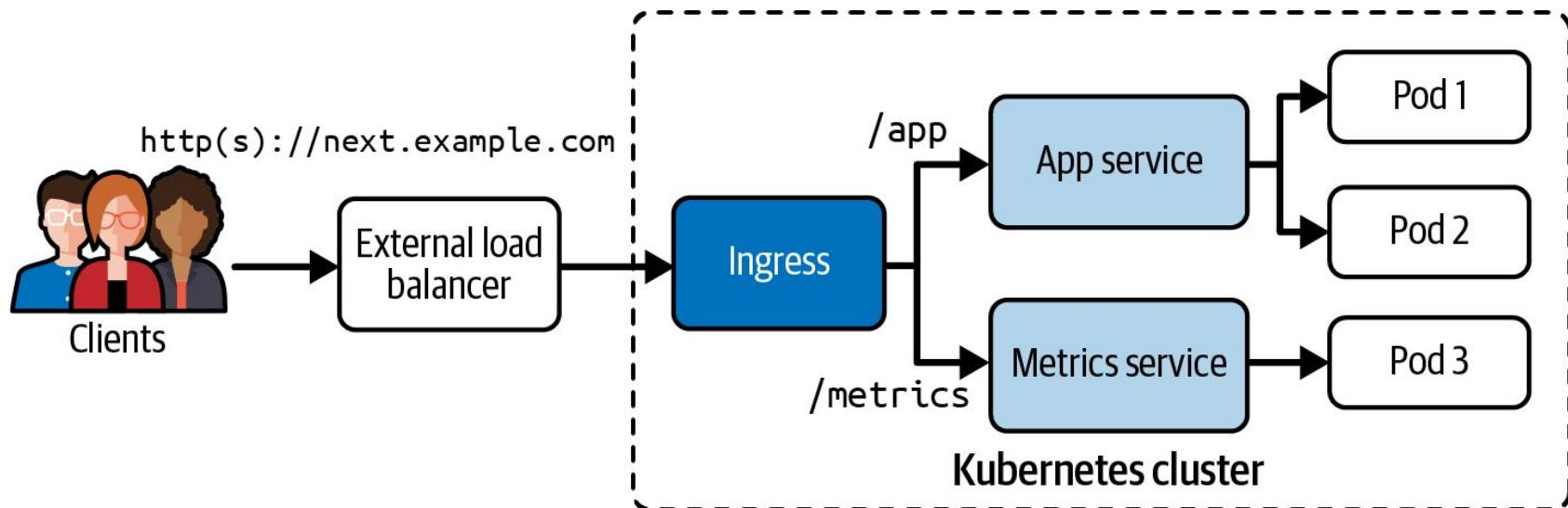
# What is an Ingress Controller?

Evaluating Ingress rules

- An Ingress cannot work without an [Ingress controller](#). The Ingress controller evaluates the collection of rules defined by an Ingress that determine traffic routing.
- Multiple Ingress controller can be deployed to a cluster. An Ingress can be configured to pick a specific one via `spec.ingressClassName`.

# Ingress Traffic Routing

The context path is mapped to a backend (Service name and port)





# Creating an Ingress with Imperative Approach

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

Allows for defining multiple rules that map to backend

```
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
```

The host definition is  
optional

# Ingress Rules

Traffic routing is controlled by rules defined on Ingress resource

Type	Example	Description
An optional host	mycompany.abc.com	If a host is provided, then the rules apply to that host. If no host is defined, then all inbound HTTP(S) traffic is handled.
A list of paths	/corellian/api	Incoming traffic must match the host and path to correctly forward the traffic to a Service.
The backend	corellian:8080	A combination of Service name and port.

# Path Types

Incoming requests match based on assigned type in rule

Path Type	Rule	Incoming Request
Exact	/corellian/api	Matches /corellian/api but does not match /corellian/test or /corellian/api/.
Prefix	/corellian/api	Matches /corellian/api and /corellian/api/ but does not match /corellian/test.

# Listing and Describing Ingresses

Renders hosts, IP addresses, and ports

```
$ kubectl get ingress
NAME      CLASS      HOSTS          ADDRESS      PORTS      AGE
corellian <none>    star-alliance.com  192.168.64.15  80        10m

$ kubectl describe ingress corellian
Name:            corellian
Namespace:       default
Address:         192.168.64.15
Default backend: default-http-backend:80 (<error: \
                                         endpoints "default-http-backend" not found>
Rules:
  Host           Path  Backends
  ----          ----  -----
  star-alliance.com
                           /corellian/api   corellian:8080 (172.17.0.5:8080)
Annotations:        <none>
Events:           <none>
```



# Configuring DNS for an Ingress

Making an Ingress convenient to use by end users

- To resolve the Ingress, you'll need to configure DNS entries to the external address.
- You'll need to either configure an A record, or a CNAME record. The [ExternalDNS](#) cluster add-on can help with managing those DNS records for you.
- If you have no domain or are using a local Kubernetes solution, you can add the mapping between IP address and domain name to `/etc/hosts`.

# Accessing an Ingress

Use host name, port, and context path

```
$ wget star-alliance.com/corellian/api --timeout=5 --tries=1
--2021-11-30 19:34:57-- http://star-alliance.com/corellian/api
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.
HTTP request sent, awaiting response... 200 OK
...
$ wget star-alliance.com/corellian/api/ --timeout=5 --tries=1
--2021-11-30 15:36:26-- http://star-alliance.com/corellian/api/
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.
HTTP request sent, awaiting response... 404 Not Found
2021-11-30 15:36:26 ERROR 404: Not Found.
```

# Exercise

Defining and using an  
Ingress





# Exam Essentials

What to focus on for the exam

- An Ingress routes HTTP(S) traffic toward one or many Services depending on the context path. Know how to configure context path matching for an Ingress.
- An Ingress doesn't work without an Ingress controller.
- Configuring TLS termination for an Ingress is not required for the exam.



