Hands-On Kubernetes

ROITraining
Maximize Your Training Investment

Welcome!

- ROI leads the industry in designing and delivering customized technology and management training solutions
- Meet your instructor
  - Name
  - Background
  - Contact info
- Let’s get started!

Intended Audience

- Architects, engineers, and developers
- Anyone who wants to learn to use Kubernetes to build cloud-native, microservice applications and automate their deployment using Kubernetes

Course Prerequisites

To get the most out of this course, you should have:

- Basic understanding of cloud computing
- Experience administering or deploying applications to Linux or Windows
- Experience programming web applications or services
- Basic understanding of data storage

Course Objectives

In this course, you will learn how to:

- Build, run, and deploy container applications using Docker
- Configure Kubernetes clusters in the cloud
- Automate application management using the `kubectl` CLI and configuration
- Deploy scalable, fault-tolerant applications using Kubernetes Deployments, Services, Jobs, CronJobs, and DaemonSets
- Migrate to new versions of services safely with zero downtime
- Save data in Kubernetes using Persistent Volumes, StatefulSets, ConfigMaps, and Secrets
- Simplify Kubernetes deployments with Helm
- Manage Kubernetes security with Role-Based Access Control (RBAC)
- Enhance Kubernetes networking with network policies and Istio

Course Contents

Course Schedule

- Start of class: ______________
- Break: ______________
- Lunch: ______________
- Resume: ______________
- Break: ______________
- Class ends: ______________

Student Introductions

Please introduce yourself stating:

- Name
- Position or role
- Current project you are working on
    - What is the current phase or stage of the project?
- Expectations or a question you'd like answered during this class

Overview of Kubernetes

Chapter Objectives

In this chapter, we will:

- Review containerization
  - Advantages of containers
  - Microservices
  - Container orchestration
- Introduce Kubernetes

Chapter Concepts

Containerized Applications

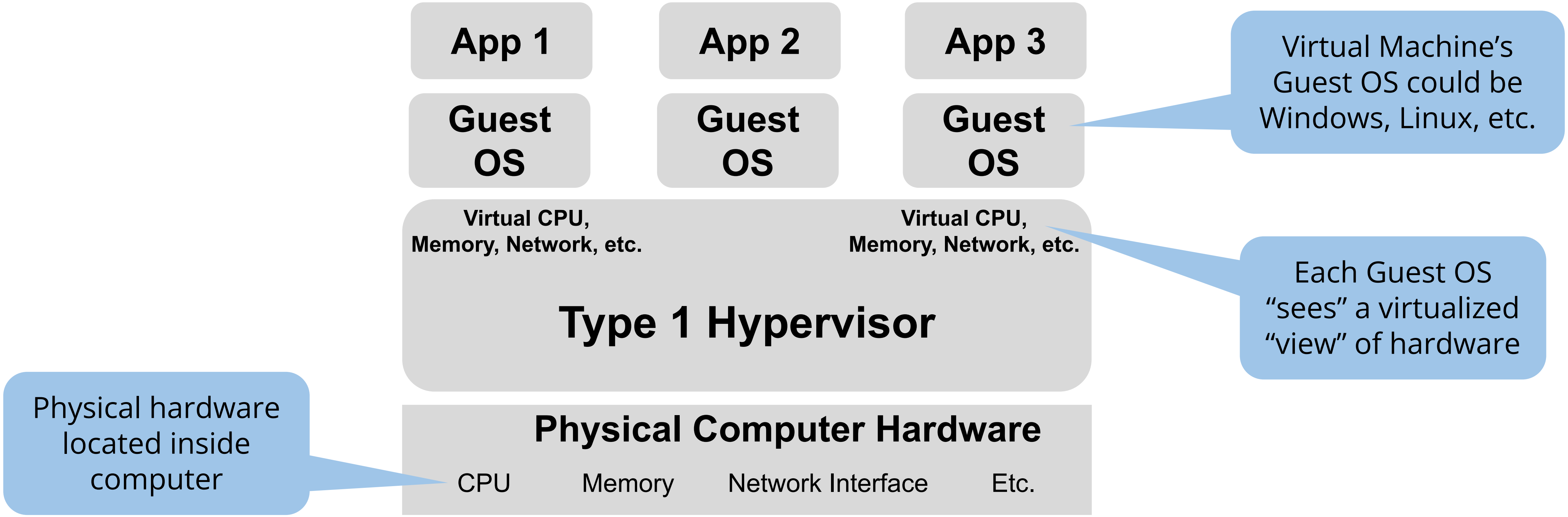
Introducing Kubernetes

Virtual Machine-Based Virtualization

- Virtual machine-based virtualization is the ability to virtualize hardware
  - Allows resources of a single physical computer to be shared among more than one virtual machine
  - Each virtual machine (VM) has its own virtual set of resources
  - Each VM provides functionality to execute entire operating systems (OS)
- Software that manages this virtualization is known as a hypervisor
  - Type 1 or native hypervisor
  - Type 2 or hosted hypervisor (not discussed here)

Type 1 (Native) Hypervisor

- A type 1 hypervisor runs directly on the machine hardware
  - Each VM runs in complete isolation from other virtual machines
  - Hypervisor exposes a subset of the actual hardware

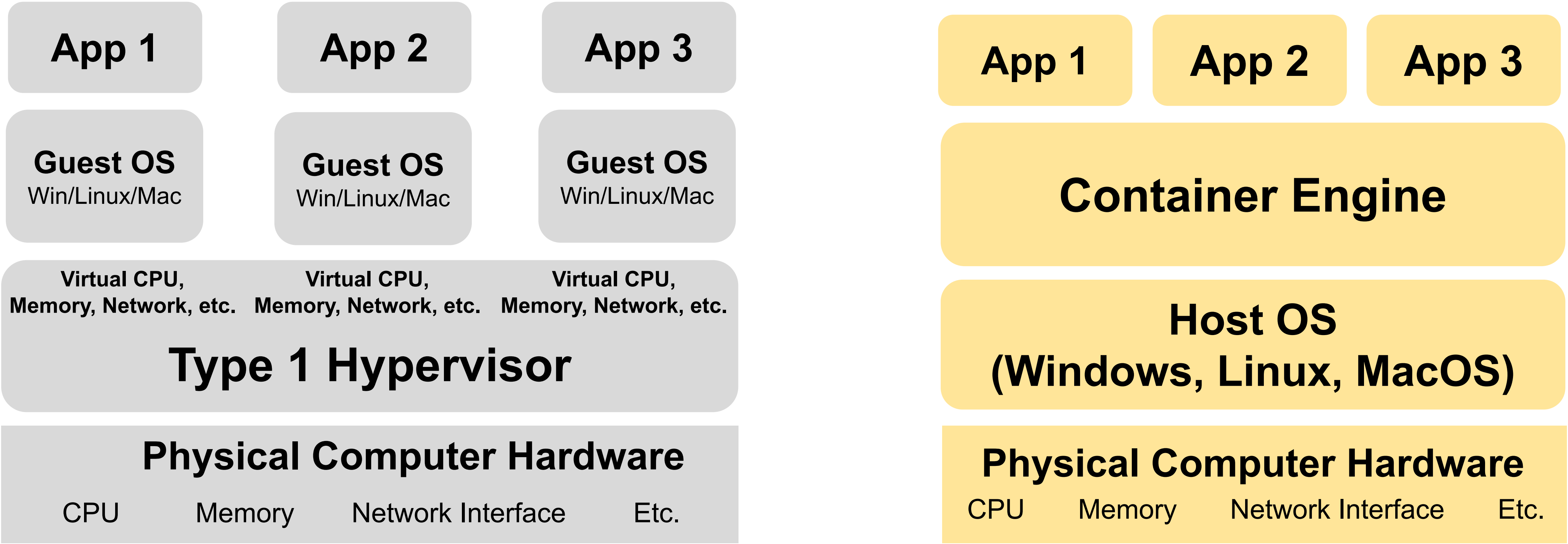


Container-Based Virtualization

- Container-based virtualization is the ability to virtualize the “user space”
  - Multiple isolated systems, called containers, access a single OS kernel
- Everything required for a piece of software to run is packaged into an isolated container
  - Applications don’t need their own copy of the OS
  - Only software dependencies, libraries, and settings required to make the software work
- Can deploy and run applications without launching an entire VM for each application

Virtual Machines vs. Containers

- Applications hosted on virtual machines require significantly more resources
  - Each application on a VM requires its own operating system
  - Containers all share the same OS

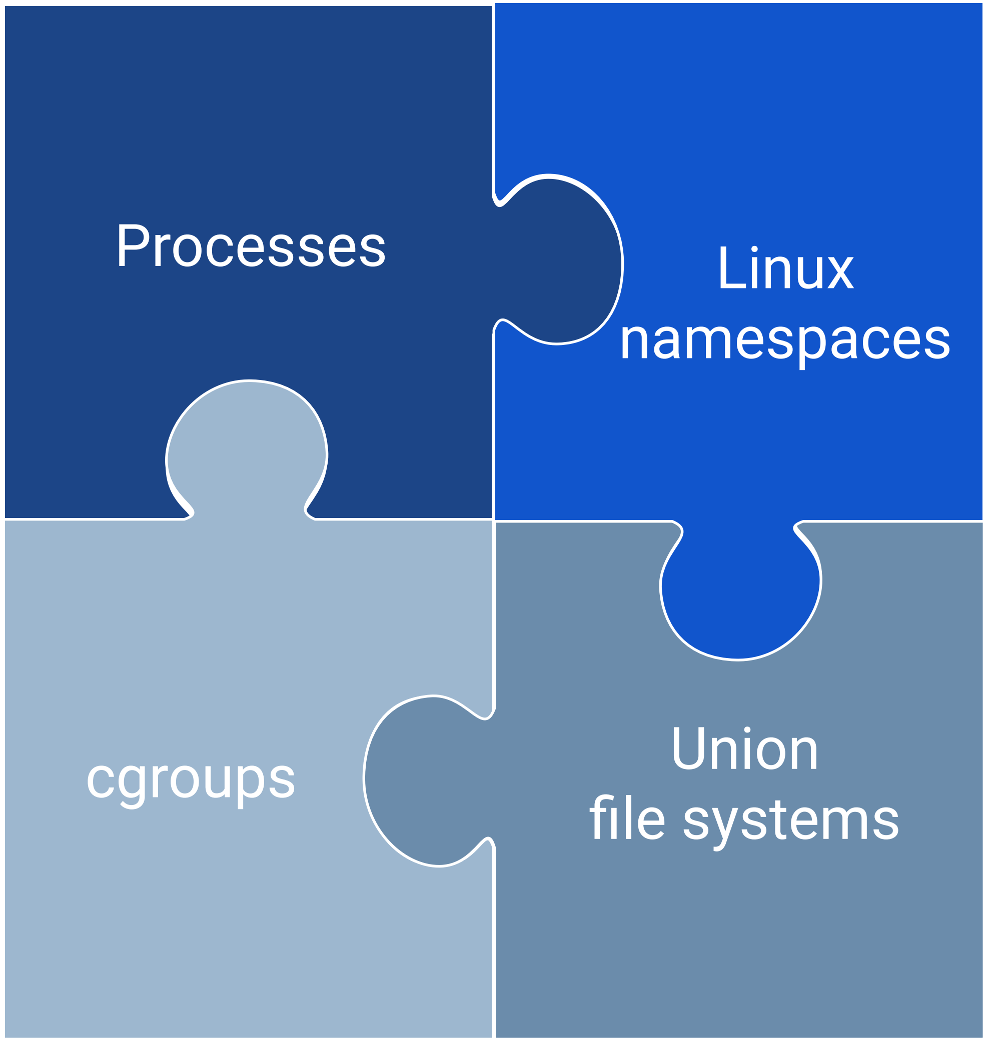


Advantages of Containers

- More efficient
  - Require less resources, less overhead
  - Can support more containers on same hardware
- Better performance
  - Containers can be very fast to start
    - Potentially less than 1 second
    - No OS to boot, which takes tens of seconds or even minutes
- Provides an agile environment
  - Can improve portability across systems
  - Encapsulates application dependencies
  - Facilitates a microservices/DevOps approach
  - Works well with a continuous integration strategy

Container Technologies

- The ability to isolate containers comes from the composition of several technologies

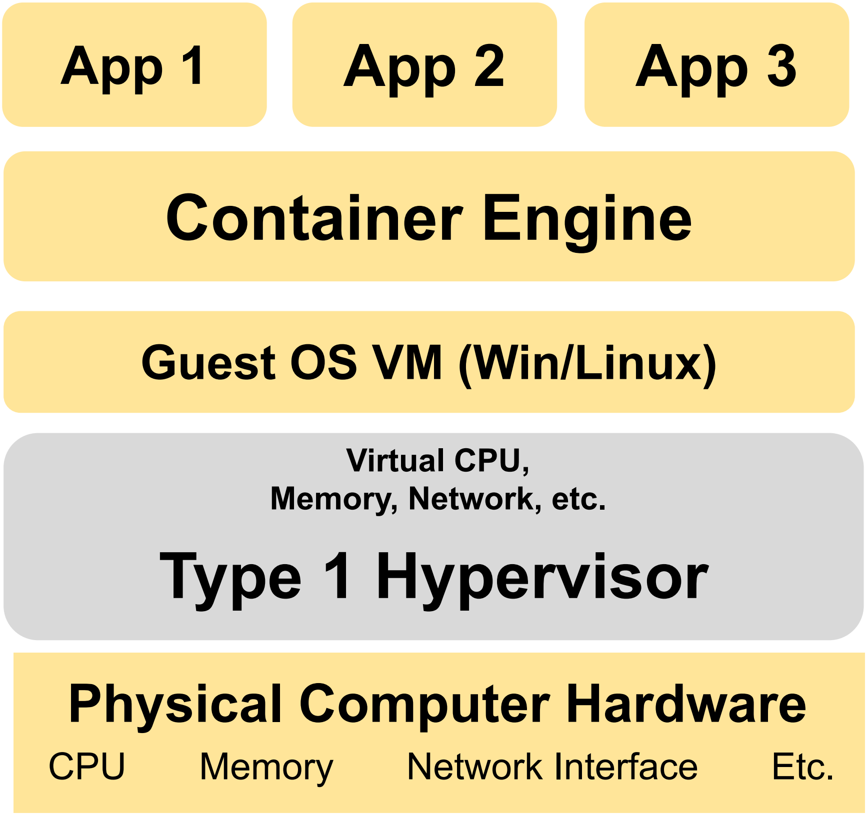


Docker

- Docker is one of the most popular container environments
  - www.docker.com
  - Docker Engine client runs natively on Linux, MacOS, and Windows

Containers and VMs

- Most containerized platforms today also use virtual machines
- Container engines run on top of virtualized servers
  - Leverages both hypervisor-based and container-based virtualization
  - Helps improve isolation and security



Advantages of Containers

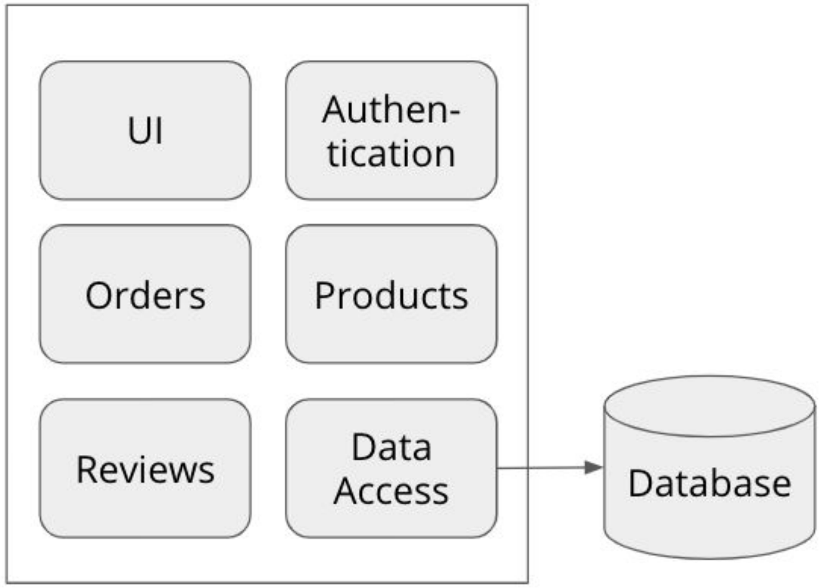
- Start very fast (milliseconds)
- Small, so easy to move around
- Isolates one program from another
- Run on any machine

Microservice Architecture

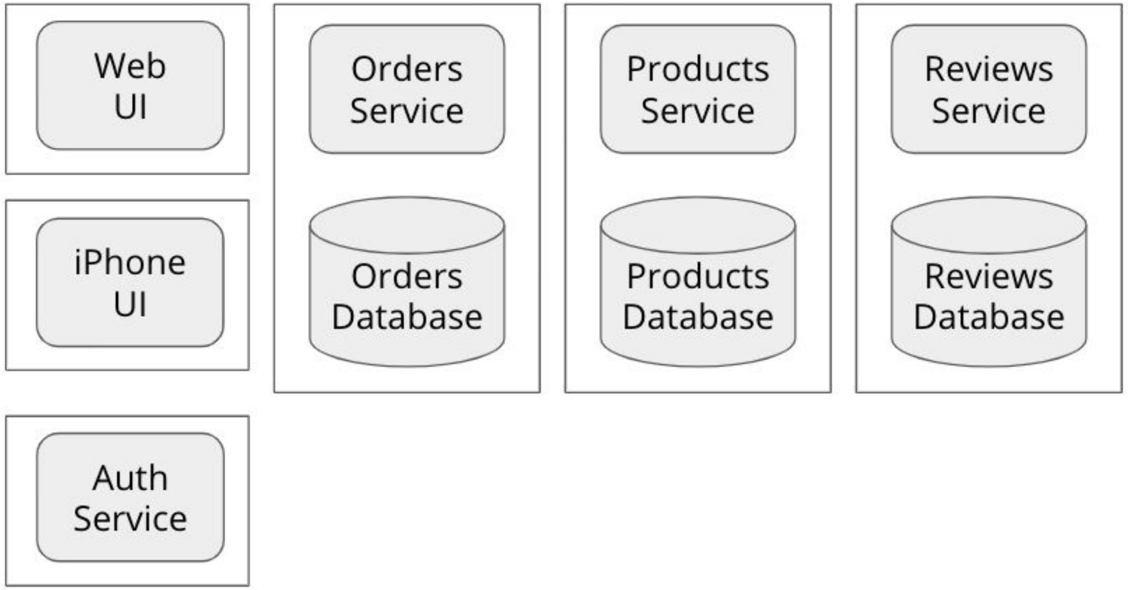
- Divide a large program into a number of smaller, independent services
  - Each service is programmed, deployed, and run separately
- Advantages of microservices include:
  - Reduced risk when deploying new versions
  - Services scale independently to optimize use of infrastructure
  - Easier to innovate and add new features
  - Can use different languages and frameworks for different services

Monolithic vs. Microservice Architecture

- Monolithic applications implement all features in a single code base
    - Single database for all data

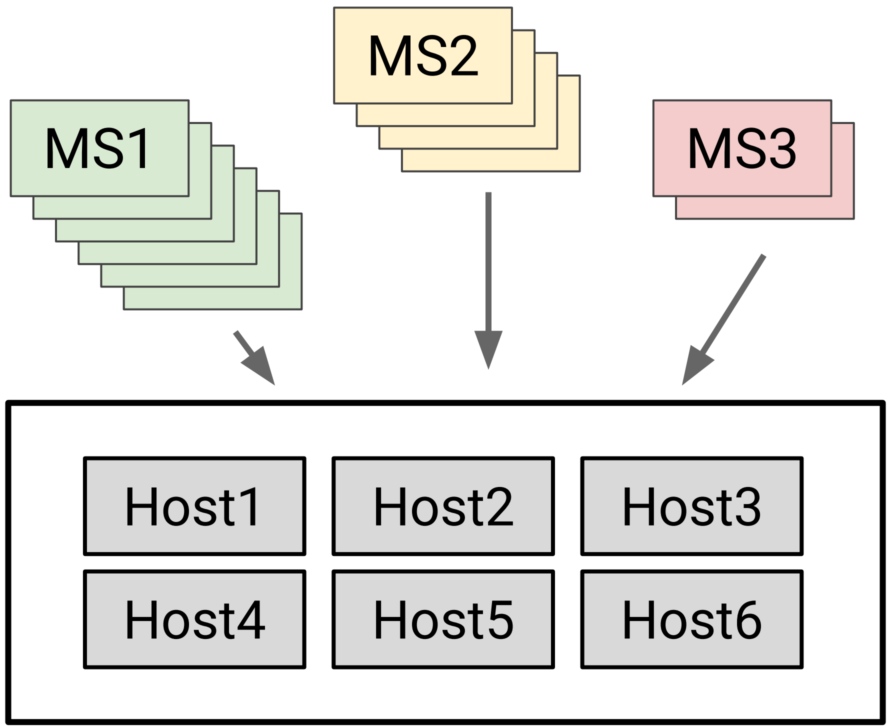


- Microservices break large programs into a number of smaller services
    - Each service manages its own data



Microservices and Containers

- Each microservice can be deployed in a container
  - Build code modularly
  - Deploy it easily
  - Scale containers and hosts independently
    - More efficient
    - Better resource utilization



Container Orchestration

- Deploying microservice applications in containers does not solve all problems
  - And actually creates some new problems
- There are many more small “services” to manage
  - Must manage multiple copies of each container on multiple VMs for fault tolerance and high availability
  - Scaling resources on and off to meet demand while minimizing costs
  - Route traffic among several copies of available containers
  - Deploy new versions of a service

Chapter Concepts

Containerized Applications

Introducing Kubernetes

Kubernetes (K8s)

- Open-source container orchestration system
  - Automates deployment, scaling, and management of containerized applications
  - Open sourced in June 2014
- Originally developed by Google to run Google's data centers
  - Now maintained by the Cloud Native Computing Foundation
  - Designed to operate at Google scale
  - Proven and tested running Google's applications

Kubernetes

- Extremely popular and active open-source project
  - www.kubernetes.io or www.k8s.io
- Can run in virtually any environment
  - Public cloud, private cloud
  - On-premises data centers
  - Virtual machines or physical hosts
- Wide support
  - Google Cloud Google Kubernetes Engine (GKE)
  - Microsoft Azure Kubernetes Service (AKS)
  - AWS Elastic Kubernetes Service (EKS)
  - Red Hat OpenShift
  - Meso and Pivotal Cloud Foundry support Kubernetes

Multi- and Hybrid-Cloud

- Kubernetes provides a set of standards that helps you run applications
  - It is an ideal foundation for a hybrid cloud strategy because it provides consistency
  - On-premises or on one or more public clouds
- You may need to deploy your application on-premises today
  - And to the cloud two months from now
  - Kubernetes can make that easy to do

Portability

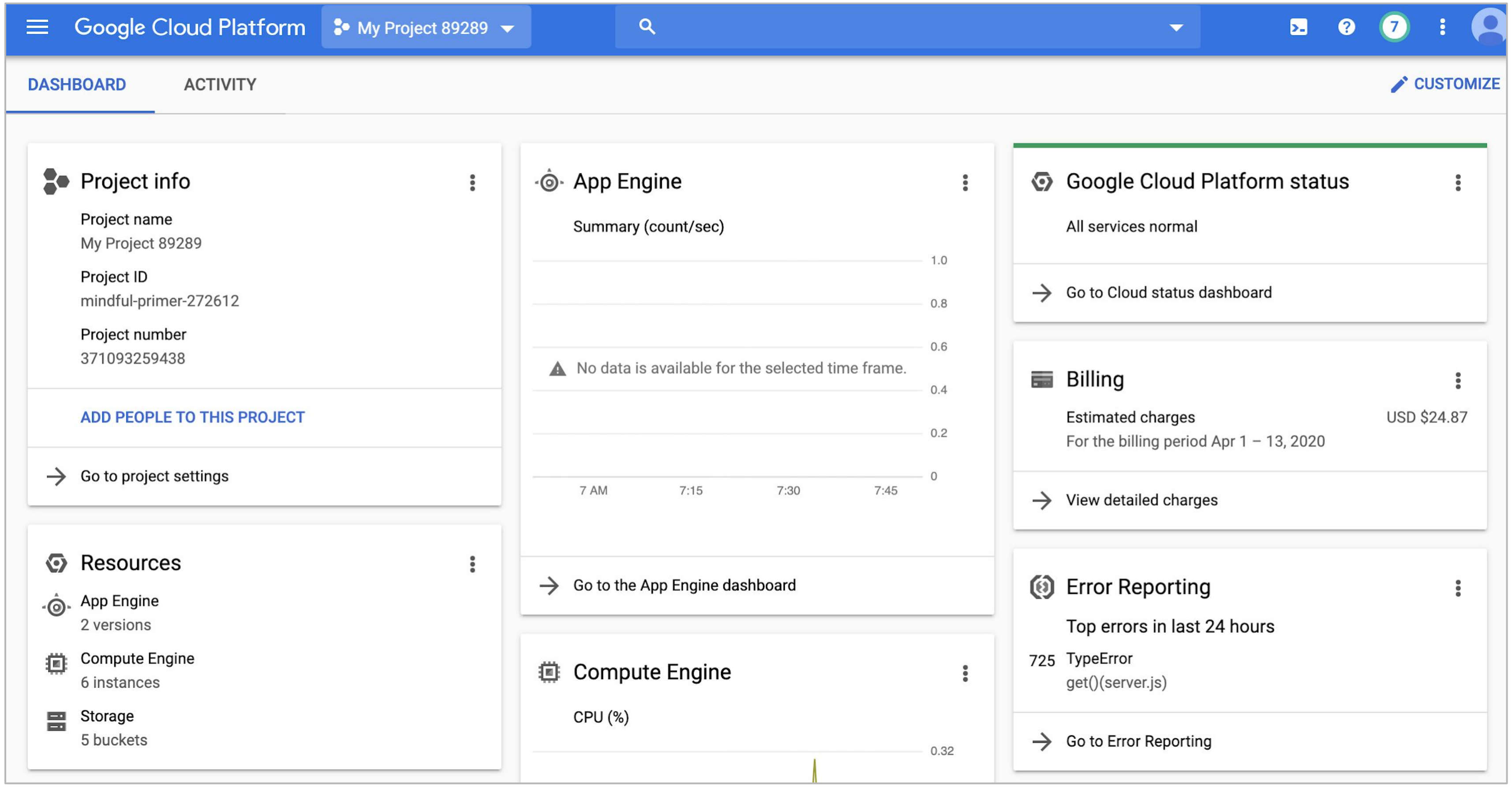
- Most topics in this course will work on any Kubernetes implementation
- In this course, we will be using Google Cloud
  - The principles learned can be applied to any cloud provider
    - AWS, Google Cloud, Azure, etc.
- You will be provided with a Google Cloud account for use during the class
  - Not available for use after class
- Another cloud implementation (AWS/Azure) will be demonstrated
- Do not worry if you will be using another cloud provider
  - Only one activity in the course is specific to Google!