# Gateway API

API specification defining a standard way to configure and manage application traffic routing

# Why is the Ingress Primitive not Sufficient?

More complex requirements are harder to implement

- The Ingress API is the standard Kubernetes way to configure external HTTP/HTTPS load balancing for Services, however, it has limitations.
- The Ingress API supports TLS termination and simple content-based request routing of HTTP traffic. Real world use cases call for more advanced features.
- Enhancing the API model of the Ingress wouldn't allow those features easily.

# Flaws of the Ingress Primitive

Potential requirements to ingress network traffic

- **Ingress controller-specific extensibility:** Advanced features like traffic splitting, rating limiting, request/response manipulation are provided by non-portable annotations for specific Ingress implementations.
- **Insufficient permission model:** The Ingress API is not well suited for multi-tenant environments that call for a strong permission model.

# What is the Gateway API?

Successor to the Ingress API

- Gateway API represents a superset of Ingress functionality, enabling more advanced use cases.
- Provides a unified and standardized API for managing traffic into and out of a Kubernetes cluster instead of individual Ingress implementations.



# Benefits of the Gateway API

A unified API for routing options

- Product-specific annotations are not needed anymore to configure routing options. The Gateway API offers a flexible way to incorporate similar features.
- Effectively, the Gateway API is a universal specification supported by multiple implementations.

# Gateway API Resources

The API offers multiple Custom Resource Definitions

- **Gateway:** Defines an instance of traffic handling infrastructure, such as cloud load balancer.
- **GatewayClass:** Each Gateway is associated with a GatewayClass, which describes the actual kind of gateway controller that will handle traffic for the Gateway.

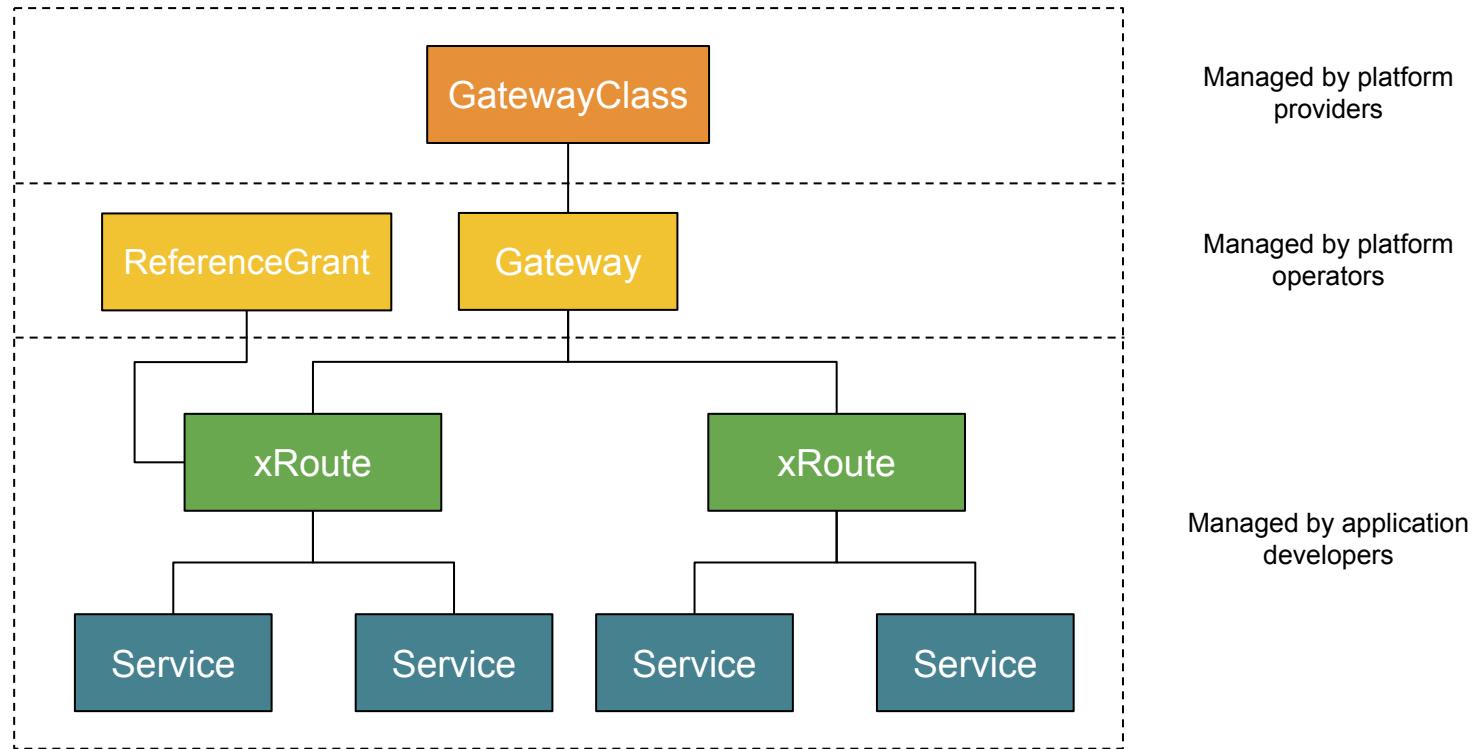
# Gateway API Resources

The API offers multiple Custom Resource Definitions

- **HTTPRoute/GRPCRoute:** Defines HTTP or GRPC-specific rules for mapping traffic from a Gateway listener to a representation of backend network endpoints. These endpoints are often represented as a Service.
- **ReferenceGrant:** Can be used to enable cross namespace references within Gateway API, e.g. routes may forward traffic to backends in other namespaces.

# Gateway API Resources Managed by Personas

Who's responsible for creating which resource?



# Installing the Gateway API CRDs

You are required to install the external resource definitions

```
$ kubectl apply -f https://github.com/kubernetes-sigs/gateway-api/releases/download/v1.2.0/standard-install.yaml
customresourcedefinition.apiextensions.k8s.io/gatewayclasses.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/gateways.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/grpcroutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/httproutes.gateway.networking.k8s.io created
customresourcedefinition.apiextensions.k8s.io/referencegrants.gateway.networking.k8s.io created
```

# GatewayClass Resource

Gateways can be implemented by different controllers

```
apiVersion: gateway.networking.k8s.io/v1
kind: GatewayClass
metadata:
  name: example-class
spec:
  controllerName: example.com/gateway-controller
```



Points to the controller  
that implements the  
Gateway API

# Gateway Resource

Network endpoint that can be used for processing traffic

```
apiVersion: gateway.networking.k8s.io/v1
kind: Gateway
metadata:
  name: example-gateway
spec:
  gatewayClassName: example-class
  listeners:
  - name: http
    protocol: HTTP
    port: 80
```

References controller  
by name

# HTTPRoute Resource

Specifies routing behavior of HTTP requests from a Gateway listener to backend

```
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: example-httproute
spec:
  parentRefs:
  - name: example-gateway
  hostnames:
  - "www.example.com"
  rules:
  - matches:
    - path:
        type: PathPrefix
        value: /login
  backendRefs:
  - name: example-svc
    port: 8080
```

Defines the routing rule  
including the URL matching and  
Service DNS name and port to  
route the traffic to



# Gateway API Implementations

Need to be installed explicitly

- Kubernetes does not provide a default implementation of the Gateway API itself.
- You will need to choose from the [list of implementations](#) best suitable to your needs and install it in your cluster.

# Ingress to Gateway API Migration

Existing Ingresses need to be replaced by Gateway API objects

- If you are planning to switch from an Ingress to the Gateway API, you need to migrate manually or use a tool.
- The tool [ingress2gateway](#) can help with the automatic conversion. The conversion results should always be tested and verified.

## Exercise

Routing HTTP traffic to a Service from outside of the cluster using the Gateway API



# Exam Essentials

What to focus on for the exam

- Understand that the Gateway API is the more flexible, and consistent replacement for the Ingress primitive.
- Become familiar with the primitives introduced by the Gateway API. Try to set up a scenario that exposes access to the Service to consumers outside of the cluster.



# Network Policies

Restricting Pod-to-Pod communication

# What is a Network Policy?

Firewall rules for Pod-to-Pod communication

- The installed [Container Network Interface \(CNI\) plugin](#) leases and assigns a new virtual IP address to every Pod when it is created.
- Communication between Pods within the same cluster is unrestricted. You can use a Pod's IP address to communicate with it.
- Network policies implement the principle of least privilege. Only allow Pods to communicate if it is needed to fulfill the requirements of architecture.
- You can disallow and allow network communication with fine-grained rules.

# What is an Network Policy Controller?

Evaluating Network Policy rules

- A Network Policy cannot work without a Network Policy controller. The Network Policy controller evaluates the collection of rules defined by a Network Policy.
- One option for such a Network Policy controller is Cilium.

# Network Policy Rules

Attributes that form the rules

Attribute	Description
podSelector	Selects the Pods in the namespace to apply the network policy to.
policyTypes	Defines the type of traffic (i.e., ingress and/or egress) the network policy applies to.
ingress	Lists the rules for incoming traffic. Each rule can define <code>from</code> and <code>ports</code> sections.
egress	Lists the rules for outgoing traffic. Each rule can define <code>to</code> and <code>ports</code> sections.

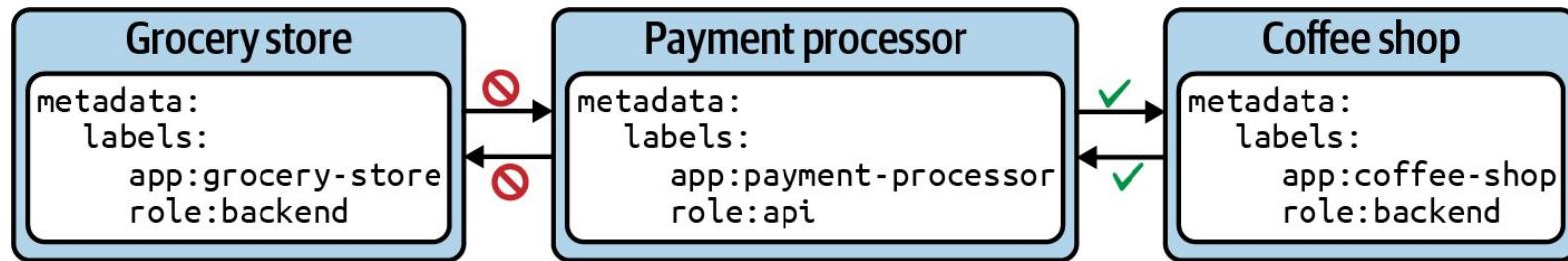
# Behavior of to and from Selectors

Ingress and egress sections define the same attributes

Attribute	Description
podSelector	Selects Pods by label(s) in the same namespace as the Network Policy which should be allowed as ingress sources or egress destinations.
namespaceSelector	Selects namespaces by label(s) for which all Pods should be allowed as ingress sources or egress destinations.
namespaceSelector and podSelector	Selects Pods by label(s) within namespaces by label(s).