Why We Are Using Google Cloud

- The Kubernetes concepts in this course apply to all clouds
- A main reason we are using Google Cloud is because of Cloud Shell
  - Pre-configured Linux VM makes setup easier
  - Built-in Code Editor
  - Allows labs to be done without configuring tools
  - Google Cloud Shell has most things we need pre-installed
    - Docker
    - eksctl
    - Helm
    - Git
    - Node.js
    - And more ...

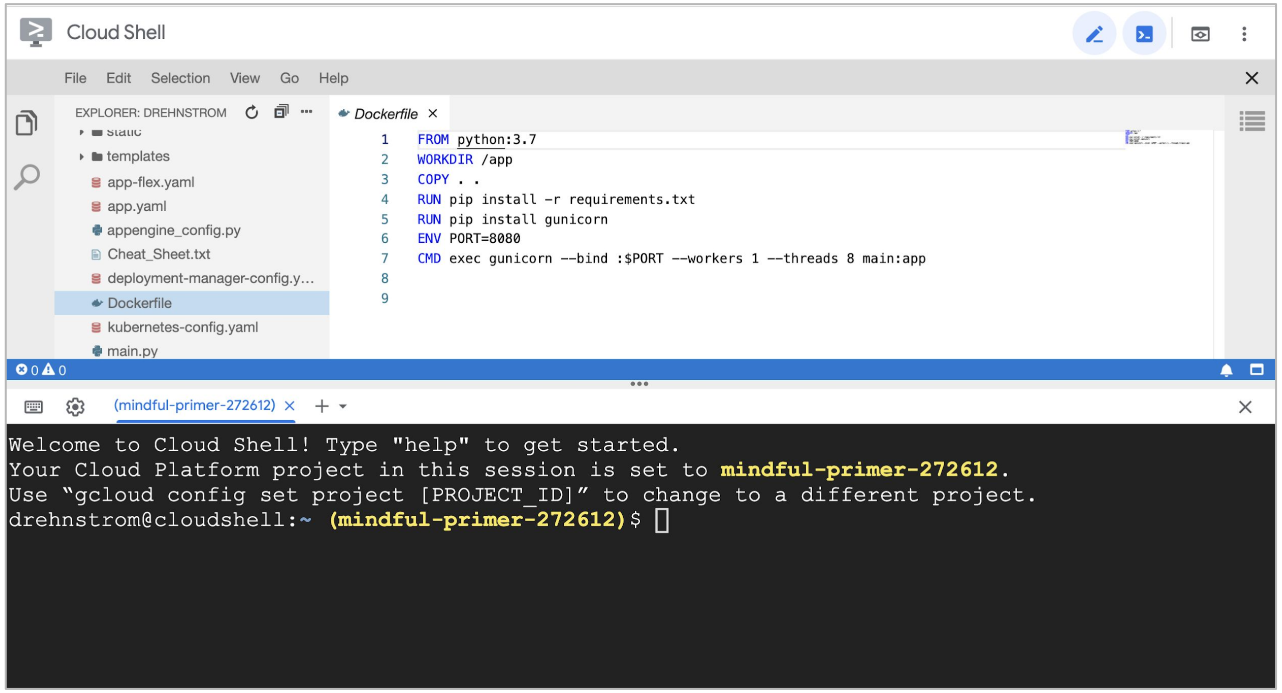
Activity: Logging In to Google Cloud

- Your instructor will explain how to get logged in to Google Cloud



Google Cloud Shell

- Online Code Editor provided free as a part of Google Cloud Shell
  - 5GB of storage per account
- Linux VM has requirements for this course pre-installed



Case Study: Events Feed

- A simple application will be provided that implements an events feed system
  - Could be the basis of an application that a company can use to announce events, news, and important messages
  - Written in Node.js
- During the course, you will:
  - Containerize the application
  - Deploy to Kubernetes
  - Load balance
  - Autoscale
  - Perform application updates and roll backs
  - And more …

Activity: Running the Case Study App

In this activity, you will:

- Download the case study application and run it locally within Cloud Shell

Chapter Summary

In this chapter, we have:

- Reviewed containerization
  - Advantages of containers
  - Microservices
  - Container orchestration
- Introduced Kubernetes

Docker

Chapter Objectives

In this chapter, you will:

- Deploy microservice applications using containers
- Leverage Docker to build, run, and manage containers

Chapter Concepts

Understanding Docker

Using Docker

Deploying Docker Containers

Docker

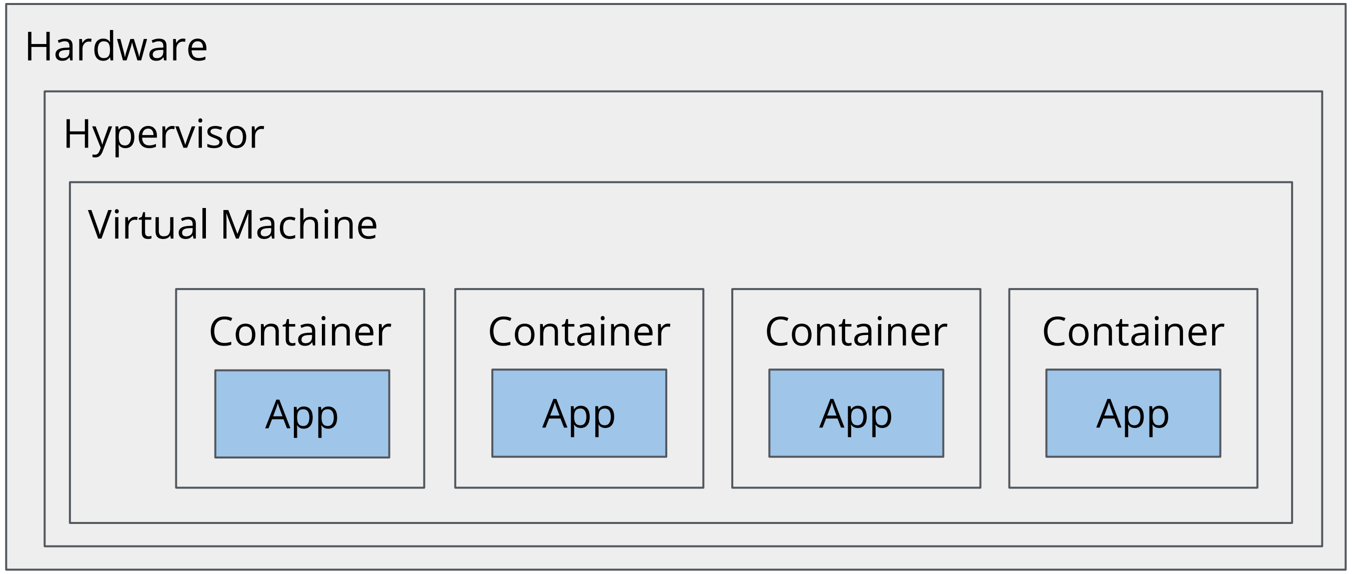
- Allows applications or microservices to be deployed to containers
    - Multiple containers can run on a single virtual machine
- Docker images are very lightweight, pre-configured virtual environments
    - Include the required software to run an application
    - Applications are inside the Docker image
- Docker images will run on any platform that has Docker installed
- Docker images allow applications to be easily moved
    - From developer to test to production environments
    - Between local and cloud-based data centers
    - Between different cloud providers

Images

- Images are deployment packages that are used to build containers
  - Containers are running instances of images
- Images are built in layers
  - Start with a base image
  - Add languages and frameworks used by your app
  - Copy in your code
  - Create environment variables
  - Specify how your application starts

Containers

- Containers are running instances of images
- Containers do not include the operating system
  - The OS requires the container software be installed (Docker)



Chapter Concepts

Understanding Docker

Using Docker

Deploying Docker Containers

Some Basic Docker Commands

Command	Description
docker build [OPTIONS] PATH | URL | - **Example:** docker build -t drehnstrom/converter-dar:latest .	Build a custom Docker container based on a `Dockerfile`. Run the command from the same folder as the `Dockerfile`.
docker run [OPTIONS] IMAGE [COMMAND] [ARG...] **Example:** docker run -d -p 8080:8080 drehnstrom/converter-dar	Run a Docker image.
docker ps [OPTIONS	List running docker images. Displays containers and their IDs.
docker stop [OPTIONS] CONTAINER [CONTAINER...] **Example:** docker stop <container-id-here>	Stop a running image.
docker login [OPTIONS] [SERVER]	Login to Docker Hub.
docker push [OPTIONS] NAME[:TAG] **Example:** docker push drehnstrom/converter-dar	Push a container to Docker Hub.
docker pull [OPTIONS] NAME[:TAG] **Example:** docker pull drehnstrom/converter-dar	Get a container from Docker Hub.

Creating Custom Docker Containers

- To build a custom image, create a file called `Dockerfile`
- Steps
  1. Start with a base image from Docker Hub or another registry
  2. Install prerequisite software onto the base image
  3. Copy your application onto the image
  4. Configure your application
  5. Specify how to start your application
- Use `docker build` command to create the container
- Once the container is created, use `docker run` command to start it

Example `Dockerfile` for a Node.js App

```
FROM launcher.gcr.io/google/nodejs
COPY . /app/
WORKDIR /app
RUN npm install
CMD ["node", "server.js"]
```

Example `Dockerfile` for Python App

```
FROM python:3
WORKDIR /usr/src/app
COPY . .
RUN pip3 install -r requirements.txt
CMD [ "python3", "./main.py" ]
```

Example Dockerfile for a .NET App

```
FROM microsoft/dotnet:latest
COPY ./ /app
WORKDIR /app
RUN ["dotnet", "restore"]
RUN ["dotnet", "build"]
EXPOSE 8080/tcp
ENTRYPOINT ["dotnet", "run", "--server.urls",
"http://0.0.0.0:8080"]
```

Example `Dockerfile` for an Nginx Website

```
FROM nginx:latest
WORKDIR /usr/share/nginx/html
COPY . .
EXPOSE 80
```

ENTRYPOINT or CMD

- Docker images must have an ENTRYPOINT or CMD declaration for a container to start
  - ENTRYPOINT and CMD instructions may seem similar
  - CMD
    - Sets default parameters that can be overridden from the Docker Command Line Interface (CLI) when a container is running
    - Use when you need a default command which users can easily override
  - ENTRYPOINT
    - Default parameters that cannot be overridden when Docker containers run with CLI parameters

ENTRYPOINT or CMD (continued)

- It is possible to have both ENTRYPOINT and CMD in your `Dockerfile`
  - The executable is defined with ENTRYPOINT
  - Specify default parameters with CMD

```
FROM alpine:3.14
ENTRYPOINT ["echo", "Hello"]
CMD ["World"]
```

Building Docker Images

- Use the `docker build` command to create the image
  - The `-t` or `--tag` parameter tags (*names*) the image (can include a version after a `:`)
  - Specify the path to the `Dockerfile`
- Tag is used later to specify which image you want to run
- Syntax:
  - `docker build -t your-image:v0.1 .`
  - `docker build -t [registry-id]/your-image:v0.1 .`

The .dockerignore File

- A `.dockerignore` file can be used to specify ignore rules and exceptions for the Docker `COPY` command
  - Place it in the root folder of the service
  - Similar concept to `.gitignore`
- A simple `.dockerignore` file for Node.js:

```
node_modules
npm-debug.log
```

Example Build Command Output

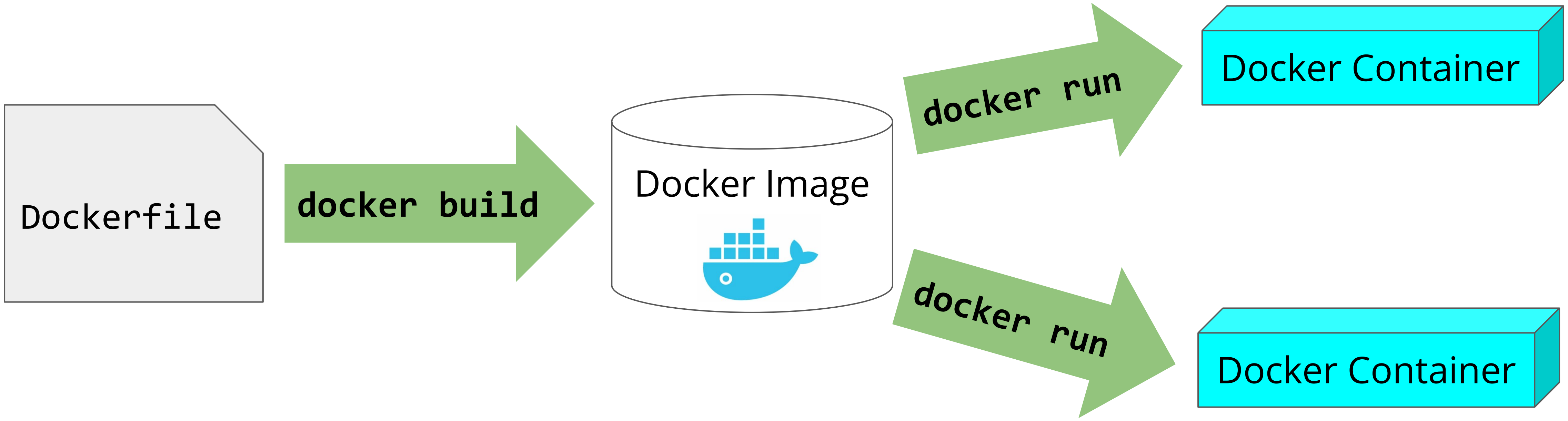
```
$ docker build -t drehnstrom/devops-demo:v0.1 .
Sending build context to Docker daemon  2.828MB
Step 1/7 : FROM python:3.7
 ---> 34a518642c76
Step 2/7 : WORKDIR /app
<< CODE OMITTED>>
Step 6/7 : ENV PORT=8080
 ---> Using cache
---> 7045daaafd44Step 7/7 : CMD exec gunicorn --bind :$PORT --workers 1 --threads 8
main:ap ---> Using cache
---> 7c32a538632e
Successfully built 7c32a538632e
Successfully tagged drehnstrom/devops-demo:v0.1
```

Starting Containers

- Use the `docker run` command to start a container based on an image
    - -p parameter specifies the port to listen on and the port to forward to
- Example:

```
$ docker run -p 8080:8080 drehnstrom/devops-demo:v0.1
[2019-07-02 12:07:13 +0000] [1] [INFO] Starting gunicorn 19.9.0[2019-07-02 12:07:13
+0000] [1] [INFO] Listening at: http://0.0.0.0:8080 (1)[2019-07-02 12:07:13 +0000] [1]
[INFO] Using worker: threads[2019-07-02 12:07:13 +0000] [8] [INFO] Booting worker with
pid: 8
```

Docker Images and Containers



Listing Containers and Images

- To see your containers, use the `docker ps` command
  - `-a` parameter shows all containers, not just those that are running

```
$ docker ps -a
CONTAINER ID   IMAGE                          COMMAND
7db7aed583f8   drehnstrom/devops-demo:v0.1    "/bin/sh -c 'exec
```

- To see your images, use the `docker images` command

```
$ docker images
REPOSITORY               TAG    IMAGE ID       CREATED            SIZE
drehnstrom/devops-demo   v0.1   7c32a538632e   23 minutes ago     946MB
python                   3.7    34a518642c76   3 weeks ago        929MB
```

Deleting Containers and Images

- Use the `docker rm` command to remove containers
  - `docker rm <CONTAINER ID>`
- To stop all running containers:
  - `docker stop $(docker ps -a -q)`
- To remove all containers:
  - `docker rm $(docker ps -a -q)`
- Use the `docker rmi` command to remove images
  - `docker rmi <IMAGE ID>`

Tutorial: Getting Started with Docker

- To learn more about Docker, do the following tutorial:
  - https://docs.docker.com/get-started/

Activity: Building Docker Images

In this activity, you will:

- Download the case study application and run it locally within Cloud Shell
- Build Docker images for the application services
- Run your Docker images locally within Cloud Shell

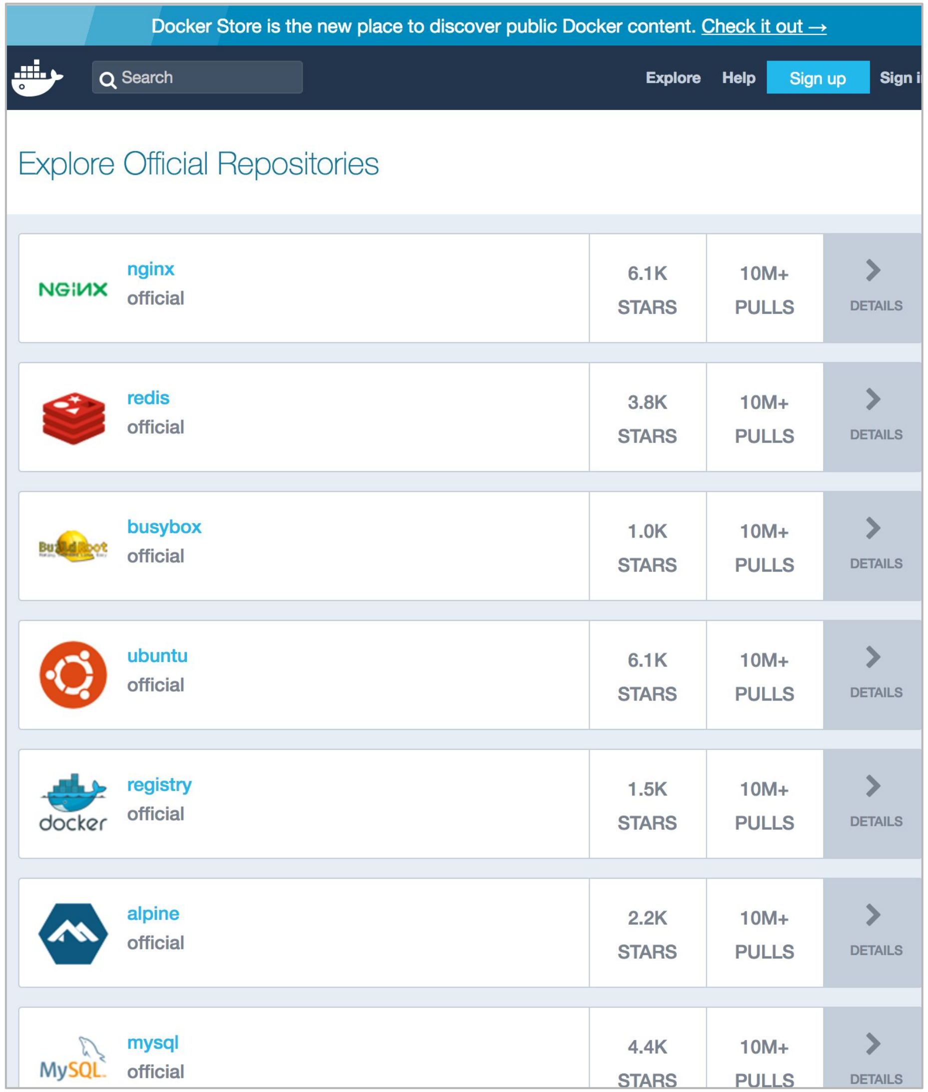
Chapter Concepts

Docker Registries

- Registries are centralized locations where Docker images can be stored
- Public registries are available to everyone
  - Base images for different environments are often stored publicly
  - Open-source applications might be stored in public registries
- Private registries are secured and managed by some organization
  - Control access to your proprietary software
- Registries are easy to create
- Access registries over the internet or your private network

Docker Hub

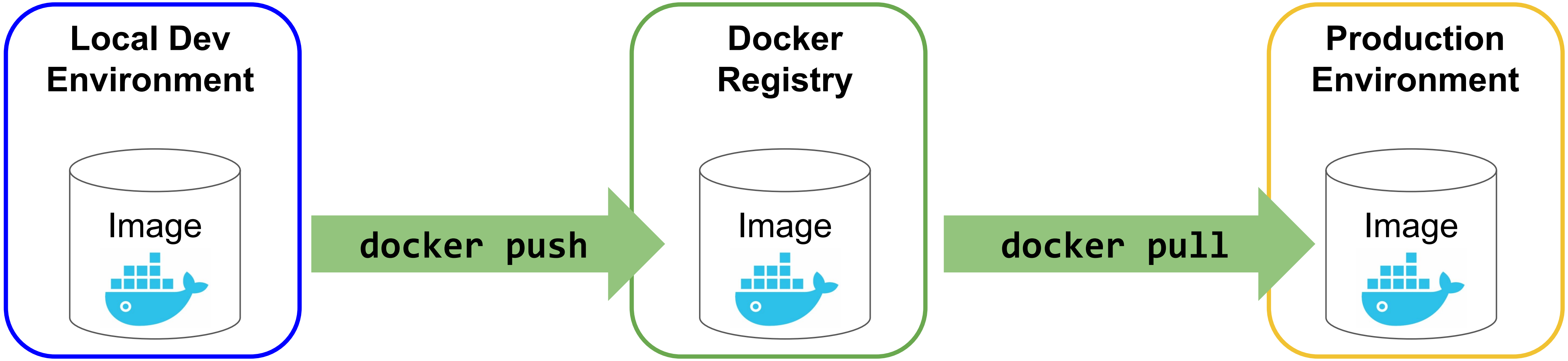
- Official registry of Docker images
  - Can create both public and private Docker repositories
- Images for many operating systems and languages
  - Starting points for building your images
- Can upload custom images
  - When deployed onto systems, your custom images are downloaded from Docker Hub



Push and Pull

- Use the `docker push` command to save an image to a repository
  - To save a container to Docker Hub:

    `docker push` ***`your-docker-id`***`/devops-demo:v1.0.0`
- Use the `docker pull` command to get an image from a repository
  - `docker pull` ***`your-docker-id`***`/devops-demo:v1.0.0`



Running Your Own Container Registry

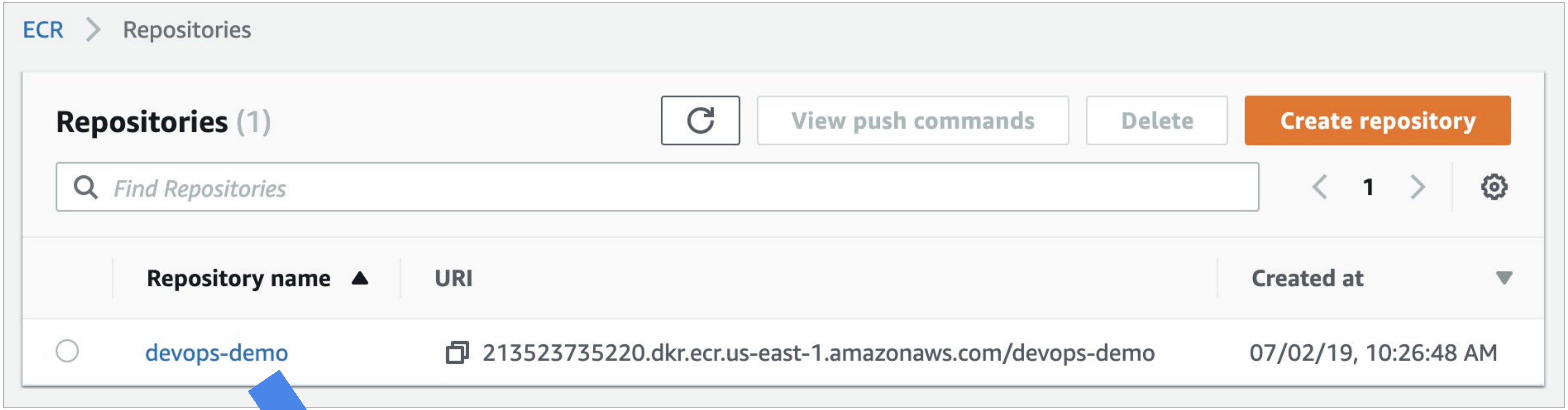
- Any server running Docker can act as a container registry
  - Just start the registry service and push images to it
- Allows complete control over storage of your Docker containers
- Example commands:

```
docker run -d -p 5000:5000 --name registry registry:2
```

```
docker pull ubuntu
docker tag ubuntu localhost:5000/myfirstimage
```

```
docker push localhost:5000/myfirstimage
docker pull localhost:5000/myfirstimage
```