# Example Network Policy

Restricting access to the payment processor Pod from other Pods



# Network Policy YAML Manifest

Key-value pairs can be parsed from different sources

```
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 Pod the policy should apply to by label selection

Allows incoming traffic from the Pod with matching labels within the same namespace

# Listing Network Policies

Doesn't provide much details apart from Pod selector labels

```
$ kubectl get networkpolicy
NAME          POD-SELECTOR          AGE
api-allow     app=payment-processor,role=api   83m
```



# Rendering Network Policy Details

Ingress and egress rules can be inspected in details

```
$ kubectl describe networkpolicy api-allow
Name:           api-allow
Namespace:      default
Created on:    2020-09-26 18:02:57 -0600 MDT
Labels:         <none>
Annotations:   <none>
Spec:
  PodSelector:    app=payment-processor,role=api
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      PodSelector: app=coffeeshop
  Not affecting egress traffic
  Policy Types: Ingress
```

# Verifying the Runtime Behavior

Communication from Pods not matching with labels will be blocked

```
$ kubectl run grocery-store --rm -it --image=busybox ↵
  -l app=grocery-store,role=backend -- /bin/sh
/ # wget --spider --timeout=1 10.0.0.51
Connecting to 10.0.0.51 (10.0.0.51:80)
wget: download timed out
/ # exit
pod "grocery-store" deleted

$ kubectl run coffeeshop --rm -it --image=busybox ↵
  -l app=coffeeshop,role=backend -- /bin/sh
/ # wget --spider --timeout=1 10.0.0.51
Connecting to 10.0.0.51 (10.0.0.51:80)
remote file exists
/ # exit
pod "coffeeshop" deleted
```

# Default Deny Network Policies

Network policies are additive

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

Selects all Pods in  
the namespace

Applies in incoming  
and outgoing traffic

# Restricting Access to Ports

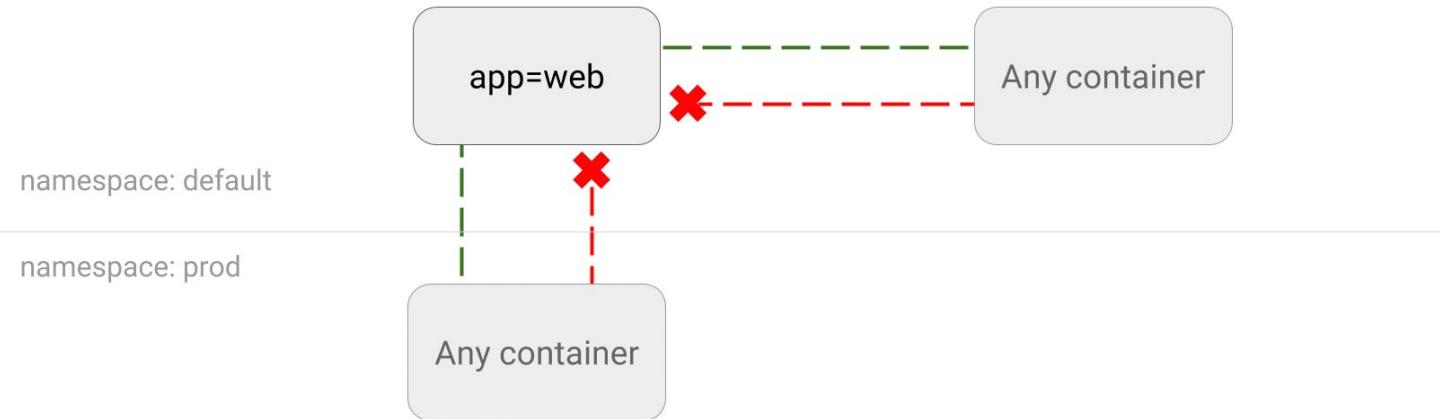
By default, all ports are accessible

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: port-allow
spec:
  podSelector:
    matchLabels:
      app: backend
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
  ports:
  - protocol: TCP
    port: 8080
```

Only allow incoming  
traffic to port 8080

# Scenario-Based Learning Resource

Exercising use cases for Network Policies is helpful with understanding concept



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

# Visualizing and Analysing a Network Policy

The page [networkpolicy.io](https://networkpolicy.io) provides policy editor

The screenshot shows the networkpolicy.io policy editor interface. At the top, there's a navigation bar with icons for back, forward, and search, followed by the URL 'editor.networkpolicy.io'. On the right side, there are buttons for 'Feedback/Questions?' and 'Ask on Slack'.

The main area displays a network policy visualization. It consists of several boxes representing different components:

- Outside Cluster**: Any endpoint
- In Namespace**: Any pod, app=coffeeshop
- In Cluster**: Everything in the cluster
- In Namespace**: { } (empty), app=payment-processor, role=api
- Outside Cluster**: Any endpoint
- In Namespace**: Any pod
- In Cluster**: Everything in the cluster, Kubernetes DNS

Red arrows point from the 'Outside Cluster' and 'In Cluster' boxes to the central 'In Namespace' box. Green arrows point from the 'In Namespace' (app=coffeeshop) box to the 'In Namespace' (empty) box, and from the 'In Cluster' box to the same central box. A small red arrow also points to the 'In Cluster' box from the bottom.

At the bottom left, there's a code editor showing the YAML representation of the policy:

```
1 apiVersion: networking.k8s.io/v1
2 kind: NetworkPolicy
3 metadata:
4   name: api-allow
5 spec:
6   podSelector:
7     matchLabels:
8       app: payment-processor
9       role: api
10  ingress:
11    - from:
12      - podSelector:
13        matchLabels:
14          app: coffeeshop
15
```

On the right side, there's a sidebar titled 'Welcome to the Network Policy Editor!' with a 'Main tutorial' section. It includes a diagram of a cluster with namespaces (namespaceA, namespaceB) and pods.

# Exercise

Restricting access to and  
from a Pod with Network  
Policies



# Exam Essentials

What to focus on for the exam

- By default, any Pod can talk any other Pod through its virtual IP address. Any container port is accessible. A Network Policy defines restrictions for Pod-to-Pod communication.
- It's best practice to create a default deny policy first to allow as little access as possible to harden security.
- Given that Network Policies are additive, you can open access with additional definitions.
- Access can be defined for egress and ingress traffic, as well as ports. Network Policies work with label selection for namespaces and Pods.

# Troubleshooting





# Topics We'll Cover

## Troubleshooting Applications

- Root cause analysis
- Fixing issues in application resources like Pods, Services, and Network Policies

## Metrics Server

- Installing the metrics server component
- Interacting with the metrics server using `kubectl`
- How does the metrics server work with enabling other features?

## Troubleshooting Clusters and Components

- Root cause analysis
- Cluster component Pods

## Container Output Streams

- Container logging
- Sidecars with logging capabilities



# Troubleshooting Pods and Containers

Built-in mechanisms for identifying the root cause of a failure

# Retrieving High-Level Pod Information

Check the status of a Pod first

```
$ kubectl get pods -n t67
```

NAME	READY	STATUS	RESTARTS	AGE
misbehaving-pod	0/1	ErrImagePull	0	2s

```
$ kubectl get pods -n k31
```

NAME	READY	STATUS	RESTARTS	AGE
successful-pod	1/1	Running	0	5s

Failed to pull  
container image

Looks successful on  
the surface-level



# Common Pod Error Statuses

Poorly-documented in Kubernetes docs, encountered often in practice

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 authentication.
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 Engine).
CreateContainerConfigError	ConfigMap or Secret referenced by container cannot be found.	Check correct name of the configuration object, verify the existence of the configuration object in the namespace.



# Inspecting Events of a Specific Pod

Detailed event information can be found in `Events` section

```
$ 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
  ...
...
```

# Inspecting Events Across all Pods

Lists events for all Pods in the given namespace

```
$ kubectl get events
```

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
3m14s	Warning	BackOff	pod/custom-cmd	Back-off restarting ↴ failed container
2s	Warning	FailedNeedsStart	cronjob/google-ping	Cannot determine if ↴ job needs to be ↴ started: too many ↴ missed start time ↴ (> 100). Set or ↴ decrease .spec. ↴ startingDeadline ↴ Seconds or check ↴ clock skew

# Inspecting Container Logs

Application failure may not necessarily surface on the Kubernetes-level

```
$ kubectl create -f crash-loop-backoff.yaml  
pod/incorrect-cmd-pod created
```

```
$ kubectl get pods incorrect-cmd-pod  
NAME                  READY   STATUS            RESTARTS   AGE  
incorrect-cmd-pod    0/1     CrashLoopBackOff   5          3m20s
```

```
$ kubectl logs incorrect-cmd-pod  
/bin/sh: unknown: not found
```



# Opening an Interactive Shell to a Container

Allows for exploring container file system and processes

```
$ kubectl logs failing-pod
/bin/sh: can't create /root/tmp/curr-date.txt: nonexistent directory

$ kubectl exec failing-pod -it -- /bin/sh
# mkdir -p ~/tmp
# cd ~/tmp
# ls -l
total 4
-rw-r--r-- 1 root root 112 May  9 23:52 curr-date.txt
# exit
```

# Distroless Containers

Some container images do not provide a shell

```
$ kubectl run minimal-pod --image=k8s.gcr.io/pause:3.1
pod/minimal-pod created

$ kubectl exec minimal-pod -it -- /bin/sh
OCI runtime exec failed: exec failed: container_linux.go:349:
↳
Starting container process caused "exec: \"/bin/sh\": stat   ↳
/bin/sh: no such file or directory": unknown
command terminated with exit code 126
```



# Debugging Using an Ephemeral Container

Using a disposable container for troubleshooting distroless containers

```
$ kubectl debug -it minimal-pod --image=busybox
Defaulting debug container name to debugger-jf98g.
If you don't see a command prompt, try pressing enter.
/ # pwd
/
/ # exit
Session ended, resume using 'kubectl attach minimal-pod -c debugger-jf98g -i -t' command when the pod is running
```

You can create a copy of a container in a new Pod with `--copy-to` for more complex debugging scenarios.

# Copying a File from the Container

The `tar` binary needs to exist in container

```
$ kubectl exec -it nginx -n code-red -- /bin/sh  
# tar cvf error-log.tar /var/log/nginx/error.log  
# exit  
  
$ kubectl cp code-red/nginx:/error-log.tar error-log.tar
```

```
$ ls  
error-log.tar
```

Namespace

File to be copied

Pod



## Exercise

# Troubleshooting a Pod





# Exam Essentials

What to focus on for the exam

- It's inevitable that Pods will run into error conditions after their creation.
- Practice all relevant `kubectl` commands that can help with diagnosing issues.
- Proactively expose yourself to failing or misbehaving Pods by provoking error conditions.



# Troubleshooting Services

Built-in mechanisms for identifying the root cause of a failure

# Checking Service Connectivity

Identify Service Type and try to connect to it

```
$ kubectl get service echoserver
NAME            TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
echoserver      ClusterIP  10.96.254.0    <none>          8080/TCP    8s

$ kubectl run tmp --image=busybox --restart=Never -it --rm \
  -- wget echoserver:8080
Connecting to echoserver (10.96.254.0:8080)
wget: can't connect to remote host (10.96.254.0:8080): Connection refused
pod "tmp" deleted
pod tmp terminated (Error)
```



Connection attempt failed

# Verifying Endpoints

Value needs to expose targeted IP address and ports of Pods

```
$ kubectl get endpoints echoserver
```

NAME	ENDPOINTS	AGE
echoserver	<none>	63s

Indicates issues with connection to Pods

```
$ kubectl get endpoints echoserver
```

NAME	ENDPOINTS	AGE
echoserver	10.244.0.3:8080,10.244.0.4:8080	63s



Proper configuration of Service provides virtual IP address and ports to Pods

# Checking Label Selection and Port Assignment

Value needs to expose targeted IP address and ports of Pods

```
$ kubectl describe service echoserver
Name:           echoserver
Namespace:      default
Labels:          svc=myservice
Annotations:    <none>
Selector:        app=echoserver
Type:           ClusterIP
IP Family Policy: SingleStack
IP Families:    IPv4
IP:             10.111.148.223
IPs:            10.111.148.223
Port:           8080-8080  8080/TCP
TargetPort:     8080/TCP
Endpoints:      10.244.0.3:8080,10.244.0.4:8080
Session Affinity: None
Events:
```

```
$ kubectl describe pod echoserver
...
Labels:          app=echoserver
Containers:      echoserver:
                  Port:  8080/TCP
```