AWS Elastic Container Registry (ECR)




```
$ docker push 213523735220.dkr.ecr.us-east-1.amazonaws.com/devops-demo:1.0.0
```

Google Cloud Container Registry

- Docker registry provided by Google Cloud
  - Private and only available to those who are given access
- Container Builder service, creates containers and stores them
- To store images in Google Container Registry their names must be prefixed with `gcr.io/`

```
docker build -t gcr.io/gcp-project-id/container-name:v1.0 .
docker push gcr.io/gcp-project-id/container-name:v1.0
```

- The Google CLI can perform a build and push in one command:

```
gcloud builds submit --tag gcr.io/gcp-project-id/container-name:v1.0 .
```

Activity: Using a Container Registry

In this activity, you will:

- Use a Container Registry to store your Docker containers
- Delete all local copies
- Run directly from the registry

- For this activity, we will use Google Container Registry
    - But another registry such as Docker Hub would work as well

Optional Homework: Docker Hub

- Visit https://www.docker.com/ and create a Docker ID
  - Docker ID allows you to use the Docker Hub Repository
- Push your images to Docker Hub

Chapter Summary

In this chapter, you have:

- Deployed microservice applications using containers
- Leveraged Docker to build, run, and manage containers

Kubernetes Clusters

Chapter Objectives

In this chapter, you will:

- Review the Kubernetes architecture
- Configure and create a Kubernetes managed cluster to host containers

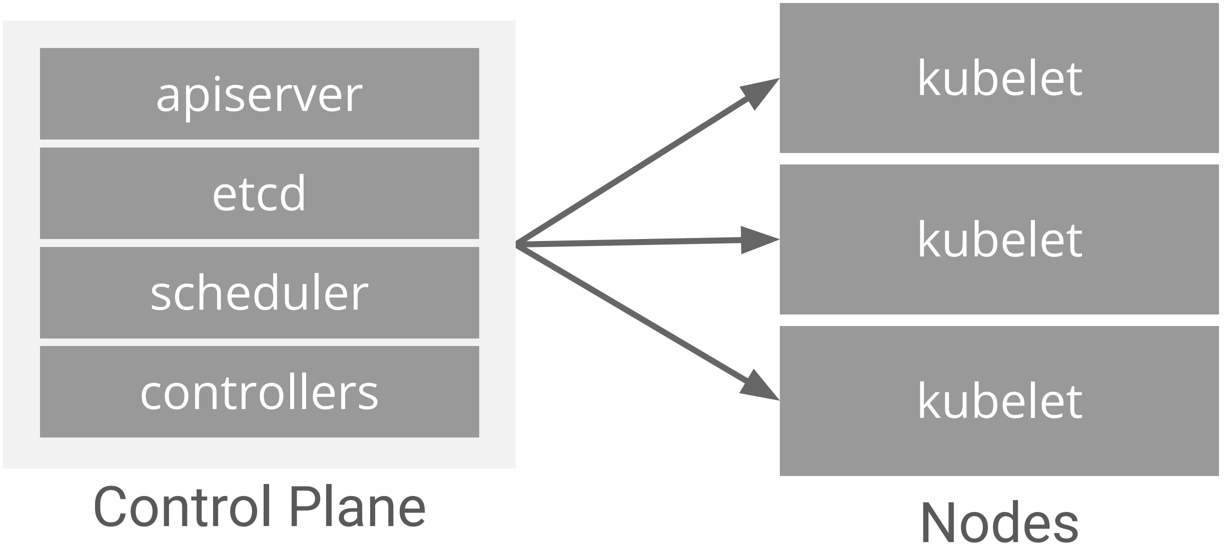
Chapter Concepts

Kubernetes Architecture

Managed Clusters

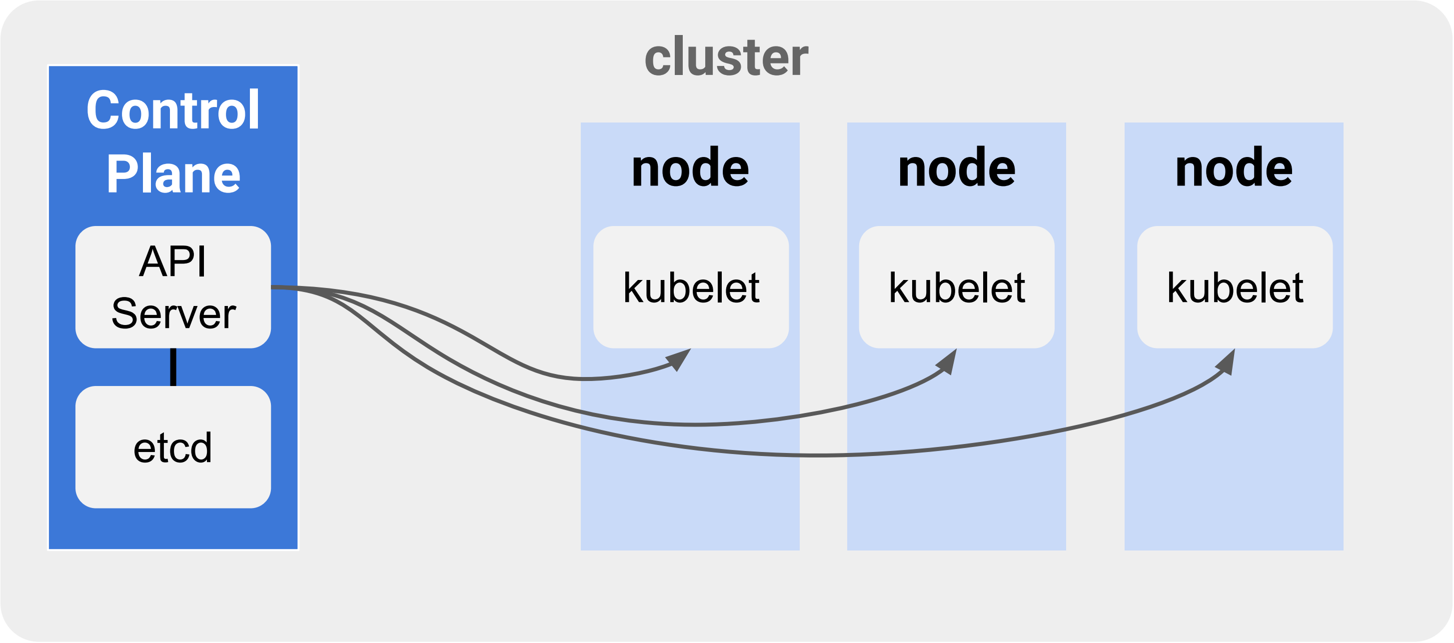
Kubernetes Architecture

- Kubernetes applications require a cluster of machines to run on
  - One or more machines are designated as the control plane
  - Multiple machines are added to the cluster as nodes
- The control plane and nodes can be physical or virtual machines



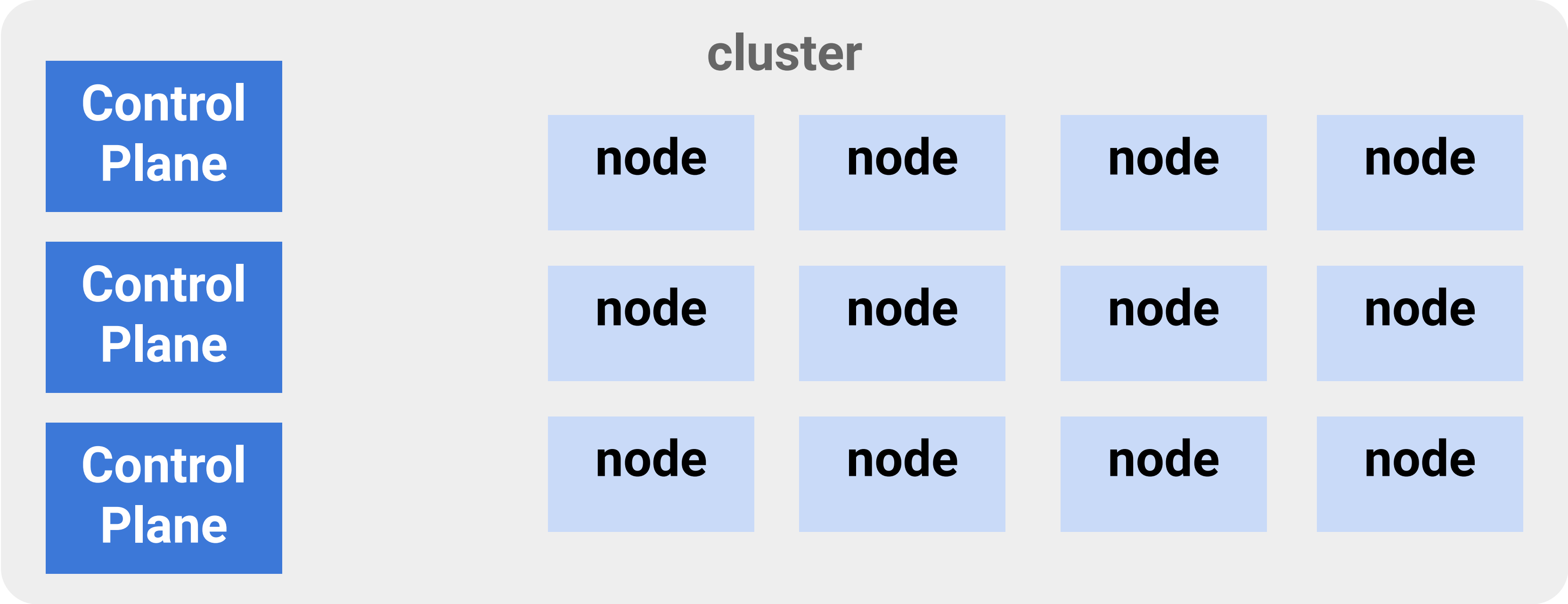
Kubernetes Clusters

- The control plane ensures applications are running on the nodes
  - Maintains the desired state of the cluster
- The nodes provide the computing and storage required by the applications
  - Also known as the data plane



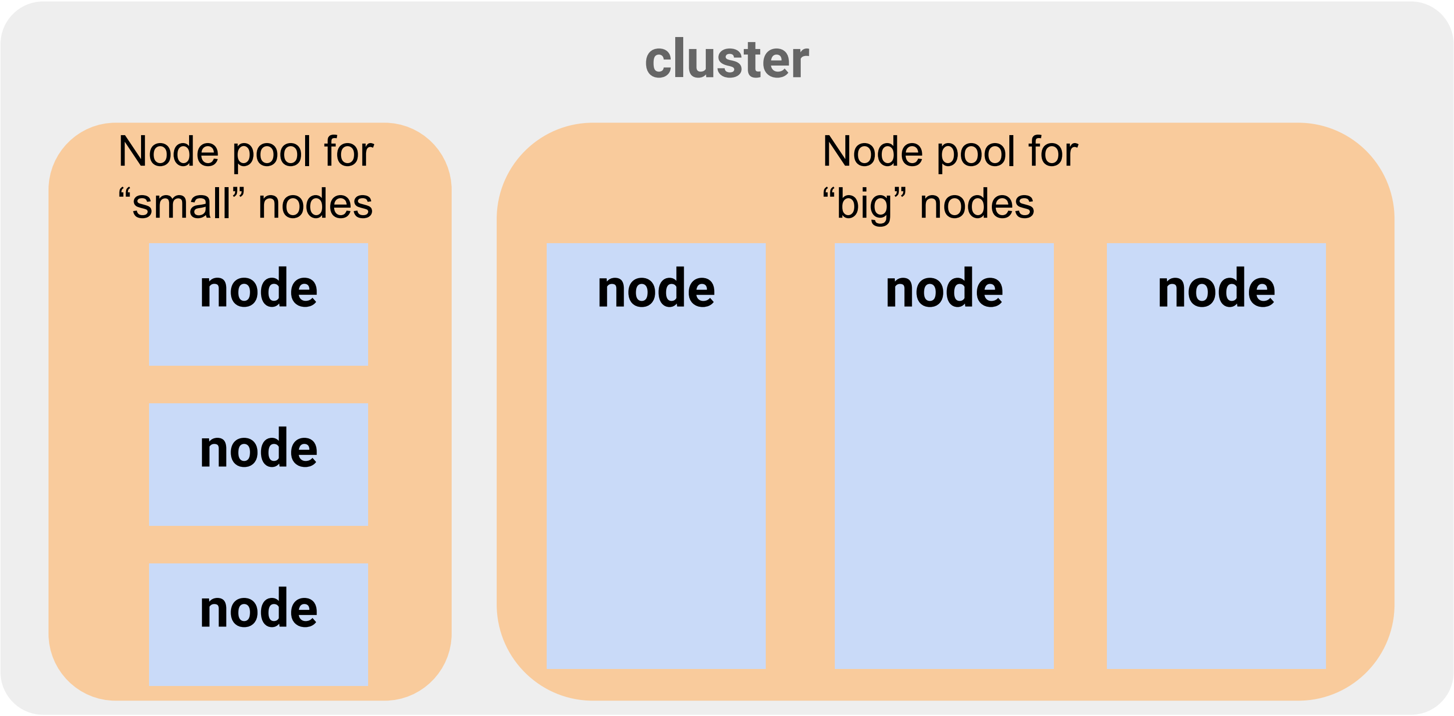
Kubernetes Clusters (continued)

- The control plane can be deployed as highly available
  - Multiple copies
- Clusters can also have many nodes



Node Pools

- All nodes in a node pool must be the same configuration
  - RAM, CPU, disk type, etc.
- Can create multiple node pools in a single cluster to manage different kinds of nodes



Manually Creating Kubernetes Clusters

- Steps (***this is vastly over-simplified!***)
  1. Buy some computers and install Linux on them
  2. Install Kubernetes:
     ```
     $ apt-get install -y kubeadm=#.#.# kubelet=#.#.# kubectl=#.#.#
     ```
  3. Configure the control plane:
     ```
     $ kubeadm init --kubernetes-version #.#.# --pod-network-cidr
     192.168.0.0/16 …
     ```
  4. Grow the cluster by adding nodes:
     ```
     kubeadm join --token ... --discovery-token-ca-cert-hash ...
     ```
  5. Hire a couple IT people to administrate the cluster
- Or use a managed service to automate cluster creation

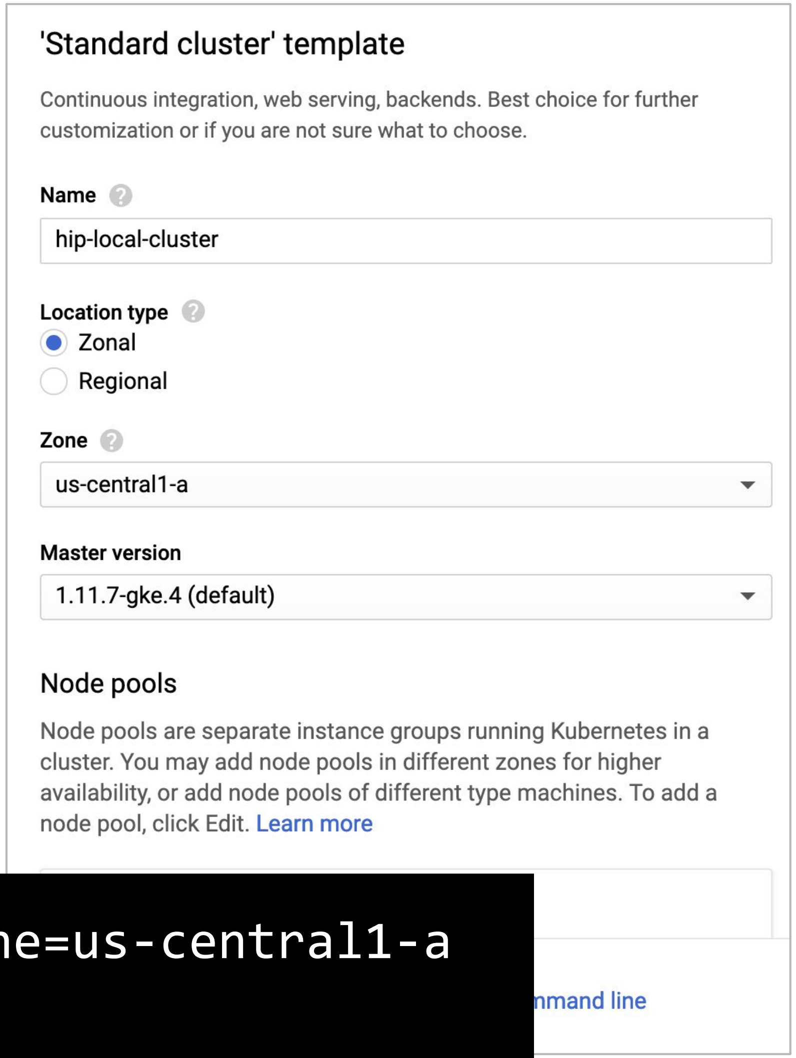
Chapter Concepts

Kubernetes Architecture

Managed Clusters

Google Kubernetes Engine (GKE)

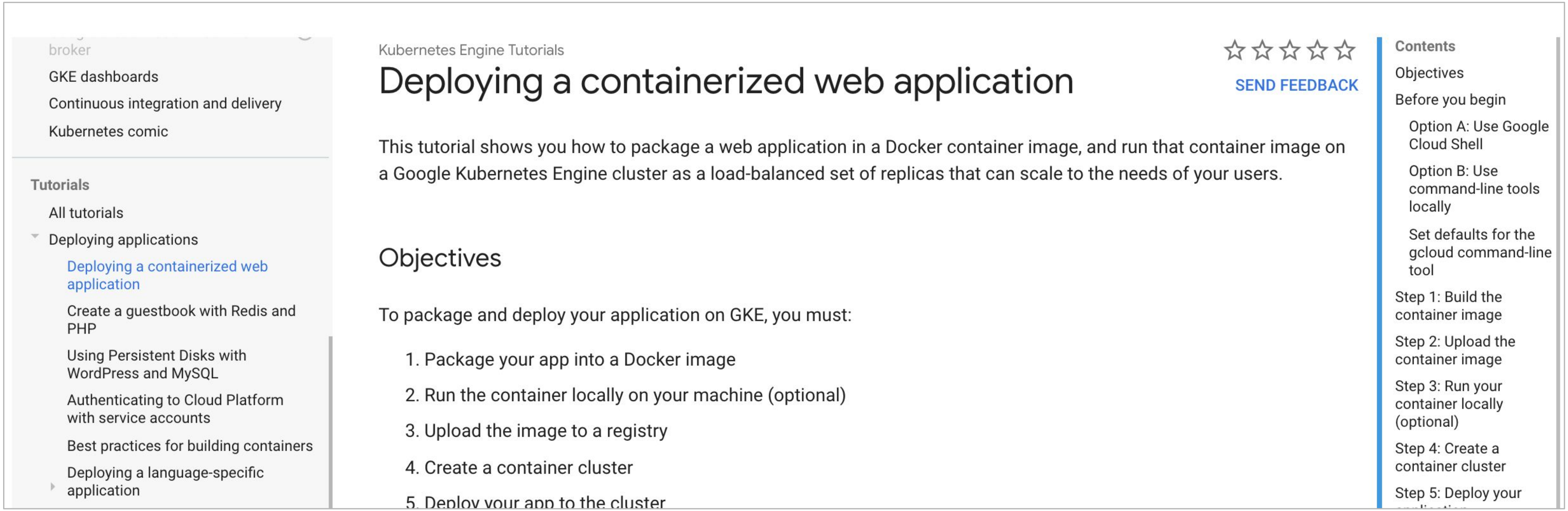
- Google's service for providing Kubernetes clusters
  - Automates cluster creation and maintenance
  - Workers are implemented as a collection of Compute Engine VMs
  - Control plane is a managed service
    - Not visible in compute engine
- Very simple creation using the web console or command line




```
$ gcloud container clusters create my-cluster --zone=us-central1-a
--project=my-project-id
```

Tutorial: Getting Started with GKE

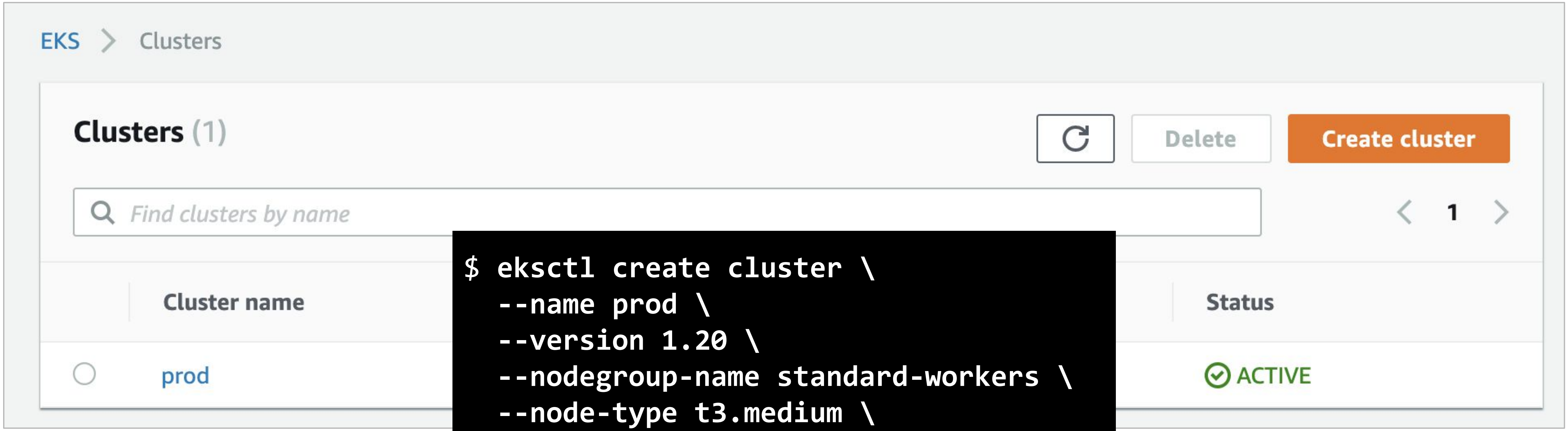
- https://cloud.google.com/kubernetes-engine/docs/tutorials/hello-app



AWS Elastic Kubernetes Service (EKS)

- Amazon's service for providing Kubernetes clusters
  - Workers are implemented as a collection of EC2 VMs
  - Control plane is a managed service
    - No visible in EC2
- Clusters can be easily created and managed using the console or the `eksctl` CLI tool

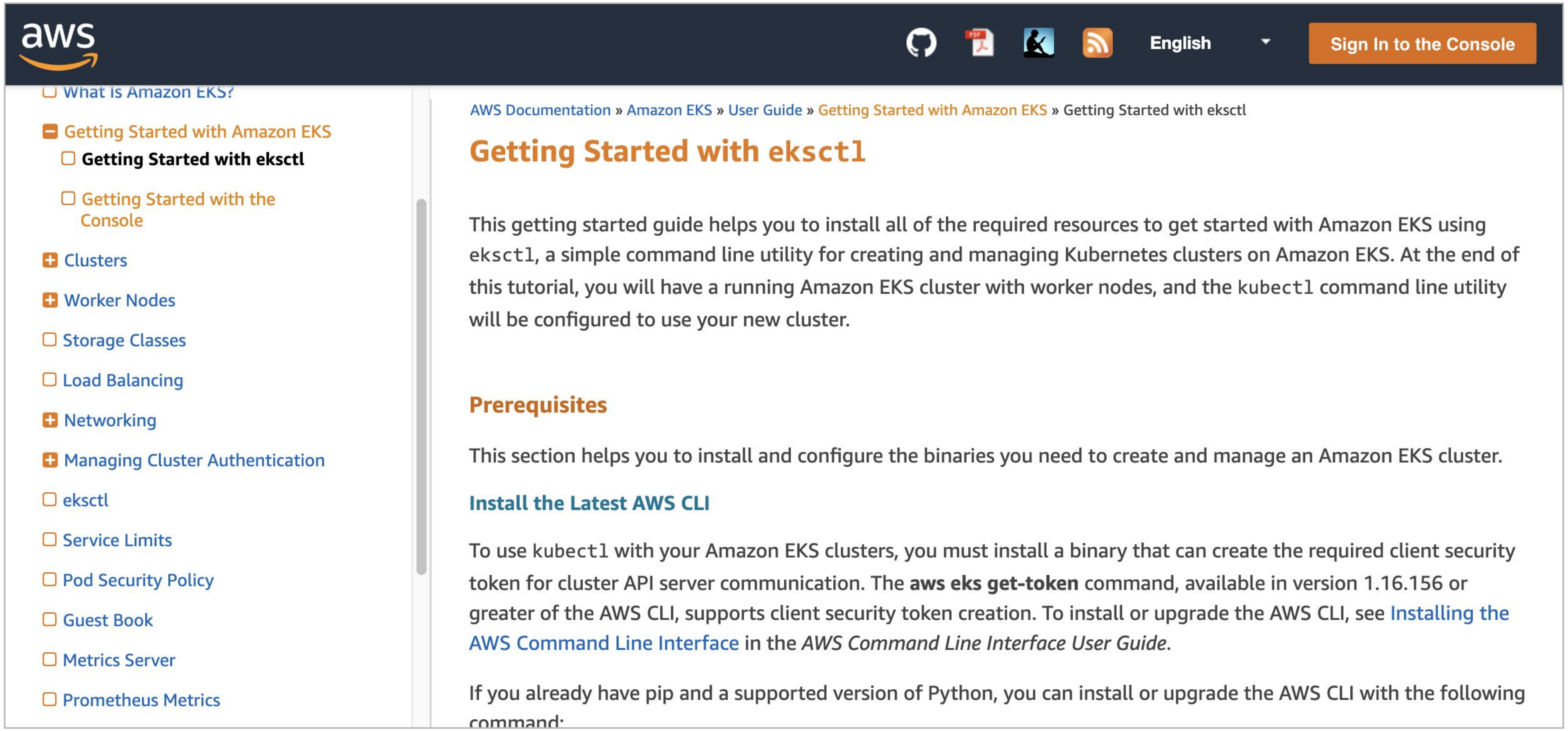
AWS Elastic Kubernetes Service (EKS) (continued)