# Checking Network Policies

Does any network policy block ingress traffic to Pod?

```
$ kubectl get networkpolicies
```

No resources found in default namespace.

```
$ kubectl get networkpolicies
```

NAME	POD-SELECTOR	AGE
default-deny-ingress	<none>	9s



Potentially blocks  
ingress traffic to all  
Pods in namespace

# Ensuring Pod Functionality

The Pod or container may have a runtime issue

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
echoserver	0/1	ErrImagePull	0	2s

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
echoserver	1/1	Running	0	5s

Failed to pull  
container image

Looks successful on  
the surface-level

## Exercise

# Troubleshooting a Service



# Exam Essentials

What to focus on for the exam

- It's easy to get the configuration of Service wrong. Any misconfiguration won't allow network traffic to reach the set of Pod it was intended for.
- Common misconfiguration include incorrect label selection and port assignments. The `kubectl get endpoints` command will give you an idea on the Pod's IP address it can route traffic to.
- Finally, make sure that the application or process running in the Pod works as expected. Try to make a call to the Pod to verify.



# Troubleshooting Control Plane Nodes

Built-in mechanisms for identifying the root cause of a failure

# Listing and Inspecting Cluster Nodes

Check the status of the cluster nodes first

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE
controlplane	Ready	control-plane	198d
worker-1	Ready	<none>	198d
worker-2	Ready	<none>	198d
worker-3	Ready	<none>	198d

Is the health status for the node anything other than “Ready”?



Does the version of a node deviate from the version of other nodes?



# Rendering Cluster Information

Shows control plane endpoints

```
$ kubectl cluster-info
```

```
Kubernetes control plane is running at https://172.29.64.5:6443  
CoreDNS is running at ↳  
https://172.29.64.5:6443/api/v1/namespaces/kube-system/services/kube-  
dns:dns/proxy
```

```
To further debug and diagnose cluster problems, use 'kubectl  
cluster-info dump'.
```

Internal DNS server

API server



# Debug and Diagnose Cluster Problems

Shows detailed logs from cluster components

```
$ kubectl cluster-info dump
...
===== START logs for container kube-proxy of pod kube-system/kube-proxy-wr9fx =====
...
===== END logs for container kube-proxy of pod kube-system/kube-proxy-wr9fx =====
===== START logs for container kube-scheduler of pod ↴
kube-system/kube-scheduler-controlplane =====
Request log error: an error on the server ("unknown") has prevented the request ↴
from succeeding (get pods kube-scheduler-controlplane)
===== END logs for container kube-scheduler of pod ↴
kube-system/kube-scheduler-controlplane =====
```

# Inspecting Control Plane Cluster Components

Check the status of the Pods in the `kube-system` namespace

```
$ kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS
coredns-6f6b679f8f-gd6ql 29m	1/1	Running	0
coredns-6f6b679f8f-ph9cl 29m	1/1	Running	0
etcd-controlplane 30m	1/1	Running	0
kube-apiserver-controlplane 30m	1/1	Running	0
kube-controller-manager-controlplane 30m	1/1	Running	0
kube-proxy-dc658 29m	1/1	Running	0
kube-proxy-wr9fx 29m	1/1	Running	0



# Checking Cluster Component Logs

Any status that does not show “Running” should be inspected further

```
$ kubectl logs pod kube-apiserver-controlplane -n kube-system  
...
```

## Exercise

Troubleshooting an issue  
with a control plane node





# Exam Essentials

What to focus on for the exam

- Understand how to identify the health status of control plane nodes.
- You can dive deeper into the cluster details using the `kubectl cluster-info` command.
- Cluster components running in the `kube-system` can fail or end up in an unhealthy state. Know how to list the Pods running the cluster components and how to render their logs to find issues.



# Troubleshooting Worker Nodes

Built-in mechanisms for identifying the root cause of a failure

# Identifying Non-Healthy Worker Nodes

The typical status you'd see here is *NotReady*

```
$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
controlplane	Ready	control-plane	198d	v1.30.1
worker-1	Ready	<none>	198d	v1.30.1
worker-2	NotReady	<none>	198d	v1.30.1
worker-3	Ready	<none>	198d	v1.30.1

# Reasons for Non-Healthy Worker Nodes

A *NotReady* state means that the node is unused

- **Insufficient resources:** The node may be low on memory or disk space.
- **Issues with the kubelet process:** The process may have crashed or stopped on the node. Therefore, it cannot communicate with the API server running on any of the control plane nodes anymore.
- **Issues with kube-proxy:** The Pod running kube-proxy is responsible for network communication from within the cluster and from the outside. The Pod transitioned into a nonfunctional state.

# Checking Available Resources

Helps with troubleshooting unavailable resources that led to error condition

```
$ kubectl describe node worker-1
...
Conditions:
  Type            Status  LastHeartbeatTime           LastTransitionTime        Reason                                  Message
  ----            -----  ---------------------           ---------------------        ----                                  -----
  NetworkUnavailable  False   Sun, 29 Sep 2024 15:41:09 +0000  Sun, 29 Sep 2024 15:41:09 +0000  FlannelIsUp
  MemoryPressure    False   Sun, 29 Sep 2024 16:32:54 +0000  Sun, 29 Sep 2024 15:40:53 +0000  KubeletHasSufficientMemory
  DiskPressure      False   Sun, 29 Sep 2024 16:32:54 +0000  Sun, 29 Sep 2024 15:40:53 +0000  KubeletHasNoDiskPressure
  PIDPressure       False   Sun, 29 Sep 2024 16:32:54 +0000  Sun, 29 Sep 2024 15:40:53 +0000  KubeletHasSufficientPID
  Ready             True    Sun, 29 Sep 2024 16:32:54 +0000  Sun, 29 Sep 2024 15:41:07 +0000  KubeletReady
  ...

```

# Checking Memory and Disk Space

Drill into deficient resources with Unix commands

```
$ 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.
```

```
$ df -h
```

Filesystem	Size	Used	Avail	Capacity	iused
/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			

# Checking the kubelet Process

Have a look at the *Active* row

```
$ systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
    Loaded: loaded (/usr/lib/systemd/system/kubelet.service; enabled; preset: enabled)
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
              └─10-kubeadm.conf
      Active: active (running) since Sun 2024-09-29 15:40:59 UTC; 58min ago
        Docs: https://kubernetes.io/docs/
   Main PID: 5055 (kubelet)
     Tasks: 44 (limit: 153)
    Memory: 43.8M (peak: 49.3M)
       CPU: 2min 526ms
      CGroup: /system.slice/kubelet.service
                └─5055 /usr/bin/kubelet
--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf
--kubeconfig=/etc/kubernetes/kubelet.c>
```



# Rendering the kubelet Logs

Provides detailed information on startup events and issues

```
$ journalctl -u kubelet.service
Sep 29 16:43:19 controlplane systemd[1]: Started kubelet.service - kubelet: The
Kubernetes Node Agent.
Sep 29 16:43:19 controlplane (kubelet) [1048]: kubelet.service: Referenced but unset
environment variable evaluates to an empty string: KU>
Sep 29 16:43:19 controlplane kubelet[1048]: E0929 16:43:19.727324      1048
run.go:72] "command failed" err="failed to load kubelet config >
Sep 29 16:43:19 controlplane systemd[1]: kubelet.service: Main process exited,
code=exited, status=1/FAILURE
Sep 29 16:43:19 controlplane systemd[1]: kubelet.service: Failed with result
'exit-code'.
Sep 29 16:43:20 controlplane systemd[1]: Stopped kubelet.service - kubelet: The
Kubernetes Node Agent.
Sep 29 16:43:24 controlplane systemd[1]: Started kubelet.service - kubelet: The
Kubernetes Node Agent.
...
...
```



# Restarting the kubelet Process

Restart the process after fixing a root cause

```
$ systemctl restart kubelet
```

## Exercise

Troubleshooting an issue  
with a worker node





# Exam Essentials

What to focus on for the exam

- Worker nodes run the workload and may need a lot of compute or storage resources. Know how to find the relevant information about the consumption of resources.
- A worker node cannot communicate with the control plane node without a running kubelet. Understand how to inspect the kubelet process status, and how to manage the kubelet background service.



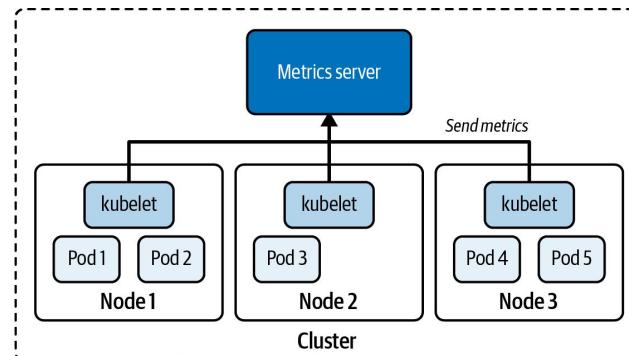
# Metrics Server

Retrieving, inspecting, and using resource metrics

# What is the Metrics Server?

Cluster-wide resource metrics aggregator

- Gathers resource consumption metrics on nodes, e.g. CPU and memory. The kubelet send those metrics to the Kubernetes API server and exposes them through the Metrics API.
- The metrics server stores data in memory and does not persist data over time.



# Installing the Metrics Server

A Kubernetes cluster does not come with the metrics server by default



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



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

# Querying for Metrics

Metrics for nodes and Pods can be rendered with the `top` command

```
$ kubectl top nodes
```

NAME	CPU (cores)	CPU%	MEMORY (bytes)	MEMORY%
controlplane	193m	9%	1239Mi	32%
node01	119m	5%	889Mi	23%

```
$ kubectl top pods
```

NAME	CPU (cores)	MEMORY (bytes)
frontend	38m	85Mi
backend	29m	59Mi



# Kubernetes Features that Tap into Metrics

Where else does the metrics server come into play? ([docs](#))

- A Horizontal Pod Autoscaler (HPA) can only determine if Deployment replicas need to be scaled up or down based on available resource consumption determined by metrics server. If the current resource consumption cannot be determined, it is indicated by <unknown> in the TARGETS column of an HPA.
- Pod scheduling on cluster nodes is based on the definition of resource requests. The scheduler looks at the resources capacity available on a cluster node and decides if the workload can be scheduled for the Pod's resource request.



## Exercise

Determining metrics for  
Pods





# Exam Essentials

What to focus on for the exam

- It's unlikely that you will have to install the metrics server component yourself.
- Understand the purpose of the metrics server and how other Kubernetes features use it to function properly.
- Practice the `kubectl top` to display runtime metrics for nodes and Pods.