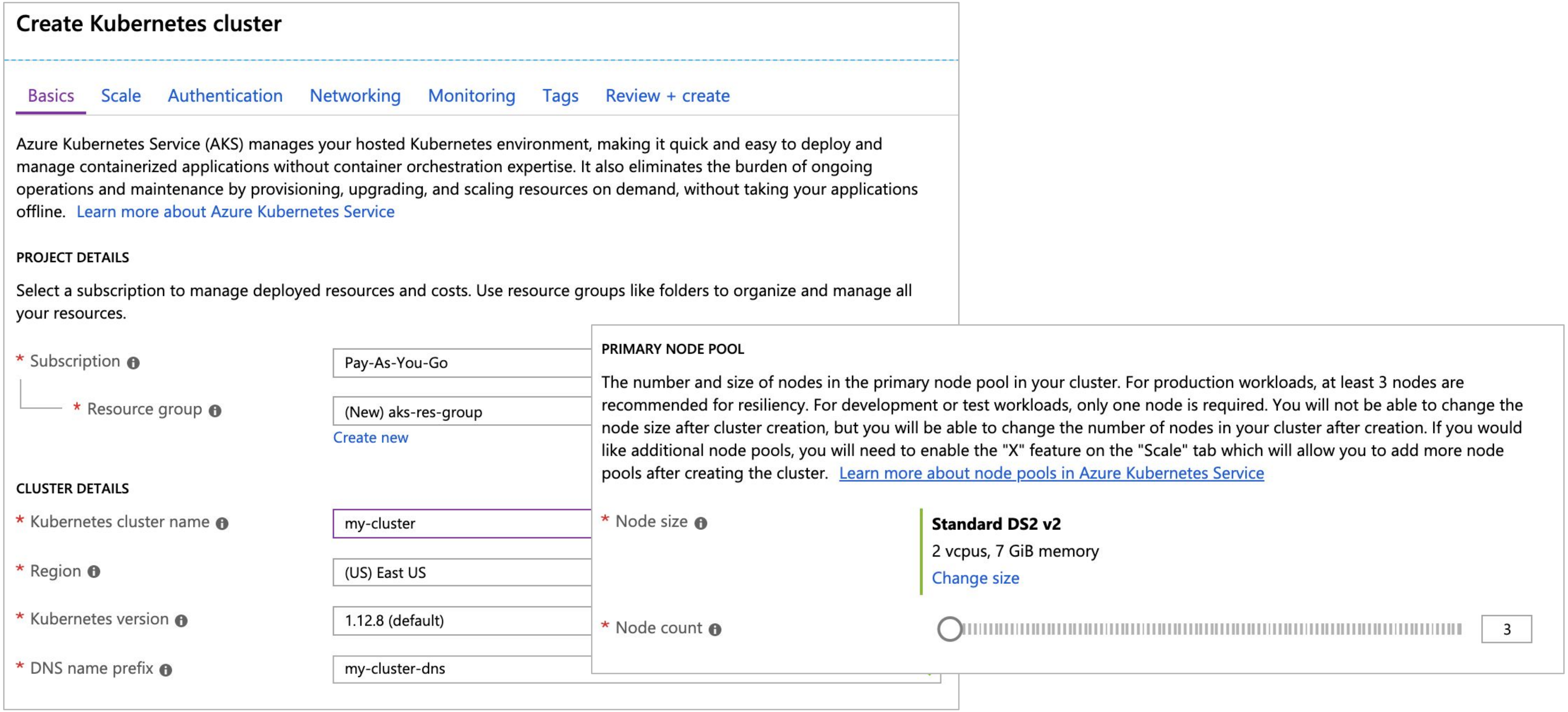

```
$ eksctl create cluster \
  --name prod \
  --version 1.20 \
  --nodegroup-name standard-workers \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --node-ami auto
```

Tutorial: Getting Started with EKS

- https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html

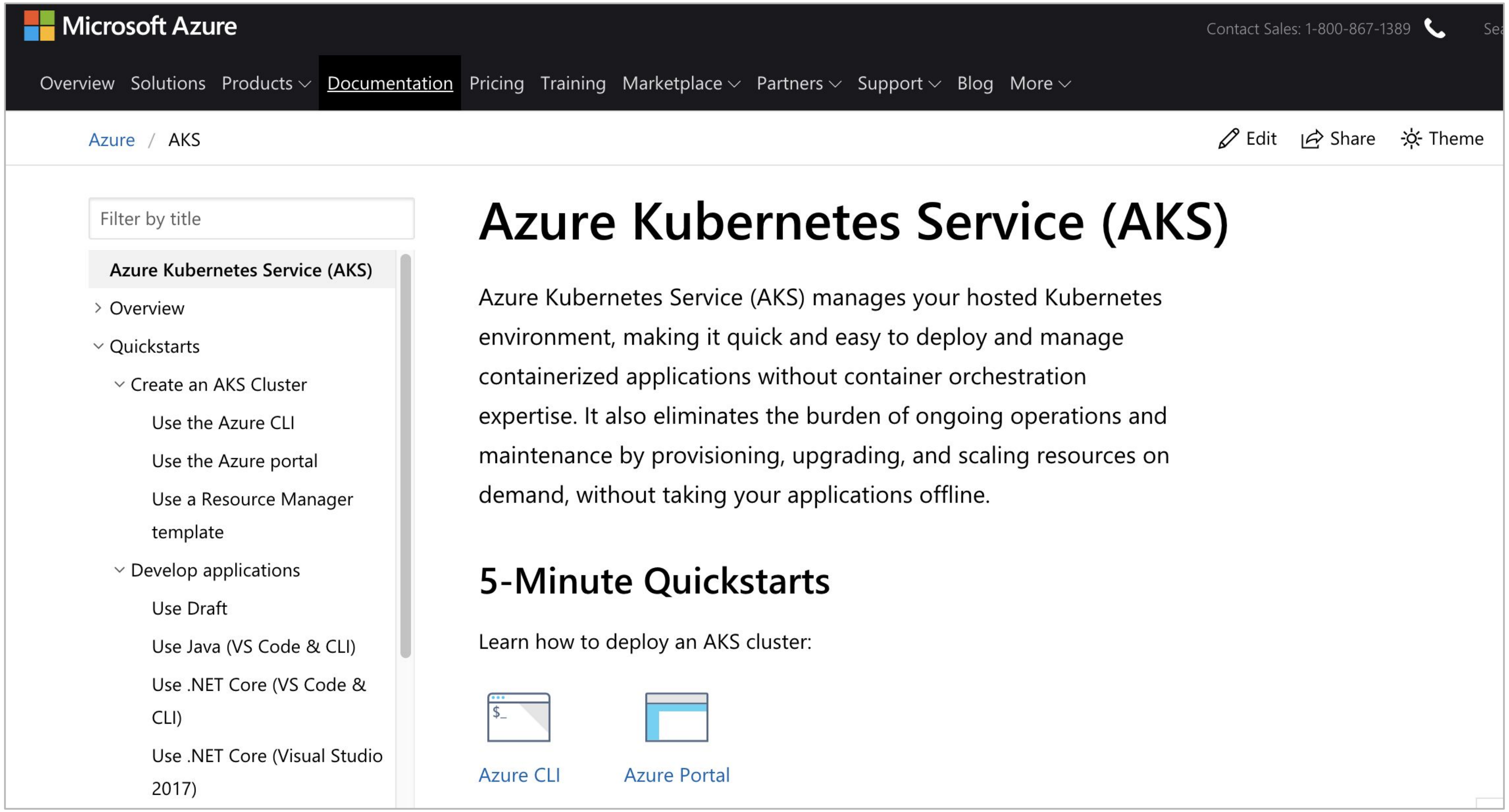


Azure Kubernetes Service (AKS)



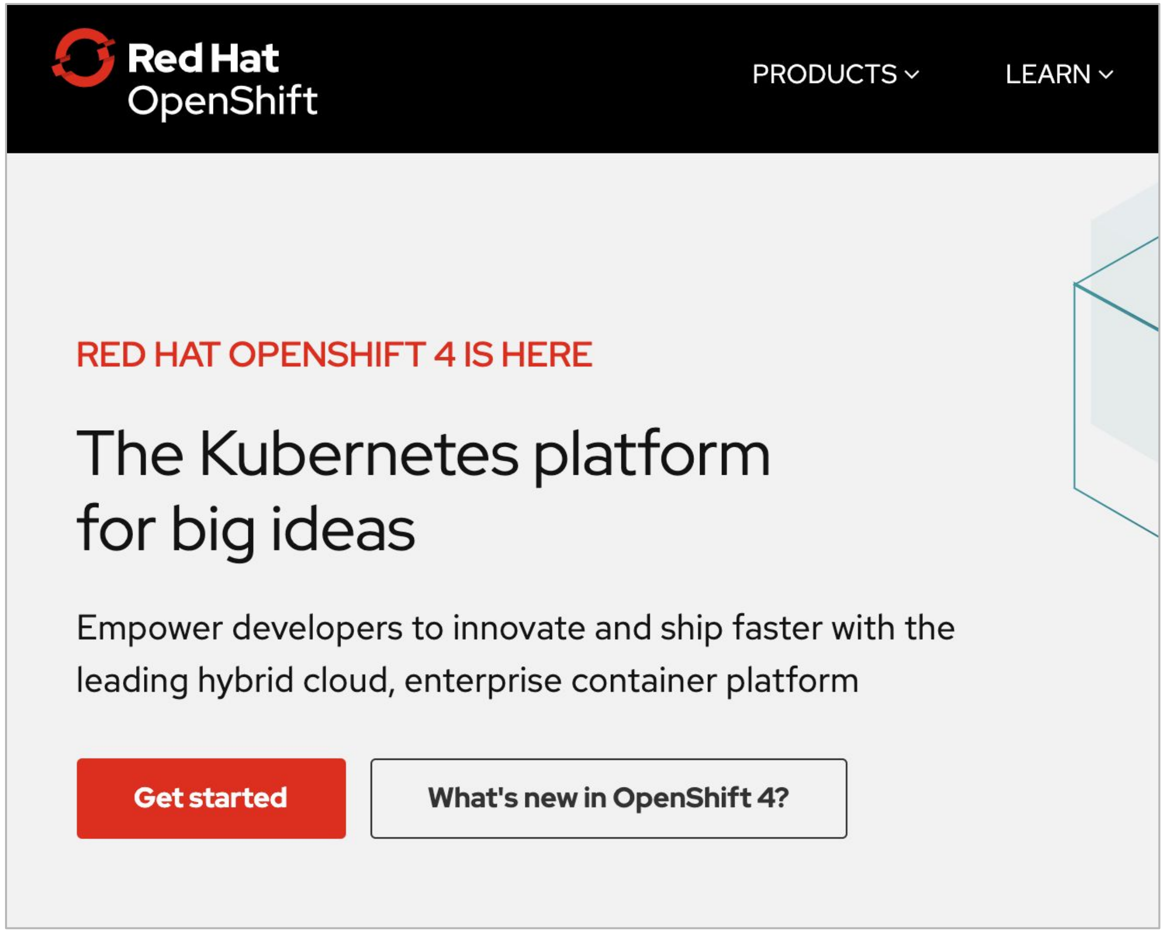
Tutorial: Getting Started with AKS

- https://docs.microsoft.com/en-us/azure/aks/



OpenShift

- Manages Kubernetes that runs on Red Hat OpenStack
  - Useful for private cloud deployments
  - Online, managed version is available



MiniKube

- Minikube is a free, open-source tool for running a Kubernetes cluster locally
  - Runs on Linux, Mac, and Windows
  - Useful for development and testing
- See: https://github.com/kubernetes/minikube
- Tutorial: https://kubernetes.io/docs/tutorials/hello-minikube/
- MiniKube is installed in Google cloud shell by default
  - Just type: `minikube start`

Activity: Creating Kubernetes Clusters

In this activity, you will:

- Create a Kubernetes cluster in the cloud

Chapter Summary

In this chapter, you have:

- Reviewed the Kubernetes architecture
- Configured and created a Kubernetes managed cluster to host containers

Kubernetes Architecture and CLI

Chapter Objectives

In this chapter, you will:

- Understand the Kubernetes application architecture
- Configure and use the `kubectl` CLI

Chapter Concepts

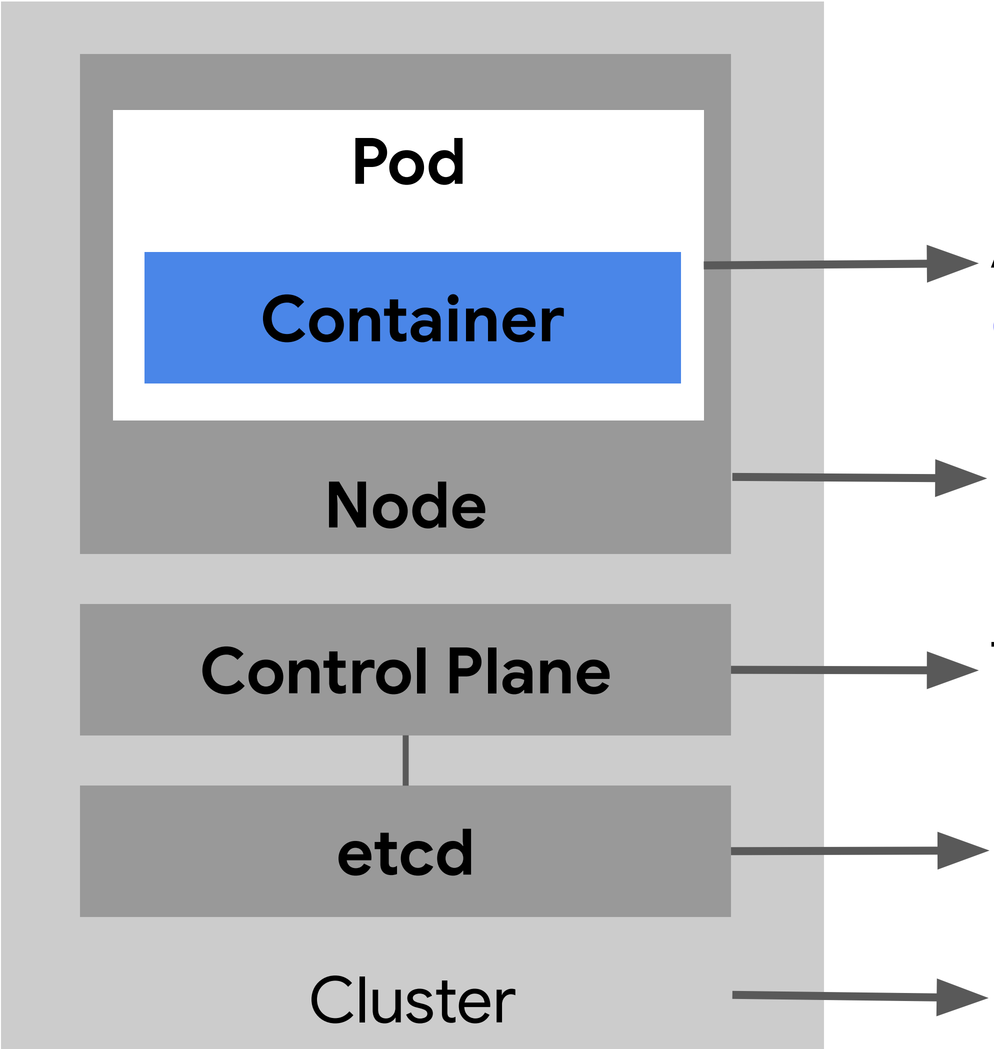
Kubernetes Application Architecture

`kubectl` CLI

Kubernetes Terms

- **Pods** are the smallest unit of deployment
  - Usually pods represent a single container
  - Kubernetes manages the pods rather than the containers directly
- **Clusters** are collections of machines that will run pods
  - Each machine is a node in the cluster
- **Deployments** are configurations that define service resources
- **ReplicaSets** are used to create multiple instances of a pod
  - Guarantee pods are healthy and the right number exist
- **Services** expose multiple pods as a consistent endpoint
  - Load balancers route requests to pods
- **Autoscalers** monitor load and turn pods on or off

Kubernetes Components



A **pod** is a collection of one or more **containers**, deployed on nodes

Nodes are machines that run the containers, talk to the control plane

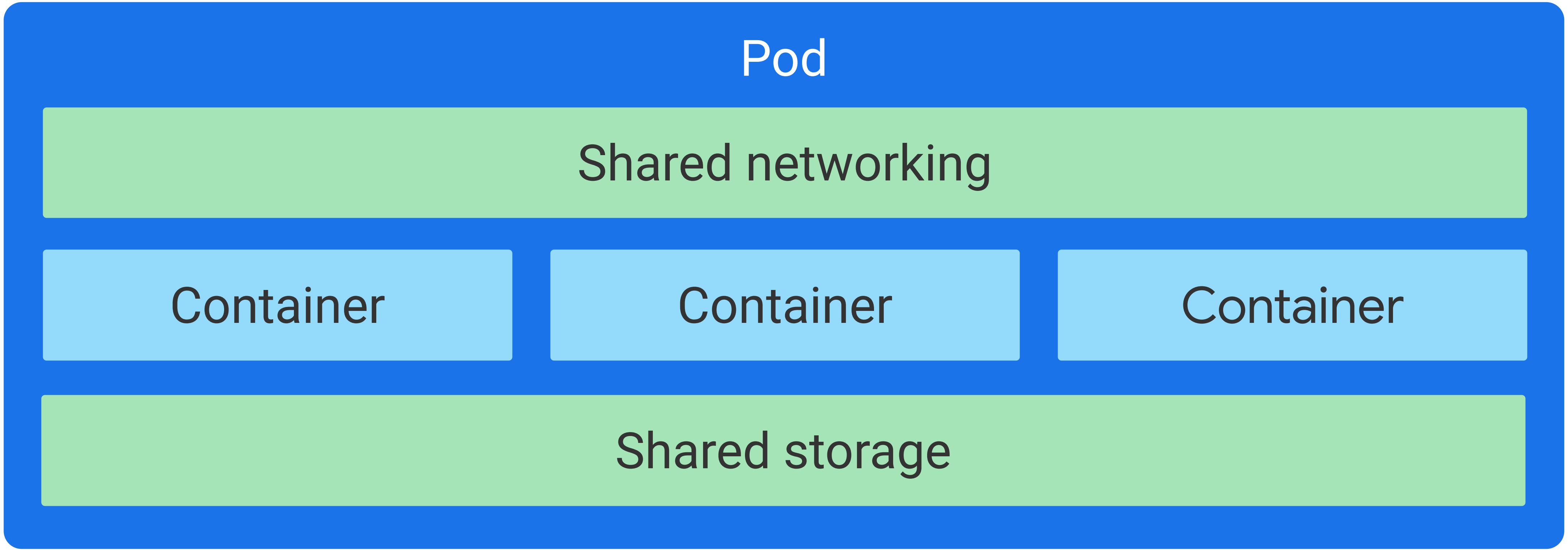
The **control plane** controls the cluster

etcd is for storing cluster state

A **cluster** is a set of nodes

Pods

- A pod is one or more containers deployed as a single unit on a node
  - Share networking, namespaces, and storage



Pods (continued)

- In hypervisor virtualization, a single VM can host multiple apps
  - Each VM can have its own IP address
  - All apps on a VM share the VM's address and port space
- A pod is a similar concept
  - Each pod gets its own IP address
  - All containers in a pod share the pod's address and port space
    - Containers in the same pod can talk to each other using `localhost`
- However, it is very common to just have one container per pod
  - The pod is just a wrapper around a single container

Deploying Pods

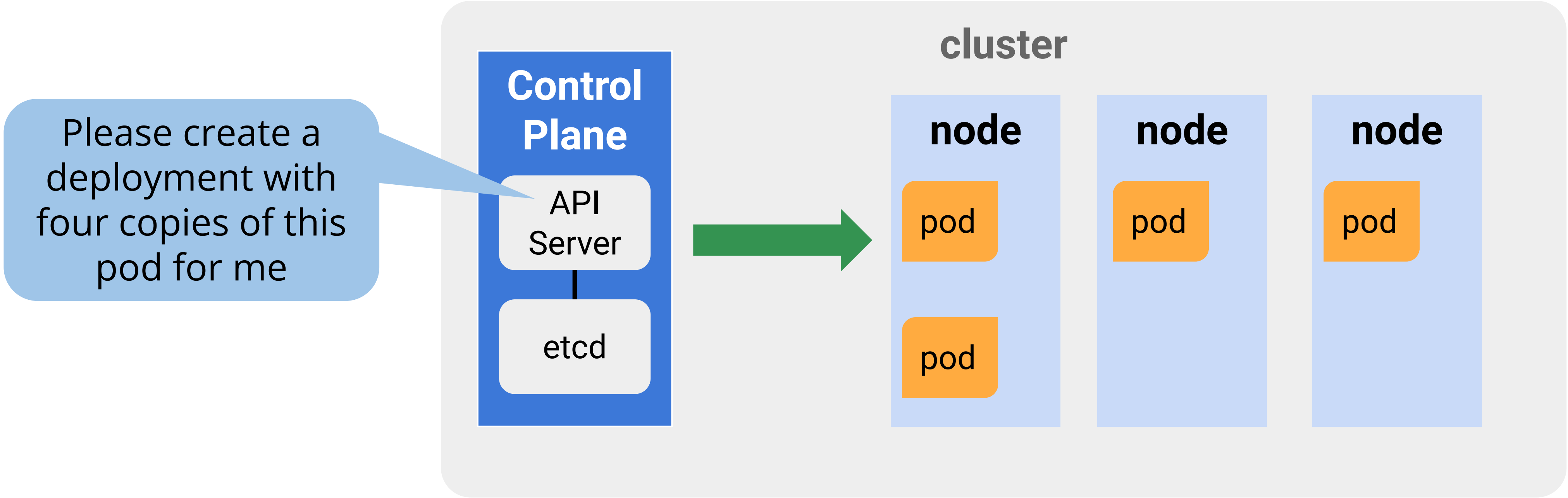
- Pods are designed to be very short-lived
  - Ephemeral and disposable
  - Pods do not automatically heal or repair themselves
- Also, if a node fails or the scheduling operation fails, the pods are not replaced
- Better to deploy pods in a fault tolerant manner
  - Using Kubernetes deployments

Kubernetes Deployments

- A deployment is better for long-lived services
  - A **deployment** is what Kubernetes calls a **controller** object
  - Manages and maintains (controls) the desired state of the pods
  - Ensures that a defined set of pods is running at any given time
    - Using something called a **ReplicaSet**
  - Deployments do much more—we will see later …
- Can specify many properties including:
  - The containers to put within the pod
  - How many copies (replicas) of the pods to deploy
  - Labels to “tag” the pods
  - And more …

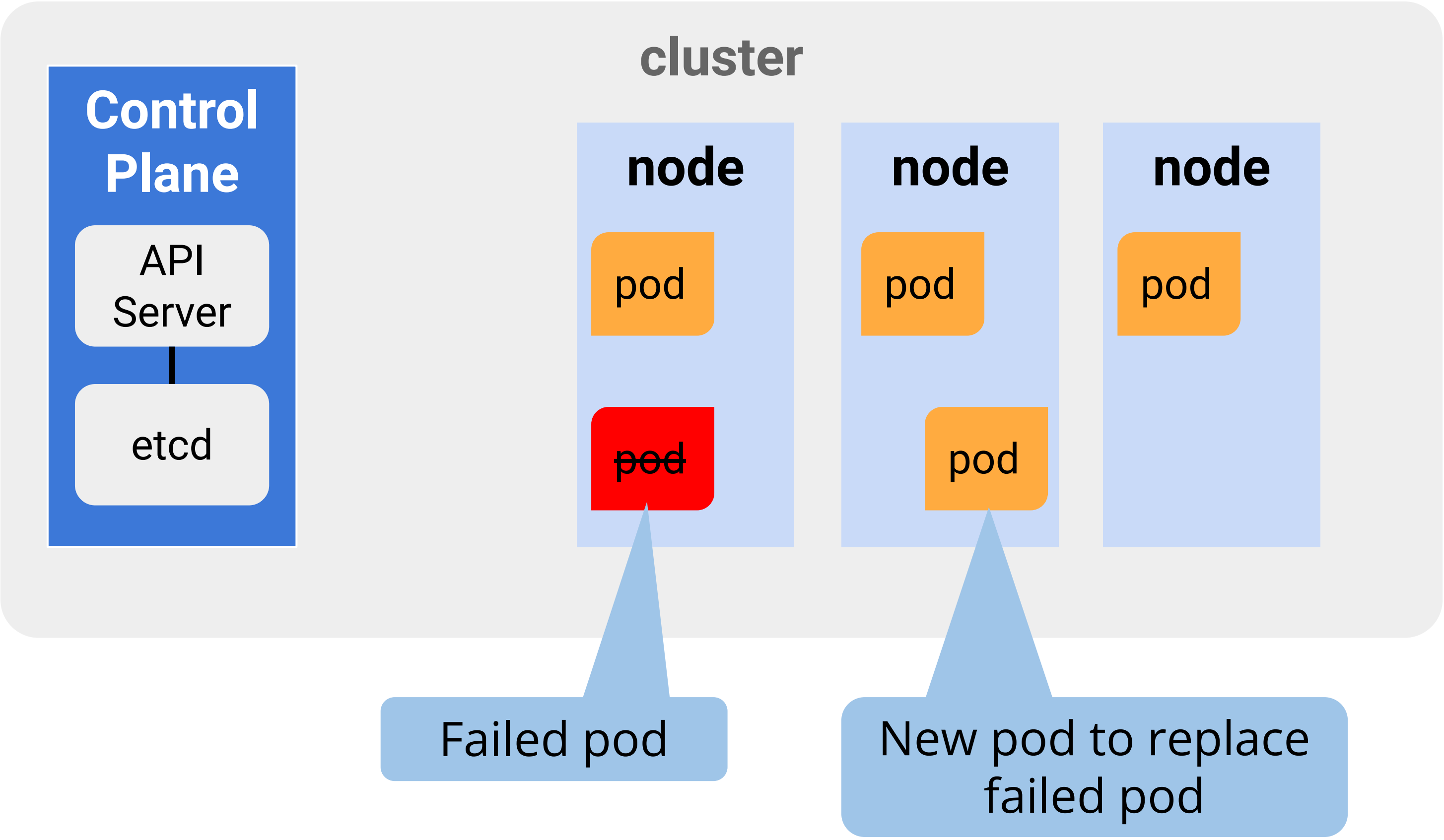
Kubernetes Deployments (continued)

- Assume a deployment with four replicas



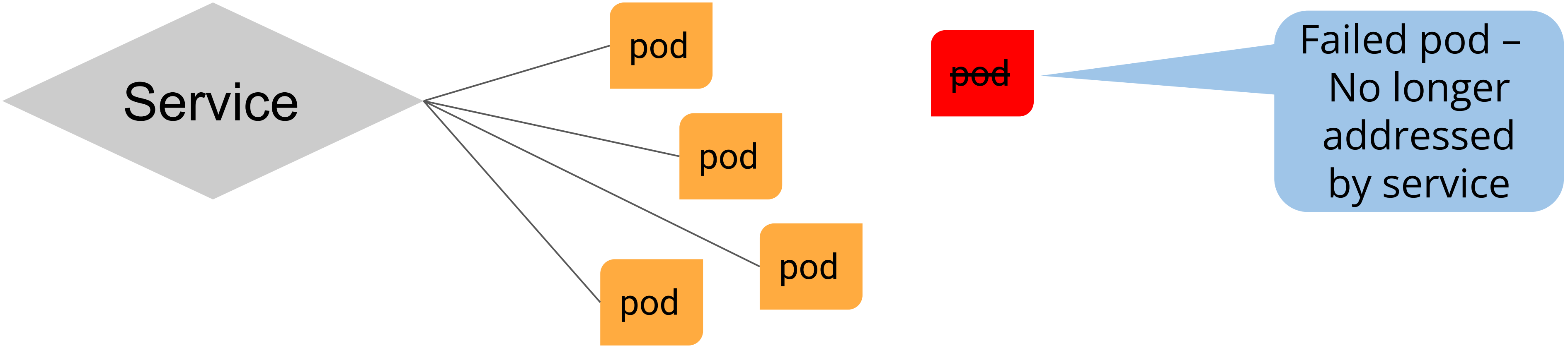
What Happens If a Pod Fails?

- The ReplicaSet replaces any failed pod



Kubernetes Service

- A just seen, a single deployment can have multiple pod replicas
  - Each pod has a different address
  - Pods can come and go quickly (the replicaset replaces failed pods)
  - Their address can change
  - Their location can change
- A service is used to expose a deployment as a consistent endpoint
  - For example: a load balancer is a type of service



Kubernetes Namespaces

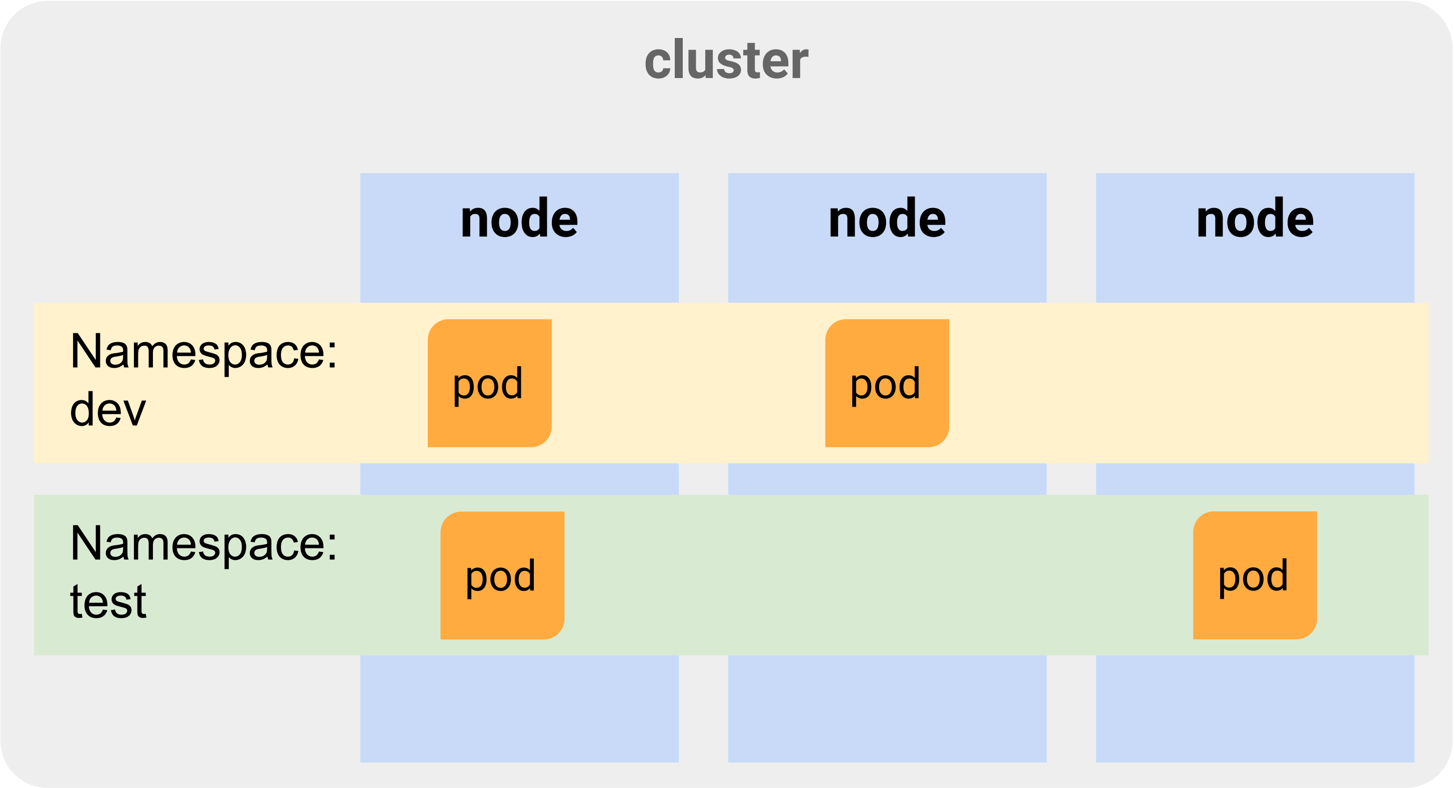
- Kubernetes clusters are designed to be shared with different pods, projects, environments, etc.
  - To get better resource utilization of the cluster
- Namespaces allow a single Kubernetes cluster to be divided into multiple “virtual clusters”
  - Can be used to divide cluster resources between multiple users
  - Multiple namespaces inside a single Kubernetes cluster are logically isolated from each other
  - Namespaces are intended for use in environments with many users spread across multiple teams or projects
  - Can help with organization, security, and even performance

Default Namespaces

- By default, Kubernetes clusters have four namespaces
  - **default:** The default namespace for objects
  - **kube-public:** For objects that should be visible and readable publicly throughout the whole cluster
  - **kube-system:** Should be left alone
  - **kube-node-lease**: Should be left alone
- Additional namespaces can be created as needed
  - Will be done later

Kubernetes Namespaces

- Namespaces provide scope for naming resources such as pods, deployments, and controllers



Chapter Concepts

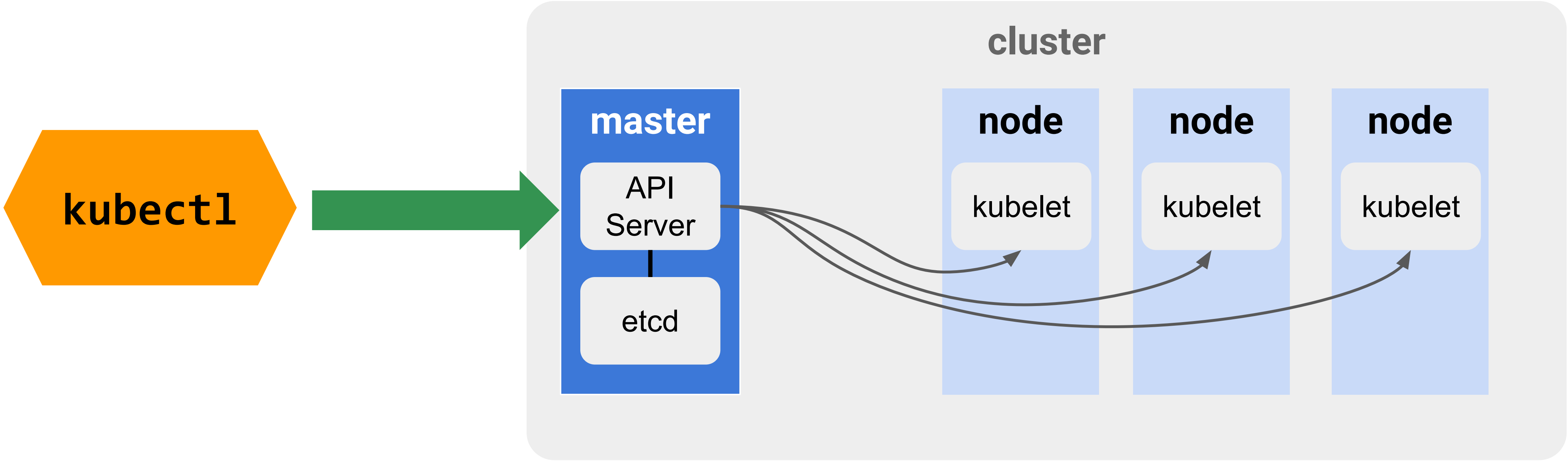
Kubernetes Application Architecture

`kubectl` CLI

Kubernetes CLI

- The API server on the control plane is the gateway to the Kubernetes cluster
  - Implements a RESTful API over HTTP
  - Performs all API operations
- Kubernetes can be automated with a combination of CLI commands and configuration code
  - `kubectl` is the Kubernetes CLI
- Provided by the Cloud Native Computing Foundation
  - Works with any Kubernetes cluster
  - See: https://kubernetes.io/

Kubernetes CLI (continued)



`kubectl` Command	Description
`kubectl get nodes`	Show information about the cluster nodes
`kubectl create -f config.yaml` `kubectl apply -f config.yaml`	Create resources specified in the config.yaml
`kubectl get pods` `kubectl describe pods`	Show the running pods
`kubectl get deployments` `kubectl describe deployments`	Show the deployments
`kubectl get hpa`	Show horizontal pod autoscalers along with their min, max, and resource metrics
`kubectl get services`	Show running services along with their addresses and ports
`kubectl delete [name]` `kubectl delete deployments/spaceinvaders`	Delete specified resources
`kubectl exec [options] [pod-name]` `kubectl exec -it spaceinvaders-55c88695bb-bcxb2 -- /bin/bash`	Execute a command in the container (the example runs a bash shell)

Configuring `kubectl`

- `kubectl` must be configured to communicate to a cluster
  - Location and credentials for the cluster
  - Information is stored in the `$HOME/.kube/config file`
- When using Google Kubernetes engine, config file is created with:

  `gcloud container clusters get-credentials <cluster-name> --zone <zone-name>`
- AWS EKS provides a similar command:

  `aws eks --region <region-name> update-kubeconfig --name <cluster_name>`
- Azure AKS:

  `az aks get-credentials --name <cluster_name> --resource-group <group-name>`

The `kubectl` Syntax

What do you want to do? | On which object type?

`kubectl <command> <type> <name> <flags>`

<command>	<type>
get	pods
describe	deployments
logs	services
exec	nodes
delete	hpa
. . .	. . .

Pods Can Be Defined in Configuration Files

- Pods are single instance
  - Remember, if a pod fails, it is not replaced automatically

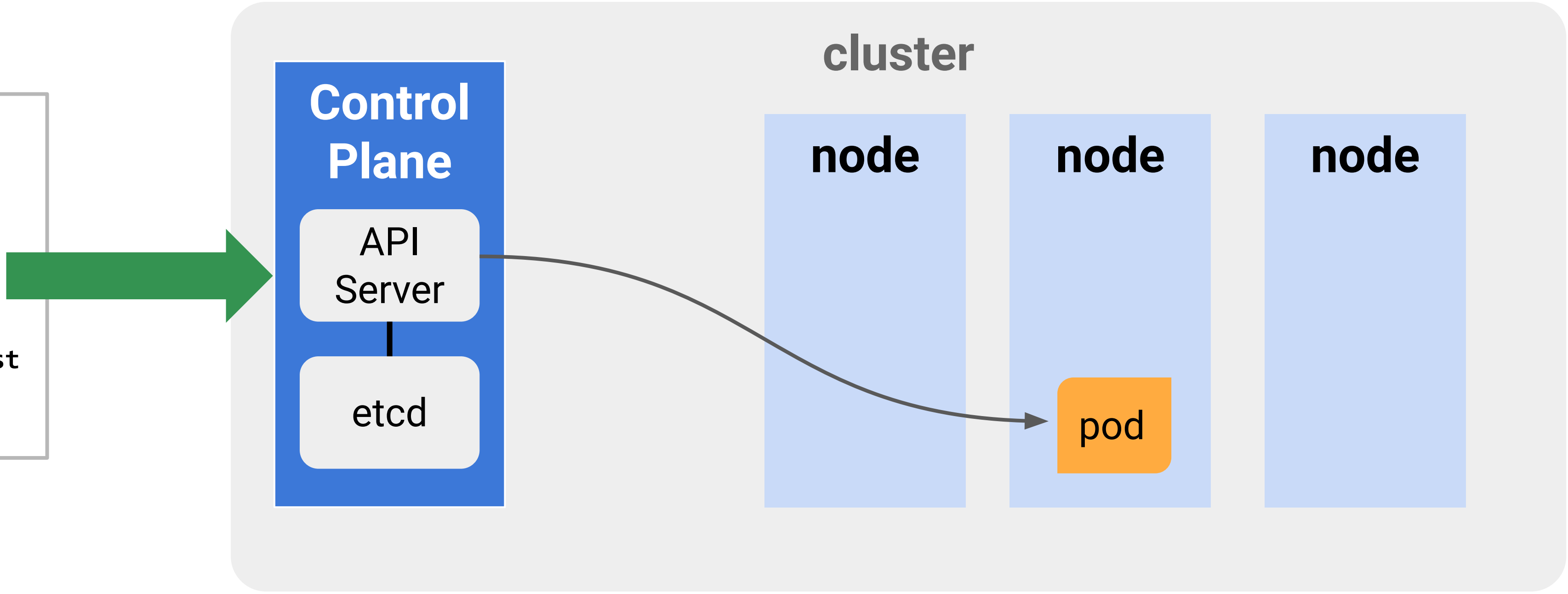
```
apiVersion: v1
kind: Pod
metadata:
  name: devops-pod
spec:
  containers:
  - name:  devops-demo
    image: drehnstrom/devops-demo:latest
    ports:
    - containerPort: 8080
```

Creating a Pod

- A pod can be created by applying a configuration file

```
kubectl apply -f pod-config.yaml
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: devops-pod
spec:
  containers:
  - name:  devops-demo
    image: drehnstrom/devops-demo:latest
    ports:
    - containerPort: 8080
```



Retrieving Information

- Information about the cluster and objects deployed can be retrieved

```
kubectl cluster-info
```

```
kubectl get nodes
```

```
kubectl get pods
```

```
kubectl describe pods
```

Executing Commands Within a Pod

- It is possible to execute commands in a container within a pod
  - Useful to examine the contents, state, or environment of a container
  - Can also be used for debugging and testing

```
kubectl exec <pod-name> -- <command>
```

```
kubectl exec devops-pod -- ls /
kubectl exec devops-pod -- ps axu
```

- If a pod has more than one container:
  - Specify container name with `-c` or `--container`:

```
kubectl exec devops -c devops-demo -- ls /
```

Opening a Shell Prompt in a Container

- It is also possible to open an interactive shell into the container
  - Using the options `-i (--stdin)` and `-t (-- tty)`

```
kubectl exec -i -t <pod-name> -- /bin/bash
```

  - Use the `-c` option if there is more than one container:

```
kubectl exec -i -t devops -c devops-demo -- /bin/bash
```

If You Need to Debug a Pod, You Can Execute a Command Inside It

If something is wrong, run a bash shell in a pod

```
$ kubectl exec -it spaceinvaders-pod -- /bin/bash
root@spaceinvaders-pod:/# curl http://localhost/
<!DOCTYPE html>

  OUTPUT OMITTED

</body>
</html>
root@spaceinvaders-pod:/# exit
$
```

Kubernetes Logging

- Kubernetes captures the standard output (stdout) and standard error output (stderr) from each container on the node to a log file
  - This file is managed by Kubernetes
  - Usually restricted to the last 10MB of logs
- Logs can be viewed using the `kubectl` command
  - Viewing container logs

```
kubectl logs <pod-name>
```

Accessing Pod Logs

- Viewing just the most recent 20 lines

```
kubectl logs --tail=20 devops
```

- Viewing just the most recent 3 hours

```
kubectl logs --since=3h devops
```

- If you want to access logs of a crashed instance, you can use `--previous`

Scalable Kubernetes Logging and Monitoring

- Managing cluster logs can get difficult
  - When the number of applications increases and the cluster runs on multiple machines
  - A more scalable solution is needed
- Kubernetes does not provide a native solution for cluster-level logging
- There are several common approaches to solve this problem
  - Use a node-level logging agent that runs on every node
  - Include a dedicated sidecar container for logging in an application pod
  - Push logs directly to a backend from within an application

Scalable Kubernetes Logging and Monitoring (continued)

- Most managed Kubernetes clusters will provide integrated cluster-level logging services
  - Google Kubernetes Engine is integrated with Google Cloud Operations Suite
  - AWS EKS is integrated with AWS Cloud Watch
  - Azure Kubernetes Service uses the Azure Monitor

Custom Logging

- Splunk is a very popular tool for processing logs and metrics
- Splunk Connect for Kubernetes provides a way to import Kubernetes logging, object, and metrics data into Splunk
  - Works by installing a fluentd container on each node
  - https://github.com/splunk/splunk-connect-for-kubernetes

Deleting a Pod

- Pods can be easily deleted
- Can delete resources by passing in the configuration file used to create them

```
kubectl delete -f pod-config.yaml
```

- Or delete the resource directly

```
kubectl delete pod <pod-name>
```

Activity: Using the `kubectl` CLI

In this activity, you will:

- Configure the `kubectl` CLI for your cluster
- Deploy a pod to your cluster with the `kubectl` CLI
- Investigate various `kubectl` commands to interact with your cluster
- Execute commands within a container running in a pod
- Delete pods

Chapter Summary

In this chapter, you have:

- Understood the Kubernetes application architecture
- Configured and used the `kubectl` CLI

Deployments and Services

Chapter Objectives

In this chapter, you will:

- Create deployments
- Leverage labels to identify different types of pods in a cluster
- Expose deployments as a consistent endpoint with services
- Control pod resource limits
- Monitor pods with aliveness and readiness probes
- Perform pod autoscaling

Chapter Concepts

Deployments

Services

Resource Requests and Limits

Fault Tolerance and Scalability

Creating Deployments

- Deployments can be created several ways

a.

```
kubectl create deployment <deployment-name> \
--image <image-name:tag> \
--replicas 4 \
--labels <key=value> \
--port <port-number> \
--generator deployment/apps.v1
```

b.

```
kubectl apply -f <deployment-file>
```

c. Cloud provider specific dashboards

Deployment File in YAML Format

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: space-invaders-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: space-invaders
  template:
    metadata:
      labels:
        app: space-invaders
    spec:
      containers:
      - image: coursedemos/spaceinvaders:v1
        name: space-invaders-container
        ports:
        - containerPort: 8080
```

Kind of resource: Deployment

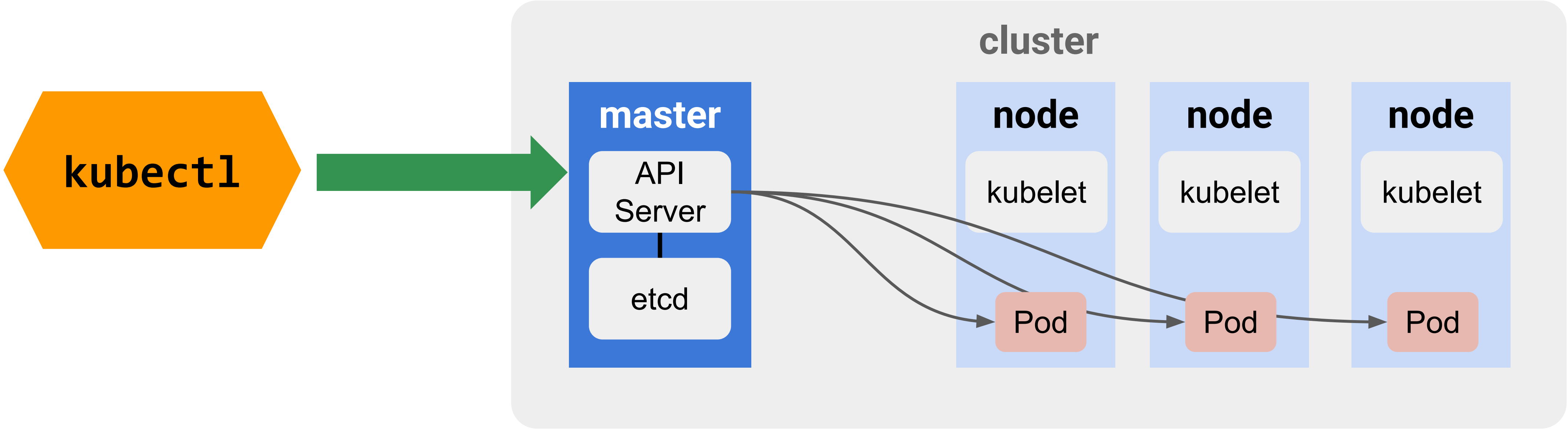
We want 3 replicas of the pod

This spec defines the pod. The same as the pod configuration.

Kubernetes Deployments

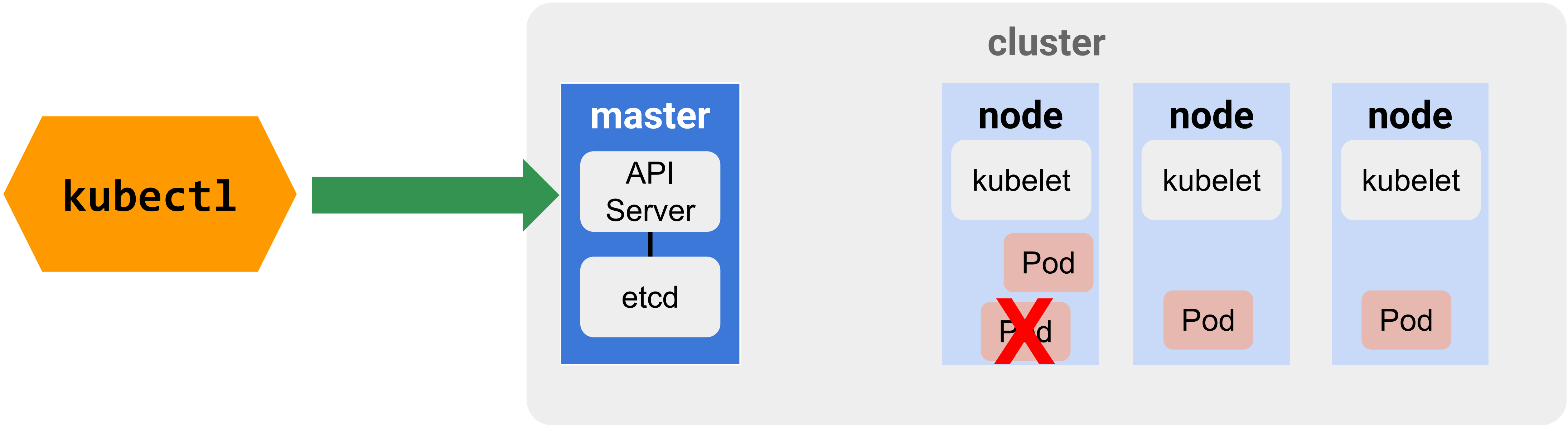
- Use `kubectl` to send the config file to the master
  - The master then decides how to deploy the pods
  - Deployments combine pods with replica sets