# Container Output Streams

Tracking standard output and error for applications

# Why Aggregate Container Logs?

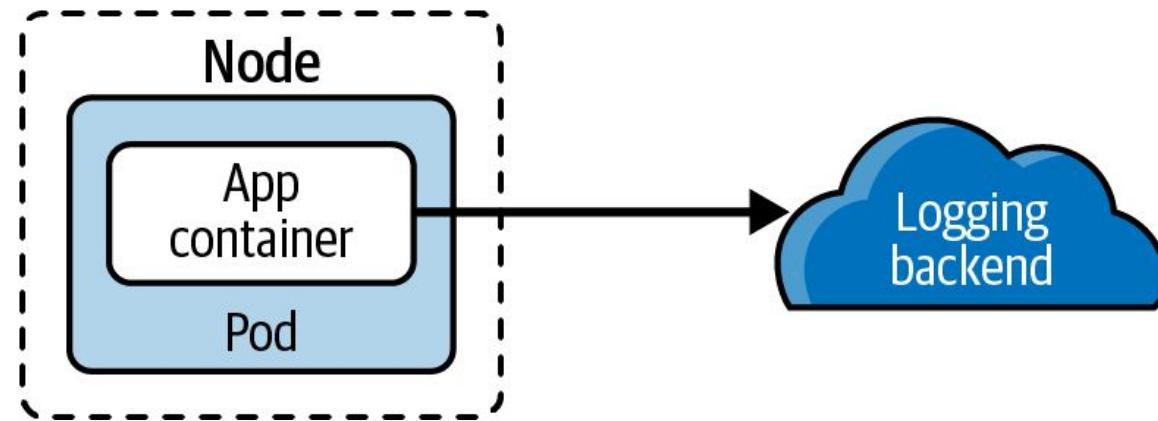
Troubleshooting of applications requires historical records

- Application can produce a lot of logs, especially when run in multiple replicas.
- You will want to store, aggregate, and rotate historical logs in external storage not bound to the lifecycle of a container.
- There are different options for sending logs to an external service.

# Pushing Directly to a Logging Backend

Easy to get started with but harder to change implementation details

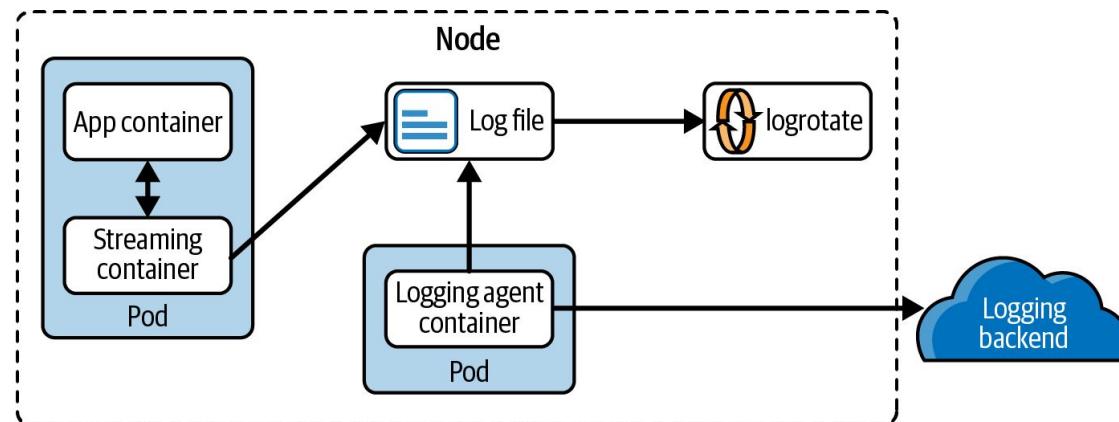
While architecturally less complex, any change to the logging backend will require a change to the application code and therefore a new deployment.



# Using a Sidecar Container

Decoupling logging functionality from application functionality

A Pod can be configured to run another sidecar container alongside the main application container. The sidecar container streams standard output and error produced by the application and redirects the streams to a different location (e.g., a logging backend or a volume mounted to the container).



# Defining More Than One Main Application Container

Add another container definition under `spec.containers`

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

Defines a second container  
running alongside the  
application container



# Creating a Pod with Multiple Application Containers

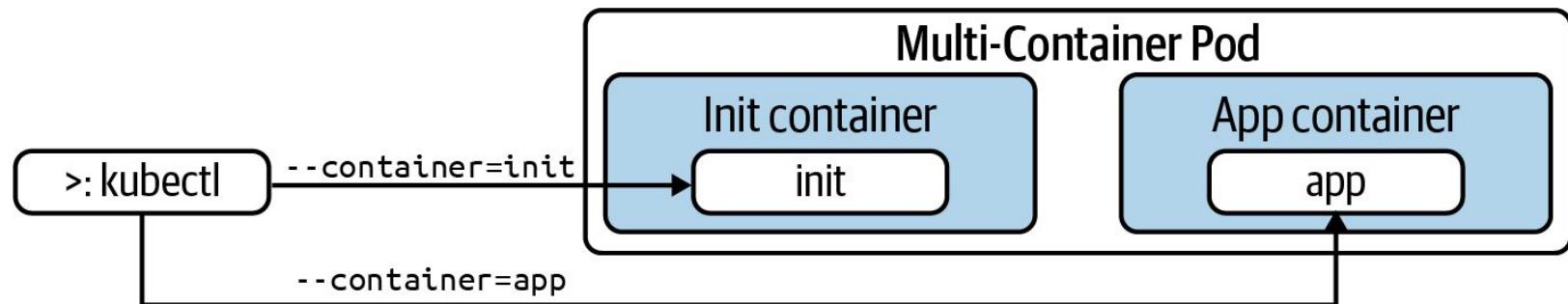
The ready column indicates the number of started containers

```
$ kubectl create -f multi-container.yaml  
pod/multi-container created
```

```
$ kubectl get pod business-app  
NAME          READY   STATUS    RESTARTS  
AGE  
multi-container  2/2     Running   0           2s
```

# Interacting with a Container

Run kubectl with `--container` or `-c` CLI option



# Exam Essentials

What to focus on for the exam

- Sidecar containers run alongside the main application container. Understand the design patterns on a high-level but do not expect having to implement them yourself.
- One use case of a sidecar container is to send logs to an external logging service.
- Know how to define a sidecar container and how to interact with it. Data between the main container and the sidecar container can be exchanged using a Volume.

The background features a vibrant red-to-yellow gradient. Overlaid on this gradient are three large, semi-transparent white circles of varying sizes, creating a sense of depth and motion.

O'REILLY®