```
kubectl apply -f kubernetes-config.yaml
```



Kubernetes Replica Set

- A replica set ensures the correct number of pods is running
  - And replaces any pod that fails



Creating a Deployment from a Configuration File

- Deploy to a cluster based on a configuration file

```
kubectl apply -f kubernetes-config.yaml
```

- Show the running pods

```
kubectl get pods
```

- Show all the deployments

```
kubectl get deployments
```

- Show details of a deployment

```
kubectl describe deployments devops-deployment
```

- Show details of a pod

```
kubectl describe pod devops-demo-pod
```

Deployments Can Be Scaled Manually

- Modify the YAML file and apply it again
  - The replica set will be updated to the new number of replicas

```
kubectl apply -f <deployment-file>
```

```
...
spec:
  replicas: 5
...
```

  - Or issue a scale command

```
kubectl scale deployment <deployment-name> --replicas=5
```

- Autoscaling will be covered later

Kubernetes Labels

- Labels are key/value pairs that can be attached to any API object
  - Used to specify identifying attributes
    - Name, environment, tier, version, etc.
- Searched and selected by Selectors
  - Allows resources to be identified independently from address, location, or other property that can change
- Labels do not directly imply any meaning to the Kubernetes system

Label Examples

- Develop your own labels and naming conventions
- Some examples:
  - `env: dev` `env: test` `env:prod`
  - `version: v1` `version: v2.0.0` `version: v3.1`
  - `tier: frontend` `tier: backend` `tier: data`
- Objects can have multiple values

```
kind: Deployment
metadata:
  name: devops-deployment
  labels:
    app: webui
    tier: frontend
    env: prod
```

Label Requirements

- Labels have a name and an optional value
  - Names and values must begin and end with an alphanumeric character
  - Can contain dashes (-), underscores (_), dots (.), and alphanumerics
- The name must be between 1 and 63 characters long
  - The name can also have an optional prefix up to 253 characters ending with a slash (/)
- The value must be between 0 and 63 characters long

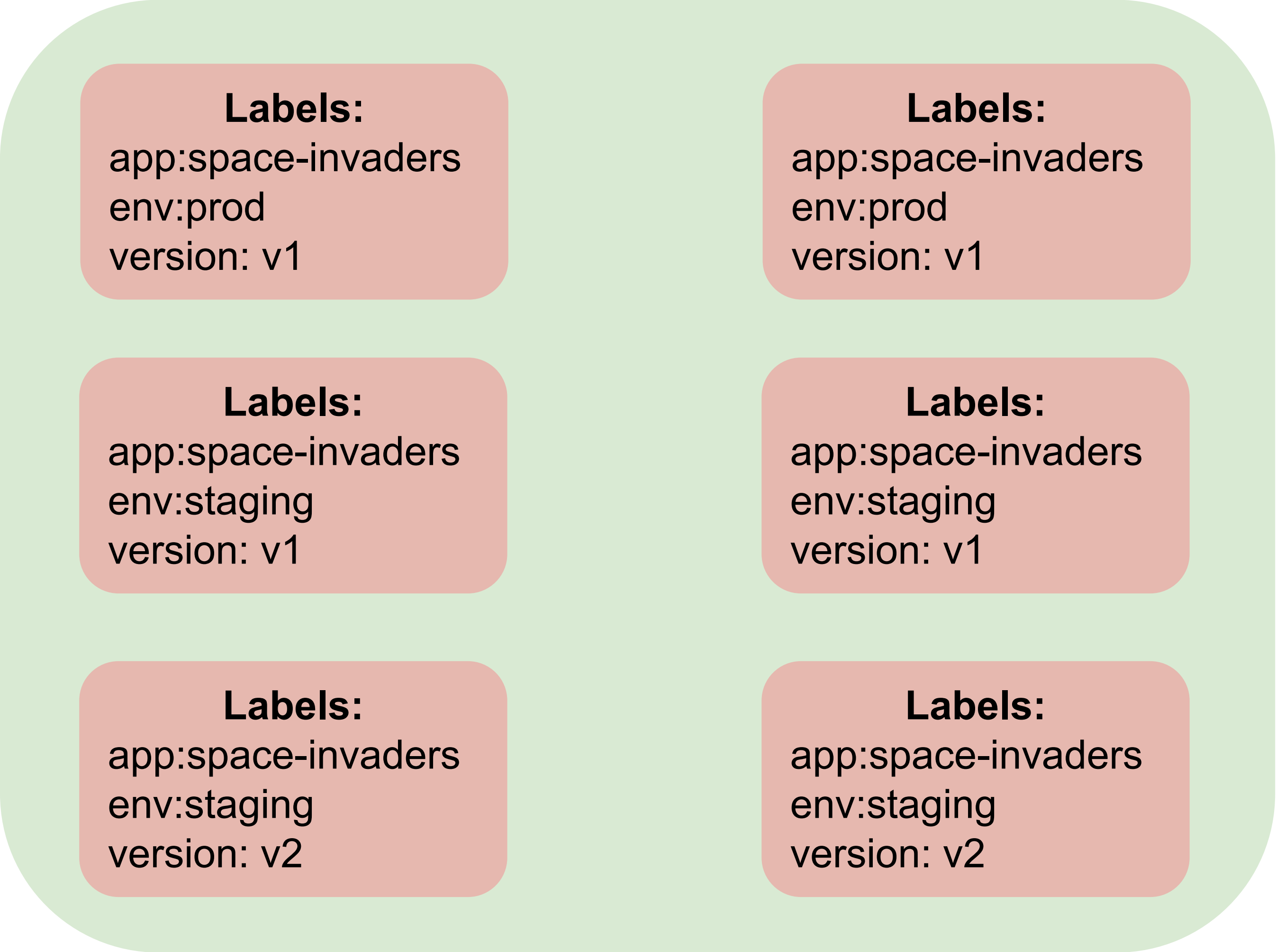
Label and Selector Examples

- Deployments commonly create many replicas of a pod
  - Each pod has a unique name and address
  - Pods can fail and be replaced causing name and address changes
- Many pods with different labels can be deployed to the same cluster
  - Deployments use labels to select which pods are part of the deployment
  - Services use labels to select which pods to expose

Label and Selector Examples (continued)

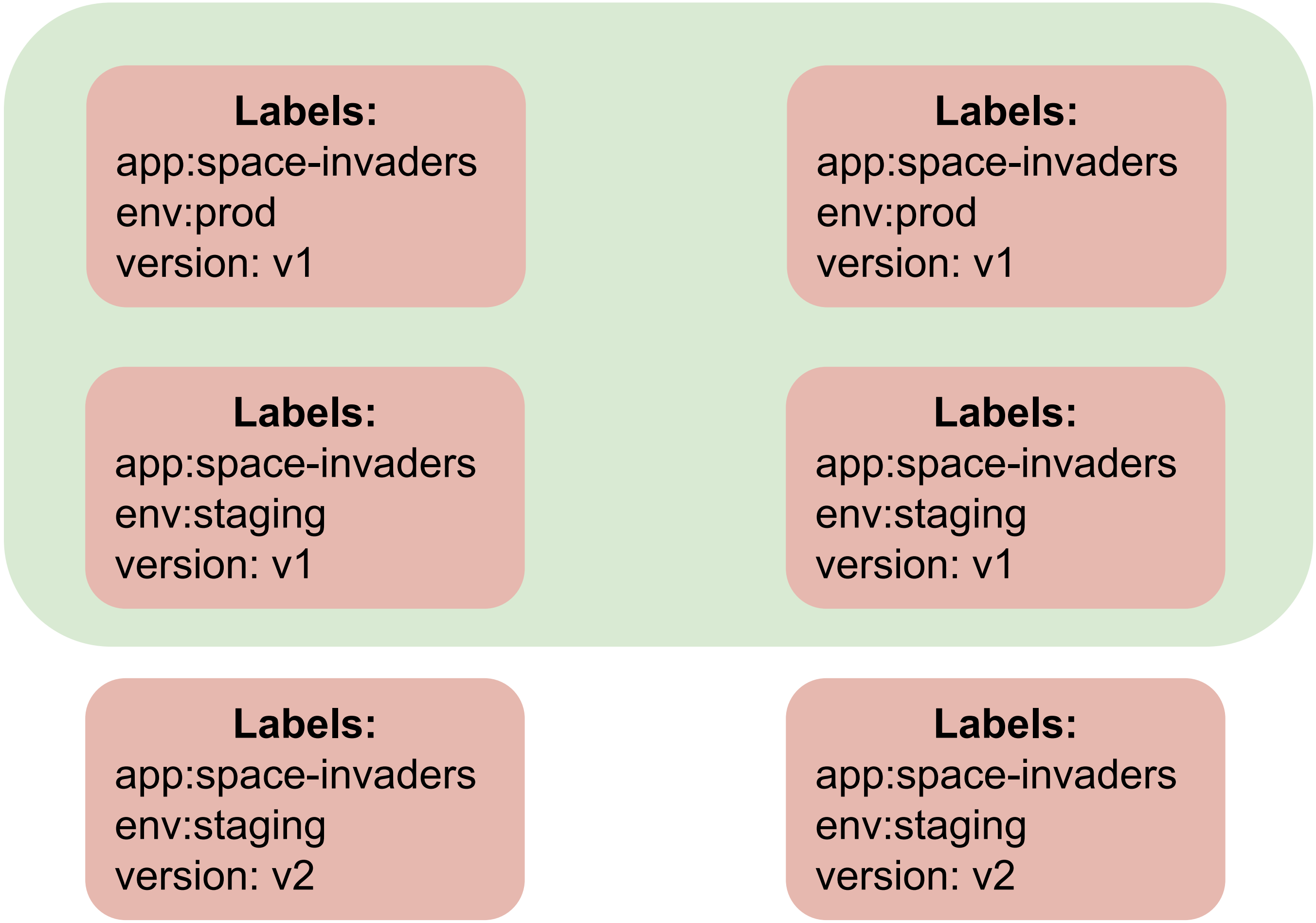
- `selector:`

  `app: space-invaders`



Label and Selector Examples (continued)

- ```
  selector:
    version: v1
  ```


  ```

Label and Selector Examples (continued)

- `selector:`

  `env: staging`

Labels:
app:space-invaders
env:prod
version: v1

Labels:
app:space-invaders
env:prod
version: v1

Labels:
app:space-invaders
env:staging
version: v1

Labels:
app:space-invaders
env:staging
version: v1

Labels:
app:space-invaders
env:staging
version: v2

Labels:
app:space-invaders
env:staging
version: v2

Label and Selector Examples (continued)

- ```
  selector:
    app: space-invaders
    env: staging
    version: v1
  ```

Labels:
app:space-invaders
env:prod
version: v1

Labels:
app:space-invaders
env:prod
version: v1

Labels:
app:space-invaders
env:staging
version: v1

Labels:
app:space-invaders
env:staging
version: v1

Labels:
app:space-invaders
env:staging
version: v2

Labels:
app:space-invaders
env:staging
version: v2
  ```

Creating Namespaces

- As seen earlier, namespaces can also be used to organize resources
  - Namespaces allow a single Kubernetes cluster to be divided into multiple “virtual clusters”
- Namespaces are very easy to create

```
$ kubectl create namespace development
```

Create a namespace called `development`

Using Namespaces

- When deploying resources, the default namespace is used unless a different namespace is specified
- It is recommended to not include the namespace within a YAML file
  - Best to keep the YAML neutral
  - Specify the namespace (with `--namespace` or `-n`) when applying the YAML

```
$ kubectl apply -f kubernetes-config.yaml --namespace=development
```

Use the namespace parameter to put resources in a particular namespace

Using Namespaces (continued)

- Must also specify namespace when viewing resources

```
$ kubectl apply -f kubernetes-config.yaml --namespace=development
deployment.apps/course-dev-app created
$
$ kubectl get pods
No resources found in default namespace.
$
$ kubectl get pods -n development
NAME                              READY   STATUS    RESTARTS   AGE
course-dev-app-86b5cb7795-fv2zx   1/1     Running   0          15s
course-dev-app-86b5cb7795-jbjms   1/1     Running   0          15s
course-dev-app-86b5cb7795-w8vg2   1/1     Running   0          15s
course-dev-app-86b5cb7795-xzzq5   1/1     Running   0          15s
```

Activity: Creating Kubernetes Deployments

In this activity, you will:

- Create a deployment for each microservice of the case study
- Manually scale the deployment replicas
- Experiment with pod lifecycle

Chapter Concepts

Deployments

Services

Resource Requests and Limits

Fault Tolerance and Scalability

Exposing Deployments

- Since pods can come and go quickly, how do you keep up with them?
  - Their address can change
  - Their location can change
- A service is how deployments are exposed so they can be accessed
  - Provides a consistent endpoint to access a group of pods

Selecting Pods to Expose

- Select which pods a service exposes by selecting pod labels
  - Service will expose all pods that match the labels

```
...
  selector:
    app: devops
    env: prod
    tier: frontend
...
```

Types of Services

ClusterIP	The default service type. Has only an internal IP address that is only accessible by other services running inside the cluster.
LoadBalancer	A service that provides an external IP address. In Google Cloud, this is implemented as a TCP load balancer. In AWS, this is implemented as an Elastic Load Balancer. Not all Kubernetes deployments would support this type of service. Can be expensive if you have lots of services, which means lots of load balancers.
NodePort	Assigns a port between 30000 and 32767 to nodes in your cluster. When a node is accessed at that port, it routes to your service.

ClusterIP Example

- A ClusterIP service has a static IP address
  - Not accessible outside the cluster
  - Other pods in the cluster can communicate with the ClusterIP IP address or service name

```
apiVersion: v1
kind: Service
metadata:
    name: demo-service
spec:
  type: ClusterIP
    selector:
        app: backend-pods
    ports:
        - protocol: TCP
            port: 8080
            targetPort: 8082
```

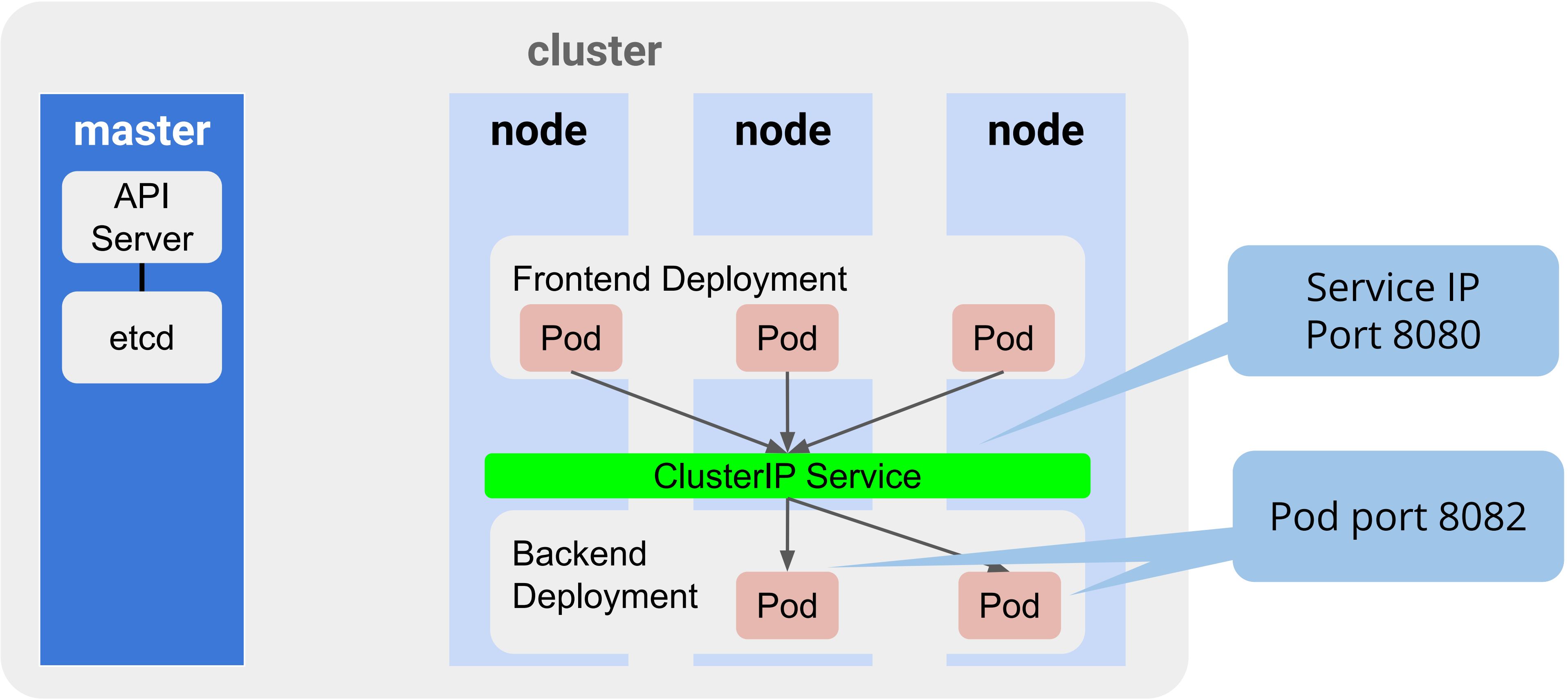
ClusterIP Example (continued)

- Example creating and viewing ClusterIP service

```
$ kubectl apply -f service.yaml
service/demo-service created
$ kubectl get service
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes      ClusterIP   10.8.0.1     <none>        443/TCP          59m
demo-service    ClusterIP   10.8.2.26    <none>        8080:31133/TCP   56s
$
```

ClusterIP Example

- ClusterIP from previous slide



Creating a NodePort Service

- NodePort is built on top of ClusterIP Service and enables external communication
  - A ClusterIP Service is automatically created to distribute the inbound traffic internally across a set of pods
- A NodePort service can be reached from outside the cluster using the IP address of any node and the corresponding NodePort port number
  - This traffic is then directed to a ClusterIP service
  - And then routed to one of the pods
- NodePort Service can be useful to expose a service through an external load balancer that you set up and manage yourself

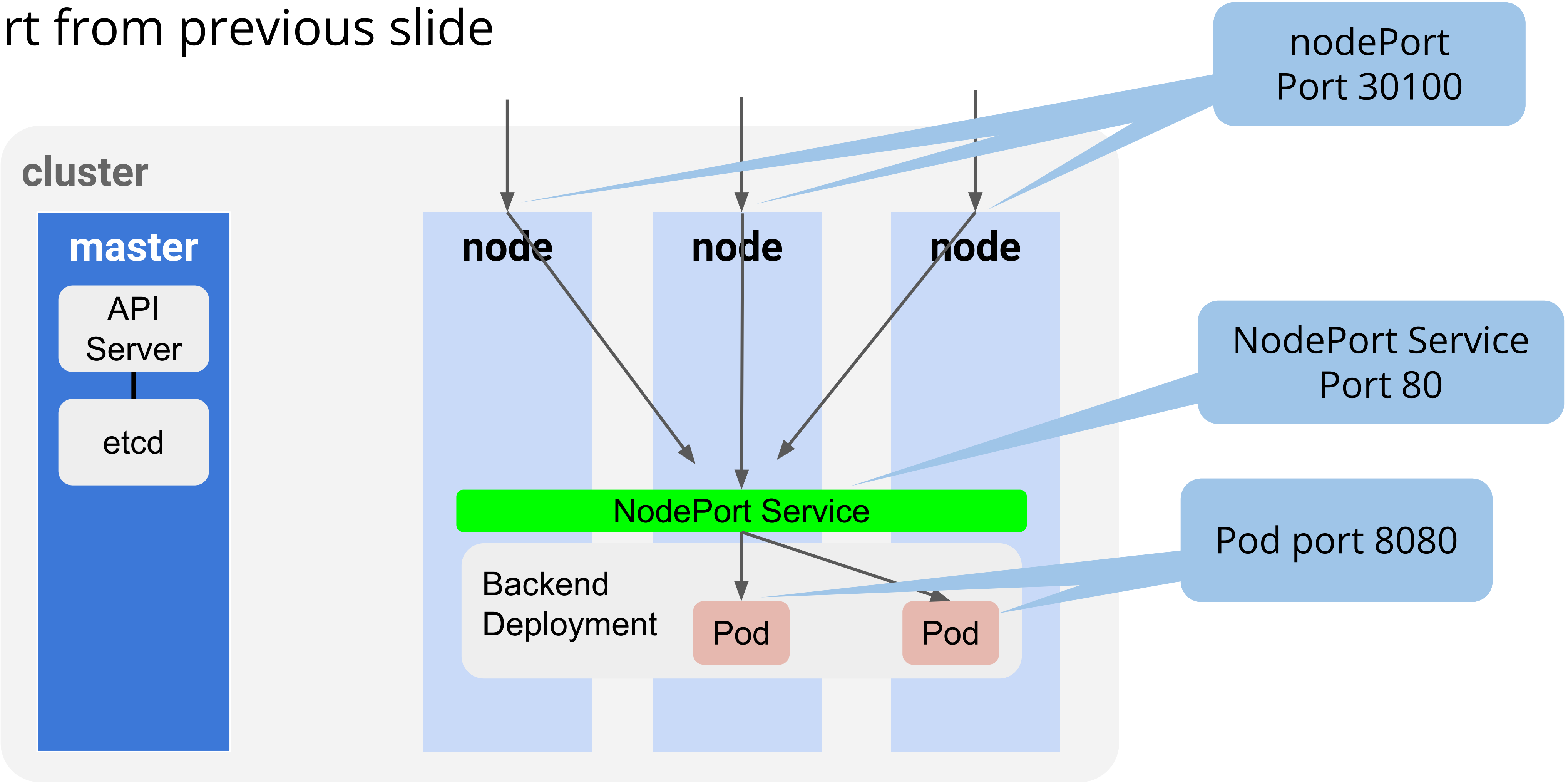
Creating a NodePort Service (continued)

- The nodePort is automatically allocated from the range 30,000 to 32767
    - Can also manually specify it, but must be within the same range

```
apiVersion: v1
kind: Service
metadata:
    name: my-service
spec:
  type: NodePort
    selector:
        app: external
    ports:
        - protocol: TCP
            nodePort: 30100
            port: 80
            targetPort: 8080
```

NodePort Example

- NodePort from previous slide



Load Balancer Service

- LoadBalancer Service exposes pods outside the cluster
  - Builds on a ClusterIP and exposes the service as an external IP address

```
apiVersion: v1
kind: Service
metadata:
    name: demo-service
spec:
  type: LoadBalancer
    selector:
        app: frontend-pods
    ports:
        - protocol: TCP
            port: 80
            targetPort: 8081
```

Another Load Balancer Example

```
apiVersion: v1
kind: Service
metadata:
  name: devops-loadbalancer
  labels:
    app: devops
    tier: frontend
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: devops
    tier: frontend
```

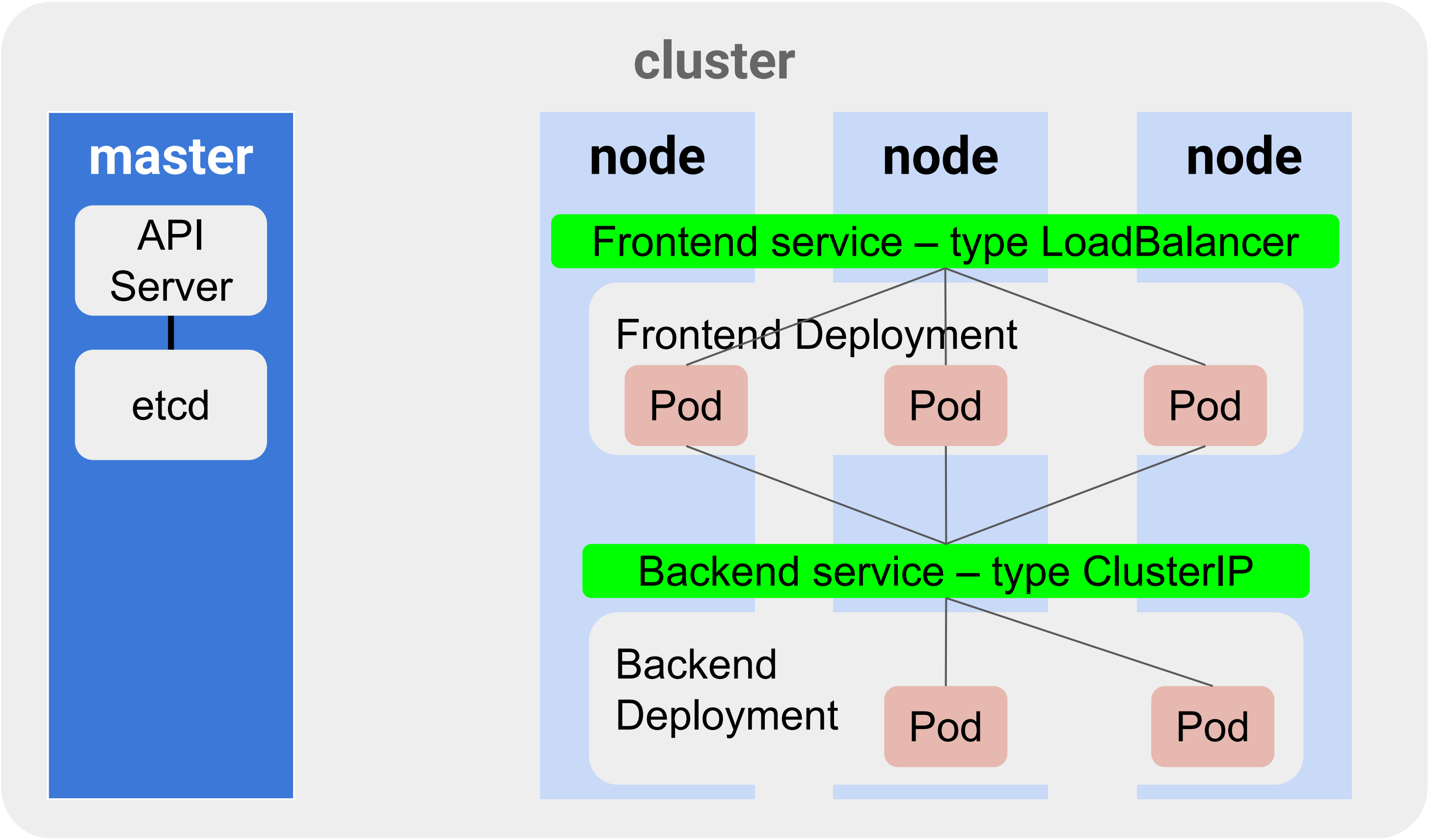
Creating a Load Balancer via Configuration (continued)

- Example creating and viewing LoadBalancer service

```
$ kubectl apply -f service.yaml
service/devops-loadbalancer created
$ kubectl get service
NAME                   TYPE           CLUSTER-IP    EXTERNAL-IP    PORT(S)         AGE
kubernetes             ClusterIP      10.8.0.1      <none>         443/TCP         59m
devops-loadbalancer    LoadBalancer   10.8.2.26     34.72.29.76    80:31133/TCP    56s
$
```

Multiple Services

- A single application can have multiple services



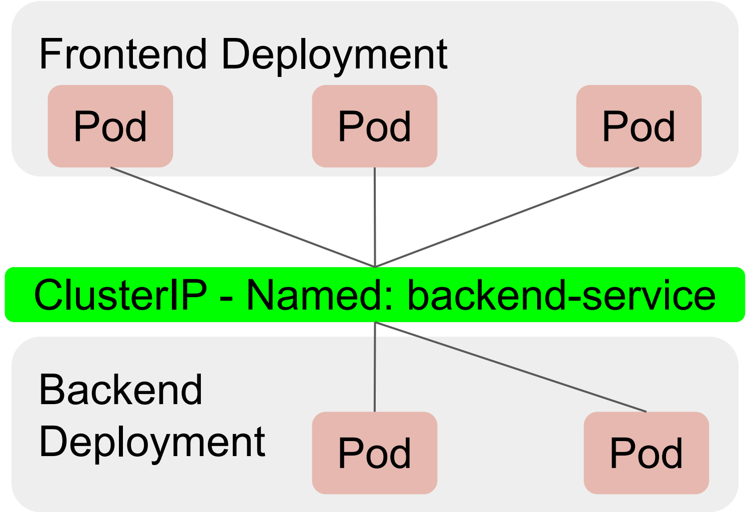
Accessing Services from Within the Cluster

- Services have their own IPs
  - Can connect to the cluster IP from within the cluster
  - But you do not know the IP address until the service is deployed
- Kubernetes creates DNS records for services and pods
  - You can contact services with consistent DNS names instead of IP addresses
  - The DNS name is just the service name
    - You know this ahead of time
  - If connecting across namespaces, the DNS name is <service-name>.<namespace-name>

Environment Variables

- Assume the configuration on the right
    - The frontend pods need to know the name of the ClusterIP service
    - Can be passed to the containers as an environment variable in the deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
... <omitted lines>
    spec:
      containers:
      - image: frontend:v1
        env:
        - name: BACKEND
          value: "backend-service"
... <omitted lines>
```



Headless Services

- Sometimes, you may not want or need the load balancing or the single IP that a load balancer and ClusterIP service provide
    - Can create what is called a Headless service
    - Specify “None” for the cluster IP
- A cluster IP is not allocated and kube-proxy does not handle these Services
    - It is used for discovering individual pods and interacting directly with the pods instead of a proxy
    - Headless Services can be used to interface with other service discovery mechanisms
    - Without being tied to a specific Kubernetes implementation

Headless Service Example

```yaml
apiVersion: v1
kind: Service
metadata:
  name: headless-demo-service
  labels:
    app: demo
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: demo
```

```
$ kubectl apply -f service.yaml
service/headless-demo-service created
$ kubectl get service
NAME                   TYPE         CLUSTER-IP   EXTERNAL-IP   PORT(S)      AGE
kubernetes             ClusterIP    10.8.0.1     <none>        443/TCP      59m
Headless-demo-service  ClusterIP    None         <none>        80/TCP       6s
$
```

Headless Service Example (continued)

Execute inside a testpod that was deployed to the cluster

Perform a nslookup of a ClusterIP service

Returns a single IP

Perform a nslookup of a Headless service

Returns the IPs of all pods behind the service

```
$ kubectl exec testpod -it -- /bin/bash
root@testpod:/# nslookup clusterip-demo-service
Server:         10.8.0.10
Address:        10.8.0.10#53

Name:   clusterip-demo-service.default.svc.cluster.local
Address: 10.8.13.64
root@testpod:/#
root@testpod:/# nslookup headless-demo-service
Server:         10.8.0.10
Address:        10.8.0.10#53

Name:   headless-demo-service.default.svc.cluster.local
Address: 10.4.0.2
Name:   headless-demo-service.default.svc.cluster.local
Address: 10.4.1.5
Name:   headless-demo-service.default.svc.cluster.local
Address: 10.4.2.6
root@testpod:/#
```

Deleting Deployments and Resources

- Use the `delete` command to destroy anything previously created
    - Specifying a configuration file will delete everything created from it

```
kubectl delete -f kubernetes-config.yaml
```

- Can also delete resources individually

```
kubectl delete pod devops-pod
kubectl delete deployment devops-deployment
kubectl delete services devops-loadbalancer
```

Activity: Creating a ClusterIP and Load Balancer

In this activity, you will:

- Create services for the two case study microservices
  - A ClusterIP service for the backend microservice
  - A load balancer for the frontend microservice

Chapter Concepts

Deployments

Services

Resource Requests and Limits

Fault Tolerance and Scalability

Managing Resources

- Kubernetes can limit the CPU and RAM resources a container can use
  - CPU and RAM (memory) are the most common but there are others
- CPU resources are measured in fractional cpu units (millicpus)
  - As a decimal value between 0 and 1
  - Or a value between 0m and 1000m
  - 0.5 or 500m are both 500 millicpu
  - 1 or 1000m is equivalent to **1 vCPU/Core** for cloud providers or 1 hyperthread on bare metal Intel processors

Managing Resources (continued)

- Memory resources are measured in bytes
  - As an integer or fixed-point number
  - Using the following suffixes: E, P, T, G, M, k
  - Can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki
  - The following example are all the same RAM value:
    - 128974848, 129e6, 129M, 123Mi

Note: MiB vs. MB

1 MB (MegaByte) = $(10^3)^2$ bytes = 1000000 bytes
1 MiB (MebiByte) = $(2^{10})^2$ bytes = 1048576 bytes

Can Control the Resources Your Pods Require and Are Allowed to Consume

```
apiVersion: apps/v1
kind: Deployment

***CODE OMITTED FOR SPACE ***

    spec:
      containers:
      - name: devops-demo
        image: drehnstrom/devops-demo:latest
          ports:
        - containerPort: 8080
        resources:
            requests:
              memory: "256Mi"
              cpu: "0.1"
            limits:
              memory: "512Mi"
              cpu: "0.5"
```

requests:
The minimum amount of resources required for each pod

limits:
The maximum amount of resources a pod is allowed to consume

Chapter Concepts

Deployments

Services

Resource Requests and Limits

Fault Tolerance and Scalability

Health Checks

- There are two types of Kubernetes health checks: Liveness and Readiness probes
  - If a Liveness probe fails, the container is restarted
  - If a Readiness probe fails, requests are not sent to the pod until it is back up and running
    - But it is not removed or restarted

Defining Liveness and Readiness Probes

- Liveness and readiness probes can be defined as:
  - A command
  - A TCP probe
  - A HTTP request
- Can specify the following properties:
  - **`initialDelaySeconds`**: time to wait before performing the first probe
  - **`periodSeconds`**: how often to perform the probe
  - **`timeoutSeconds`**: how long the probe has to respond

Liveness and Readiness Probe Configuration

```
apiVersion: apps/v1
kind: Deployment
*** CODE OMITTED ***
    spec:
      containers:
      *** CODE OMITTED ***
         readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 30
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 15
          periodSeconds: 15
```

Added to the Containers section of the Deployment

Liveness and Readiness Probe Configuration (continued)

- The exec command option invokes a command within the container

```
apiVersion: apps/v1
kind: Deployment
*** CODE OMITTED ***
    spec:
      containers:
      *** CODE OMITTED ***
         livenessProbe:
           exec:
             command:
               - /bin/status
           initialDelaySeconds: 15
           periodSeconds: 10
           timeoutSeconds: 5
```

Autoscaling

- Kubernetes can manage the scaling of pods in a deployment
- Horizontal scaling creates more copies (replicas) of a pod
- Vertical scaling increases the resources (CPU, RAM) available to a pod

Horizontal Pod Autoscaler (HPA)

- Kubernetes has a Horizontal Pod Autoscaler
  - Can automatically scale the number of pods
  - Based on CPU or other metrics defined with the Kubernetes Metrics Server
  - Can specify the min and max replica count
- By default it queries the resource utilization every 15 seconds
  - This time can be set with the `--horizontal-pod-autoscaler-sync-period` flag

Creating an HPA via Configuration

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: devops-autoscaler
spec:
  scaleTargetRef:
    apiVersion: apps/v1beta1
    kind: Deployment
    name: devops-deployment
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 60
```

HPA Cooldown

- Autoscaling can sometimes cause thrashing
  - The number of replicas rapidly fluctuates up and down
- Thrashing can be minimized with a downscale stabilization time window
  - Prevent downscale operations from happening too quickly
  - Specified with the `--horizontal-pod-autoscaler-downscale-stabilization` flag
  - Default is 5 minutes

Vertical Pod Autoscaler (VPA)

- The Vertical Pod Autoscaler can recommend values for CPU and memory requests and limits, or it can automatically update the values
- Limitations:
  - Running pods must be recreated to modify the resource requests
    - It evicts a pod if it needs to change the pod's resource requests
  - Not meant for sudden increases in resource usage
    - Provides stable recommendations over a longer time period
    - Use HPA for sudden increases
  - Does not yet work with JVM-based workloads
  - The Vertical Pod Autoscaler recommendations might exceed available resources

Vertical Pod Autoscaler (VPA) (continued)

- The Vertical Pod Autoscaler can run in auto-update mode or recommendation mode
  - Auto-update mode will recreate the pod automatically
  - Auto-update mode off will create recommendations
    - Analyzes the CPU and memory needs of the containers and records those recommendations in its status field
    - View the recommendations made by the Vertical Pod Autoscaler:
      `kubectl describe vpa`

Activity: Adding Resource Requests and Limits, and Autoscaling

In this activity, you will:

- Add resource limits to limit the resources each container can use
- Autoscale the pods with an HPA

Chapter Summary

In this chapter, you have:

- Created deployments
- Leveraged labels to identify different types of pods in a cluster
- Exposed deployments as a consistent endpoint with services
- Controlled pod resource limits
- Monitored pods with aliveness and readiness probes
- Performed pod autoscaling

Version Management

Chapter Objectives

In this chapter, you will:

- Deploy new versions of containers with zero or little downtime
- Perform rolling updates and roll back deployments
- Perform blue/green deployments and canary releases

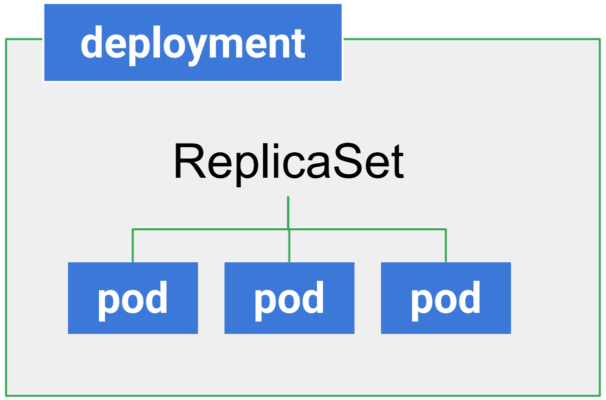
Chapter Concepts

Deploying New Container Versions

Testing New Versions

Updating Pods in a Deployment

- Deployments rely on ReplicaSets to manage and run pods
  - Eventually, you will need to upgrade pods in the deployment
  - There are several ways to manage this update

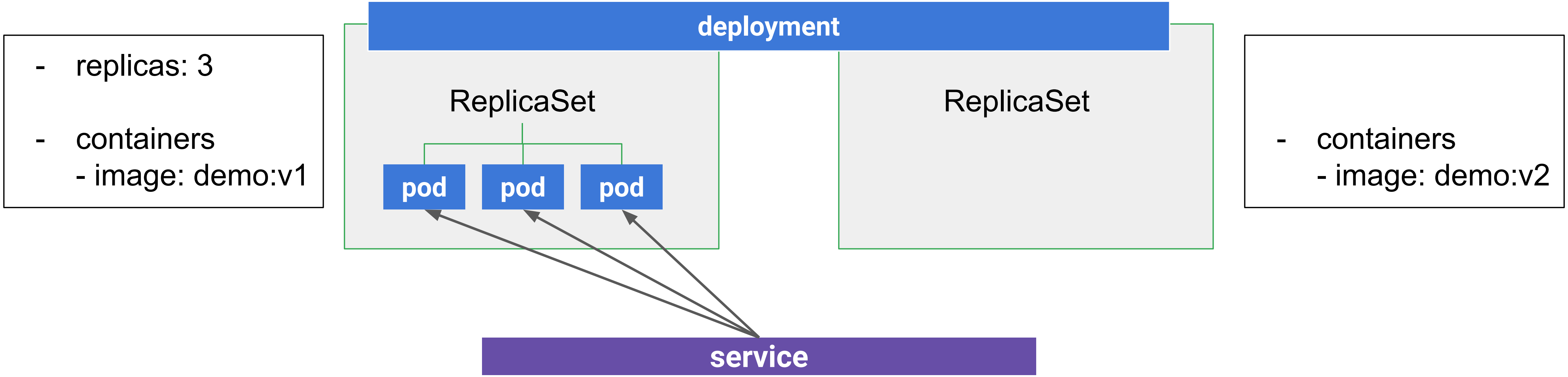


Rolling Updates

- Rolling updates allow you to gradually update from one image version to another
  - A rolling update is triggered by default when the pod template is changed
- The pod template can be changed several ways, including:
  - By changing the image name or version in the yaml and reapplying
  - By calling `kubectl set image ...`

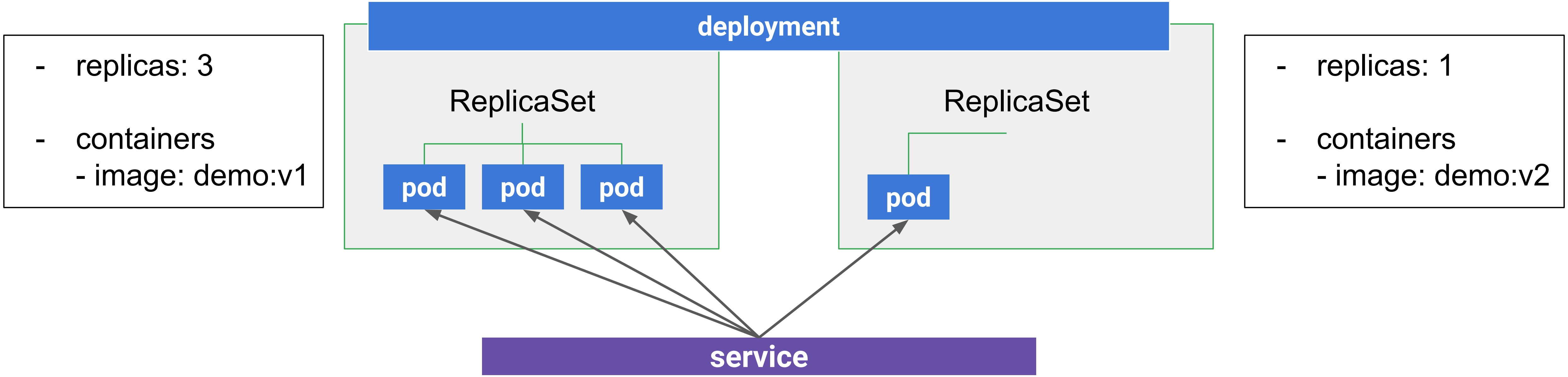
Rolling Updates (continued)

- A rolling update causes the deployment to create a second ReplicaSet



Rolling Updates (continued)

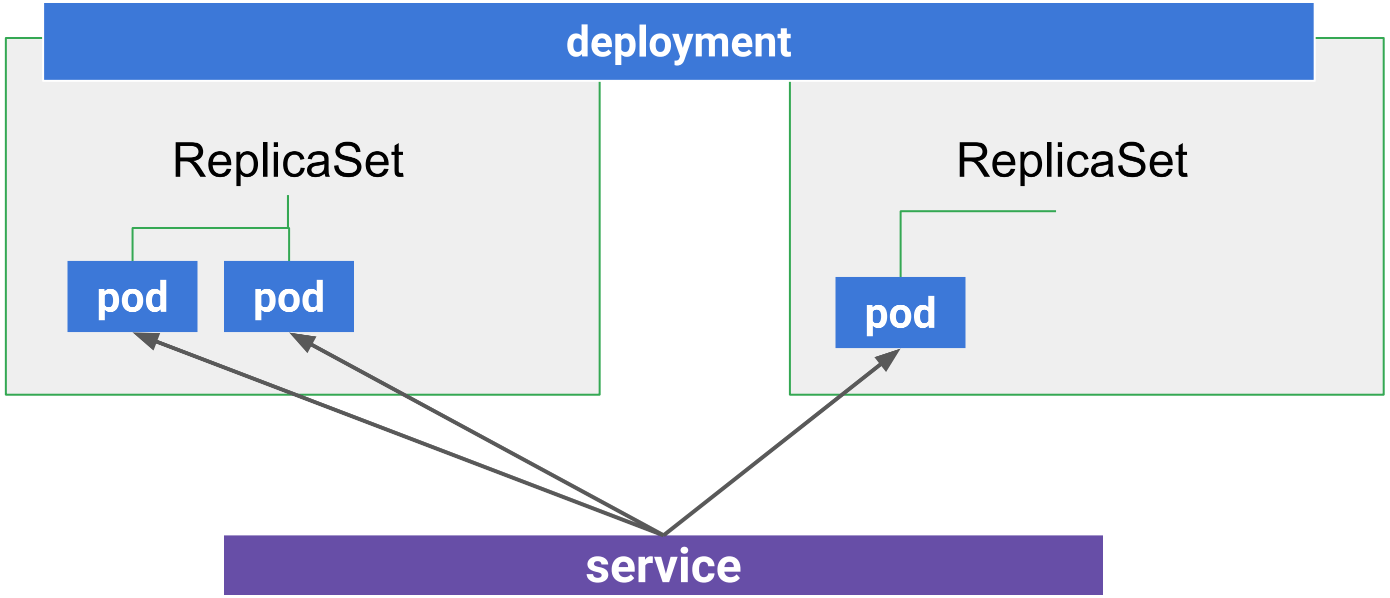
- New pods are gradually added to the new ReplicaSet



Rolling Updates (continued)

- Old pods are gradually removed from the old ReplicaSet

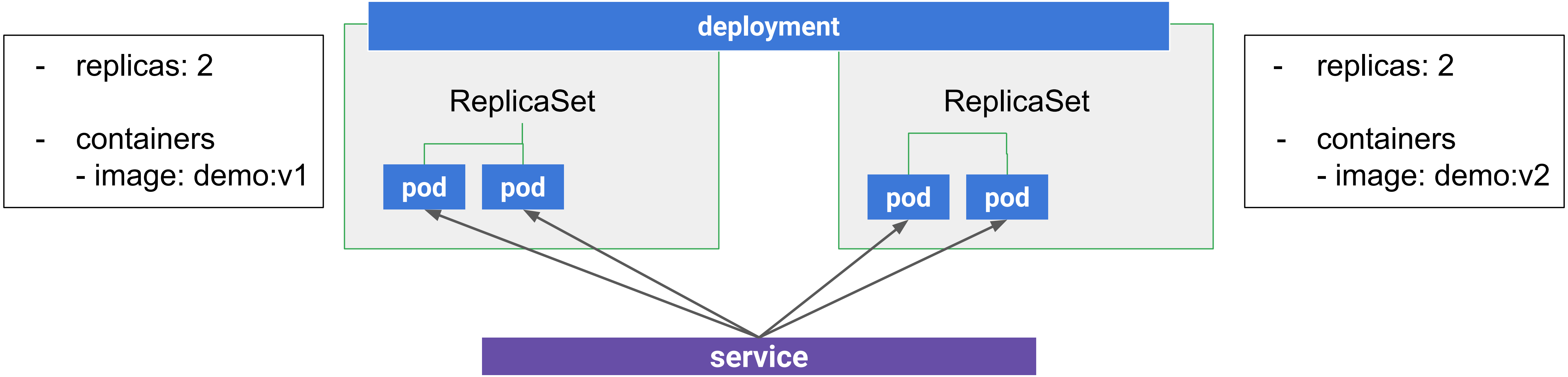
- replicas: 2
- containers
  - image: demo:v1



- replicas: 1
- containers
  - image: demo:v2

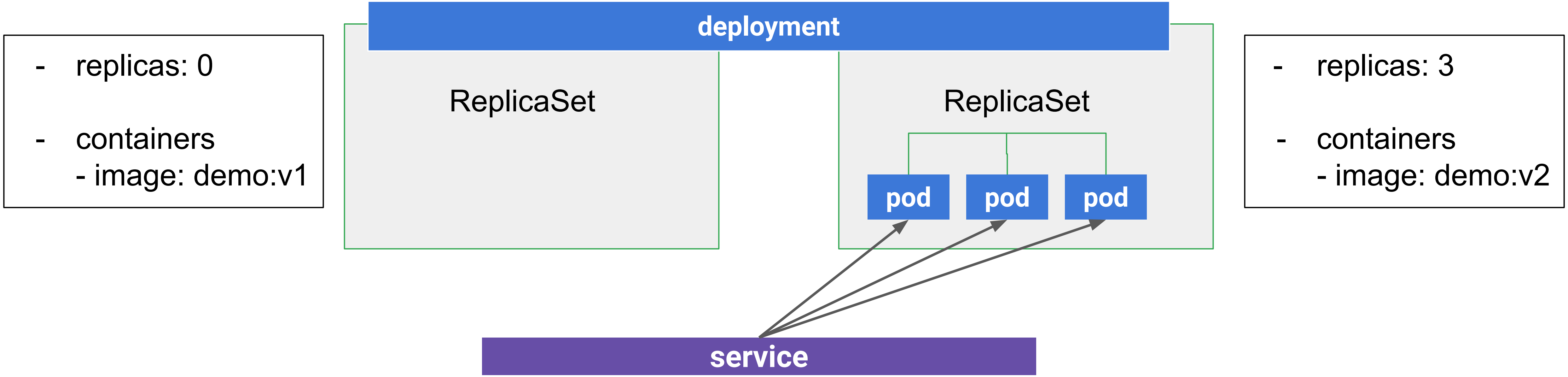
Rolling Updates (continued)

- During a rolling update, the number of running pods will fluctuate
  - This amount can be controlled



Rolling Updates (continued)

- This continues until the new version is rolled out



Controlling a Rolling Update

- The number of pods created and deleted at a time can be controlled
- `maxSurge`
  - Maximum number of new pods that will be created at a time
  - Can be a number or a percentage
  - Defaults to 25%
- `maxUnavailable`
  - Maximum number of old pods that will be deleted at a time
  - Can be a number or a percentage
  - Defaults to 25%

Update Strategy

- There are actually two update strategies
  - **RollingUpdate**
    - What we just discussed
    - The default
  - **Recreate**
    - All old pods are terminated before any new pods are added
- Can be set on the deployments

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: update-demo
spec:
 replicas: 3
 strategy:
 type: RollingUpdate
 rollingUpdate:
   maxSurge: 1
   maxUnavailable: 25%
 selector:
...
```

Controlling the Update

- The status and history of an update can be retrieved
- Rolling updates can also be paused, resumed, or rolled back

```
# Watch update status
kubectl rollout status deployment/update-demo

# Pause deployment
kubectl rollout pause deployment/update-demo

# Resume deployment
kubectl rollout resume deployment/update-demo

# View rollout history
kubectl rollout history deployment/update-demo
```

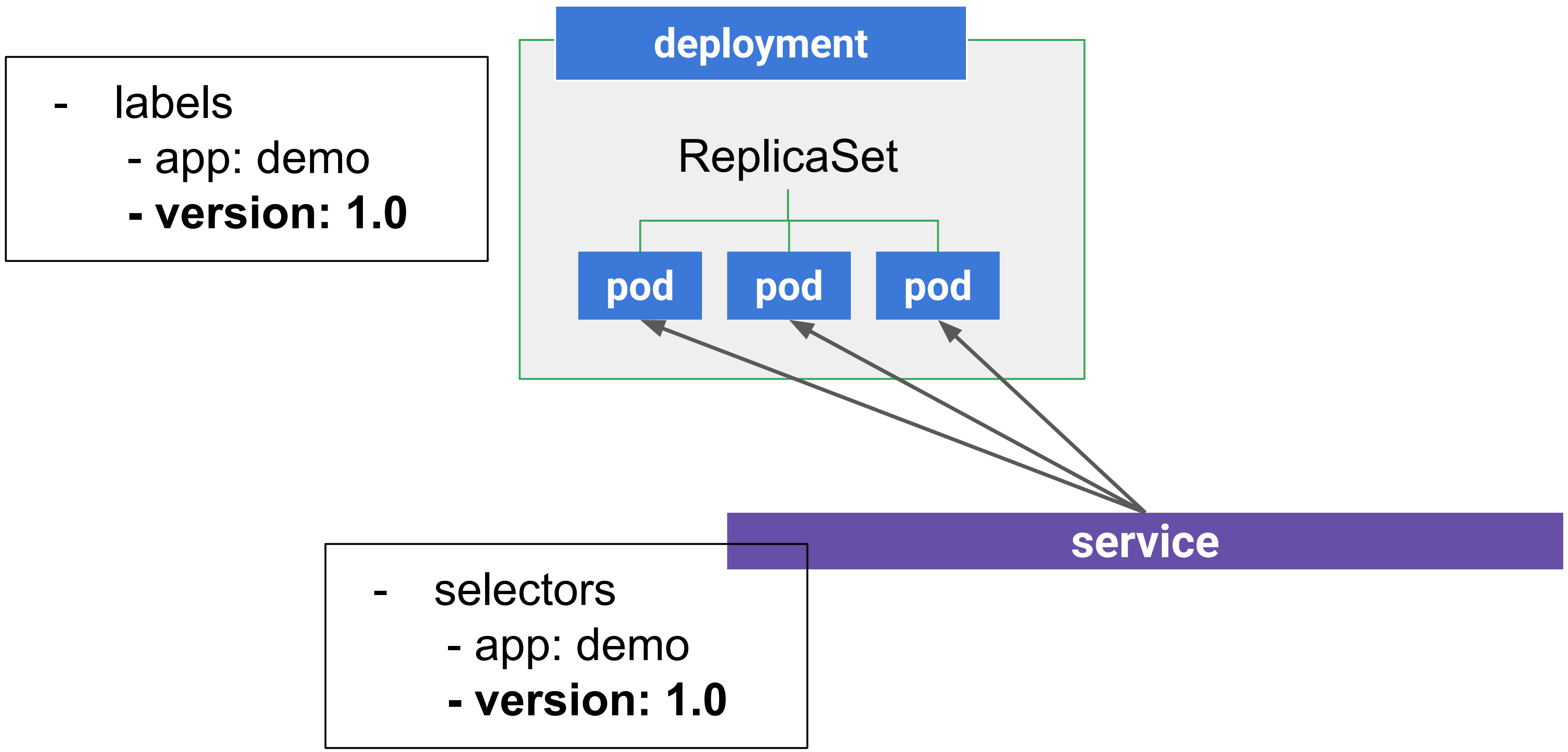
Chapter Concepts

Deploying New Container Versions

Testing New Versions

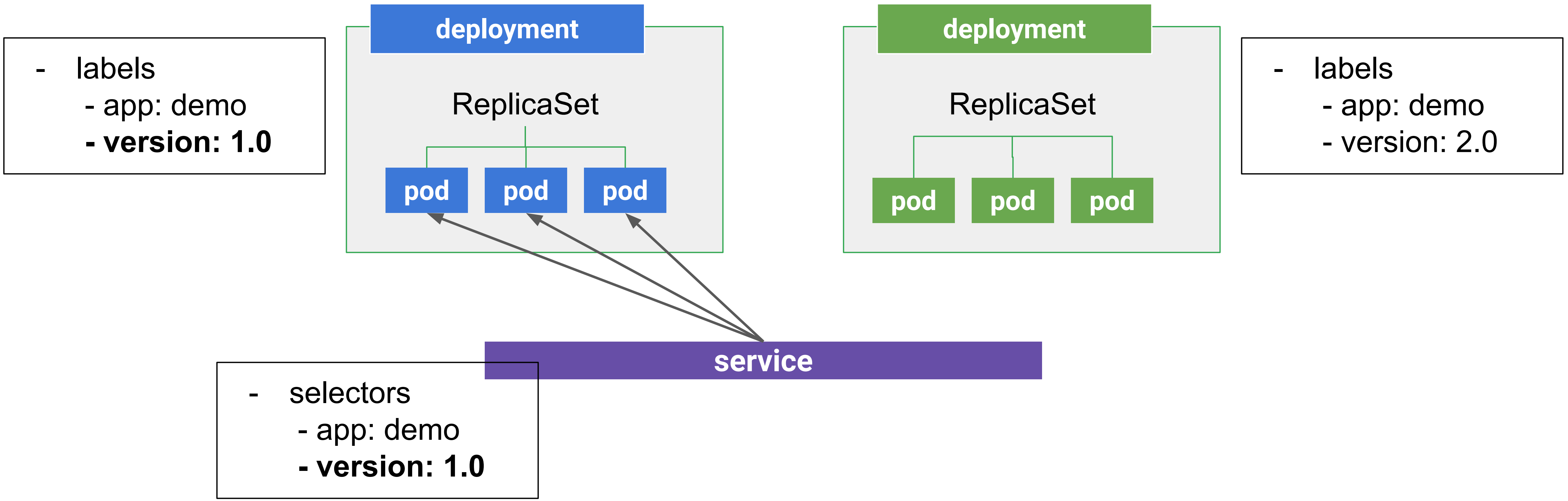
Blue/Green Deployments

- A Blue/Green deployment switches all traffic from one deployment to another
  - And can easily switch between



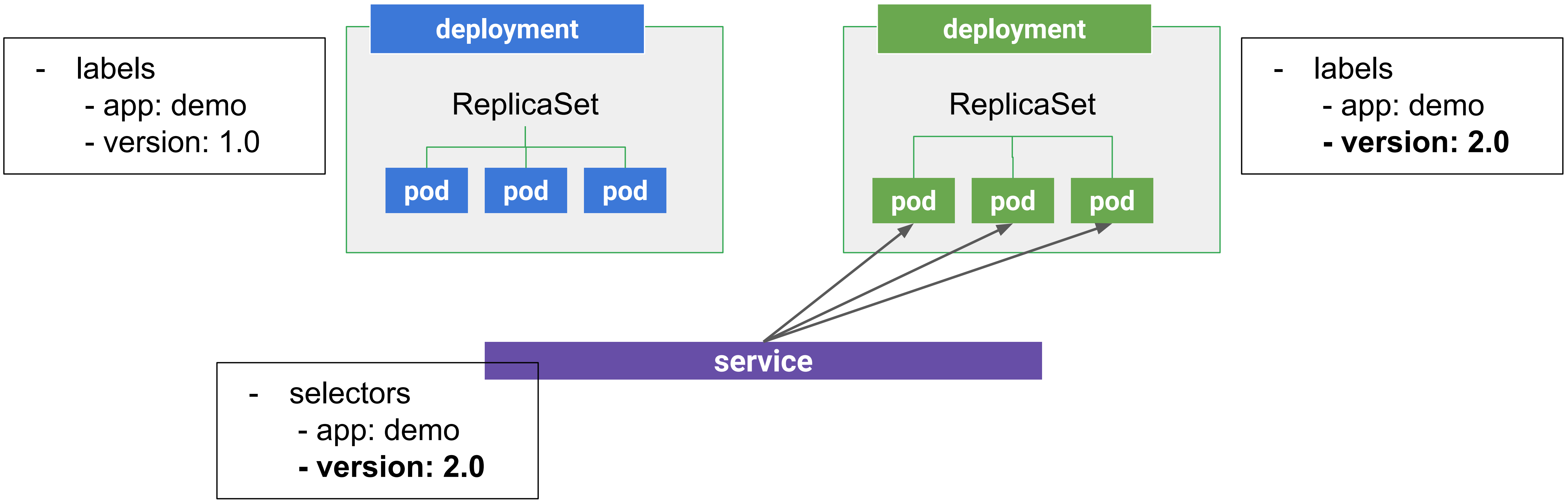
Blue/Green Deployments (continued)

- Create a new deployment for version 2 (green)



Blue/Green Deployments (continued)

- Update the service selector to the new version

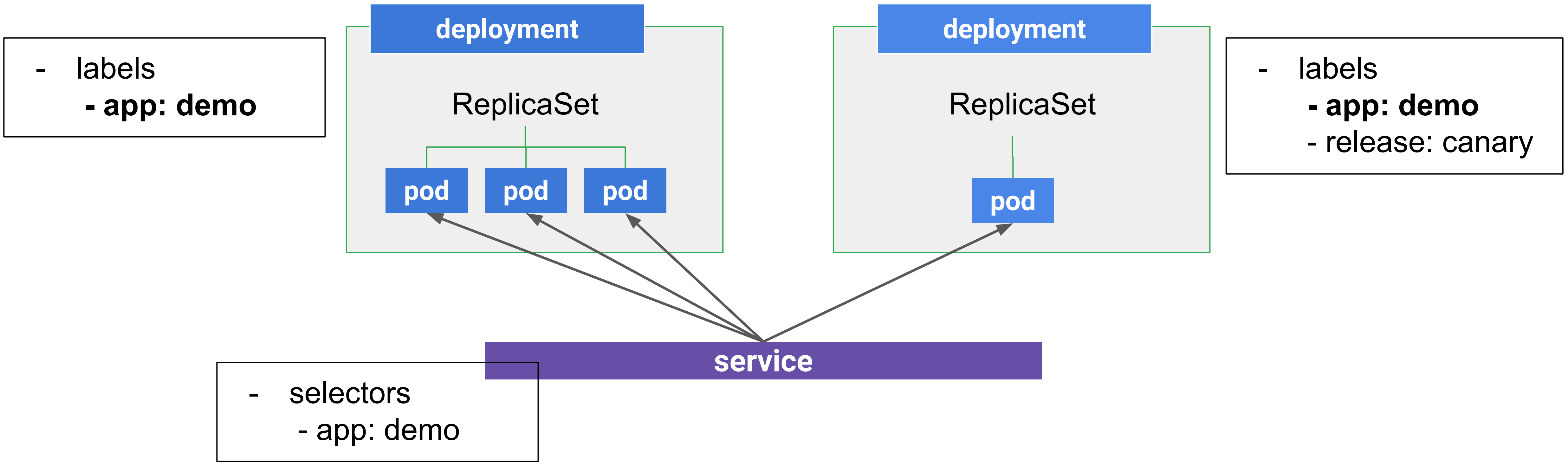


Canary Release

- A canary release is a software update technique where a new version of a service is put into production alongside old versions
  - A small subset of selected traffic is routed to the new (canary) release
- Can try out a new version of your application against a small subset of live requests
  - When satisfied, you can roll it out to the new deployment
  - Or pull it back

Canary Deployment

- Can be implemented by a service backed with multiple deployments
  - Service selects a label common to both deployments



Activity: Performing Rolling Updates and Blue/Green Deployments

In this activity, you will:

- Perform a rolling update of your pods
- Implement a Blue/Green deployment
- Create a simple canary release

Chapter Summary

In this chapter, you have:

- Deployed new versions of containers with zero or little downtime
- Performed rolling updates and roll back deployments
- Performed blue/green deployments and canary releases

Persistent Storage

Chapter Objectives

In this chapter, you will:

- Allocate disk space for pods
- Implement StatefulSet to run and maintain sets of pods
- Use configmaps and secrets to decouple configuration from pods

Chapter Concepts

Allocating Storage for Pods

StatefulSets

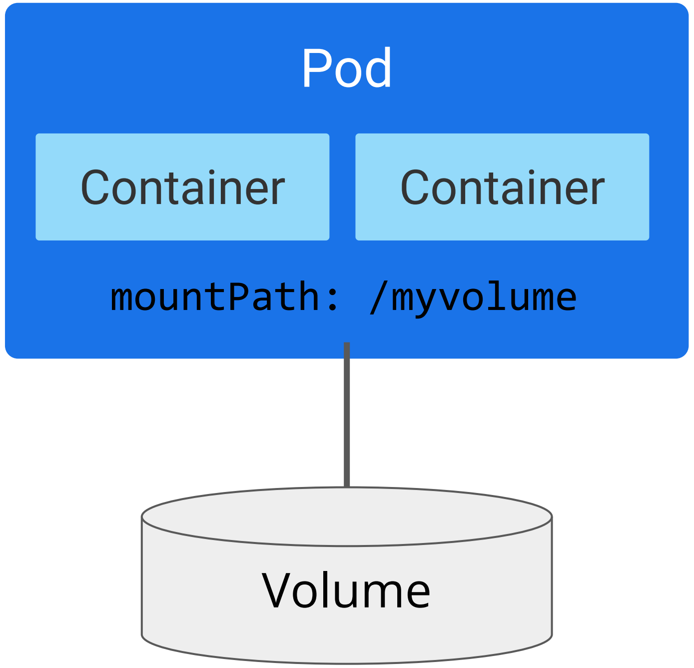
Configuration Data

Reading and Writing to Storage Volumes

- Microservices should store most state in some type of data store
  - That is external to the service
  - A relational database, a NoSQL database, etc.
- However, applications sometimes need persistent storage to:
  - Cache data
  - Save temp files or a scratch space
  - Share data between containers in the same pod
  - Etc.

Kubernetes Volumes

- A volume is storage that is accessible to containers in a pod
  - Some volumes are ephemeral
    - Follow the life of the pod
  - Some volumes are persistent
    - Continue to exist even if the pod is deleted
- Volumes are attached to pods, not containers
  - Containers in the pod can share data



`emptyDir` Volume Type

- An `emptyDir` is an example of an ephemeral volume
  - Follows the pod's lifecycle
- Initially does not contain any information
- All containers in the pod can read and write files in the volume
- When a pod is removed from a node, the volume is deleted

```
...
kind: Deployment
...
  spec:
   containers:
    - name: web
      image: nginx
      volumeMounts:
      - mountPath: /tempvol
        name: empty-test
    volumes:
    - name: empty-test
      emptyDir: {}
```

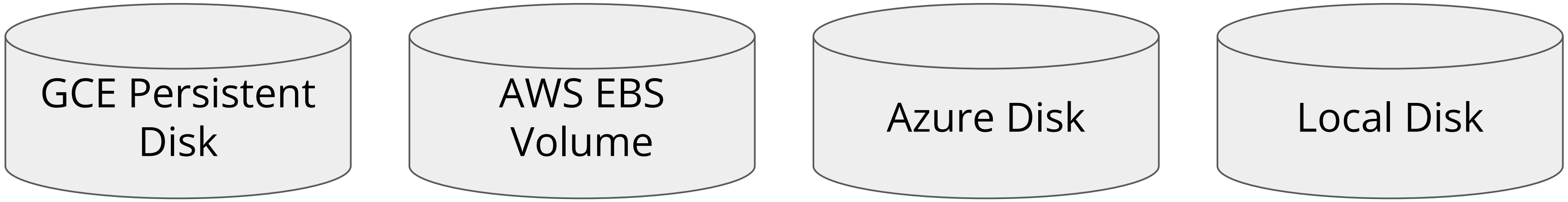
Using NFS Volumes

- NFS volumes can be attached to pods
- The NFS server could be anywhere
  - The NFS server must already exist
    - Must be provisioned outside of Kubernetes
  - The pods must have network access to the NFS server
  - Data on the NFS volume will outlive the pod
- NFS volumes can be accessed from multiple pods at the same time
  - An NFS can be used to share data between pods!

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: nginx
spec:
    containers:
     - name: web
       image: nginx
       volumeMounts:
       - mountPath: /mnt/vol
         name: nfs
    volumes:
    - name: nfs
      server:10.10.10.2
      path: "/"
      readOnly: false
```

Kubernetes PersistentVolumes

- What if a pod needs its own persistent volume?
- Kubernetes PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned
  - And can be attached to a pod
  - Lifecycle independent of any individual pod
- The volumes themselves are cluster implementation specific
  - The volume must be provisioned for the cluster environment
    - A local cluster, Google Cloud, or another cloud provider



Kubernetes PersistentVolumeClaim

- A PersistentVolumeClaim (PVC) is a request for storage by a user
    - If a matching volume exists, it will be claimed
    - Provisioning can also be dynamic
- PersistentVolumeClaims and PersistentVolumes separate storage consumption from provisioning
    - Allows for provisioned storage capacity to be claimed by the application
- The cluster administrator configures storage classes for dynamic provisioning to occur
    - Most cloud-based Kubernetes clusters already do this
    - For example, Google Kubernetes Engine will dynamically provision a Compute Engine persistent disk for each PVC

PVC Example

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pvc-demo
  labels:
    name: new-nginx
spec:
  volumes:
    - name: pv-storage
      persistentVolumeClaim:
        claimName: my-web-disk
  containers:
  - name: nginx-pod
    image: nginx:latest
    ports:
    - containerPort: 80
    volumeMounts:
    - name: pv-storage
      mountPath: "/var/www/html"
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-web-disk
spec:
  storageClassName: "standard"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5000Mi
```

PVC Example (continued)

```
$ kubectl apply -f pvc-demo.yaml
pod/pvc-demo created
persistentvolumeclaim/my-web-disk created
$
$ kubectl get pvc
NAME            STATUS   VOLUME                              CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-web-disk     Bound    pvc-afcc-4dd7-4491-a3fd-5a5e874     5Gi        RWO            standard       45s
$
$ kubectl get pod
NAME                             READY   STATUS    RESTARTS   AGE
pvc-demo                         1/1     Running   0          71s
$
$ kubectl exec -it pvc-demo -- /bin/bash
root@pvc-demo:/# cd /var/www/html/
root@pvc-demo:/var/www/html# echo "This is a test file" > test.html
root@pvc-demo:/var/www/html# ls
lost+found  test.html
```

PVC Example (continued)

- If a matching PV already exists, Kubernetes will bind it to the claim
- If the PVC's `storageClassName` is defined and an appropriate PV does not already exist, Kubernetes will try to dynamically provision a PV
  - If `storageClassName` is omitted, the PVC will use the default StorageClass for that cluster
    - Google Kubernetes Engine clusters default to "standard" which is a standard (magnetic) persistent disk

Deleting PVC

- Deleting a PVC will also delete the underlying provisioned PV
  - Generally, good practice to clean up volumes when no longer needed
- To retain the PV when the PVC is deleted:
  - Set its `persistentVolumeReclaimPolicy` to `Retain`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-web-disk
spec:
  storageClassName: "standard"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5000Mi
  persistentVolumeReclaimPolicy: Retain
```

PV AccessModes

- The AccessModes determine how the PV can be read from or written to
  - ReadWriteOnce
  - ReadOnlyMany
  - ReadWriteMany

```
...
  accessModes:
    - ReadWriteOnce
```

```
...
  accessModes:
    - ReadOnlyMany
```

```
...
  accessModes:
    - ReadWriteMany
```

- Some underlying environments may not support all access modes
  - For most applications, persistent disks are mounted as ReadWriteOnce
  - Most persistent disk implementations will not support ReadWriteMany
    - For Example, GCE persistent disk
    - NFS volumes do support ReadWriteMany

Chapter Concepts

Allocating Storage for Pods

StatefulSets

Configuration Data

StatefulSets

- A StatefulSet is like a Deployment designed for stateful applications
  - Manages pods that are based on an identical container spec
- A StatefulSet is different than a Deployment
  - Pods are created from the same spec, but are not interchangeable
    - Pods have a persistent identifier that it maintains across any rescheduling
  - Pods are given a stable name
  - Each pod is sequentially named
  - If a pod fails, it is replaced by a pod with the same name
  - Pods are linked to a stably named persistent storage

StatefulSets (continued)

- A StatefulSet can have one the following pod management policies:
  - **`OrderedReady`**: The default – pods are started one at a time sequentially
    - Each pod must achieve Running and Ready state before launching next pod
  - **`Parallel`**: Pods are launched in parallel without waiting for pods to start

VolumeClaimTemplates

- For stateful pods, each pod needs its own long-term storage
  - So it can maintain its own state
- StatefulSet pods are still susceptible to failure
  - Stable pod identifiers allow existing volumes to be matched to new pods that replace any that have failed
- StatefulSets use `VolumeClaimTemplates` to create unique PVCs for each pod

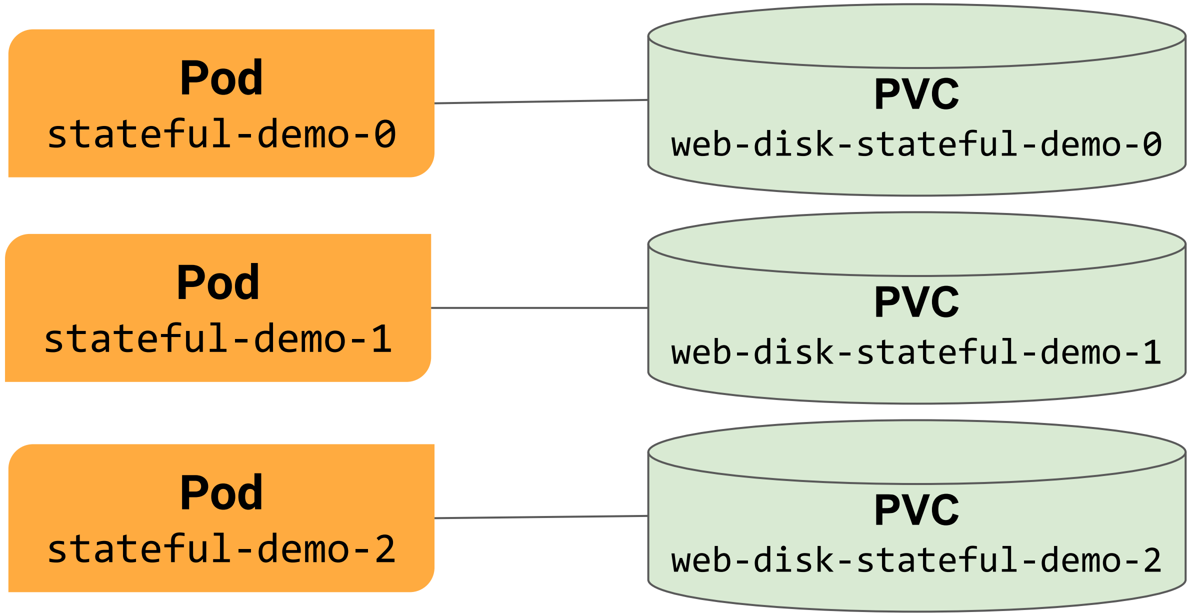
Persistent Storage

- When a pod from a StatefulSet is terminated (deleted, scaled down, etc.):
  - The associated volumes will not be deleted
- This ensures data persistence

StatefulSet Example

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: stateful-demo
spec:
  selector:
    matchLabels:
      app: MyApp
  serviceName: statefulset-demo-service
  replicas: 3
  template:
    metadata:
      labels:
        app: MyApp
    spec:
      containers:
      - name: stateful-set-container
        image: nginx
        ports:
        - containerPort: 80
          name: http
        volumeMounts:
        - name: web-disk
          mountPath: "/var/www/html"
```

```yaml
  volumeClaimTemplates:
  - metadata:
      name: web-disk
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 30Gi
```



Service for a StatefulSet

- StatefulSets currently require a service to control the network identity of the pods
  - You must create this service and reference it from the StatefulSet
    - Notice the following line on the previous slide:
      `serviceName: statefulset-demo-service`
  - Any service type is fine
- You may not want or need the load balancing and a single IP that a load balancer and ClusterIP service provide
  - Can create a headless service in this case

Service for a StatefulSet (continued)

- Below is a service that could be used with the previous StatefulSet demo

```yaml
kind: Service
apiVersion: v1
metadata:
  name: statefulset-demo-service
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
```

Activity: Deploying StatefulSets

In this activity, you will:

- Experiment with StatefulSets to provide persistent storage to pods

Chapter Concepts

Allocating Storage for Pods

StatefulSets

Configuration Data

ConfigMaps and Secrets

- ConfigMaps provide the ability to pass non-sensitive string data into pods
  - Configuration information like port numbers, command line variables, environment variables, etc.
- Secrets provide the ability to pass sensitive string data into pods
  - Passwords, keys, etc.
  - Only exposed to pods as needed

ConfigMaps and Secrets (continued)

- ConfigMaps and secrets both work similarly
- Can be accessed from a pod in three ways:
  - As a container environment variable
  - In pod commands
  - By creating a Volume

Kubernetes Secrets

- A secret is an object that contains a small amount of sensitive data
  - Such as a password, a token, or a key
- Putting this information in a secret is safer and more flexible than putting it in a pod definition or in a Docker image
  - A secret is only sent to a node if a pod on that node requires it
  - Not written to disk—stored in a tmpfs on the nodes
  - Deleted once the pod that depends on it is deleted
  - One pod does not have access to the secrets of another pod
  - Secrets are encrypted at the storage layer in etcd

Creating Kubernetes Secrets

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Not encrypted, base64 encoded

```
$ kubectl apply -f demo-secrets.yaml
$ kubectl get secret mysecret -o yaml
```

To get the secret

- For more info see: https://kubernetes.io/docs/concepts/configuration/secret/

Accessing Secret Data Using a Volume

- Within a Deployment, define a volumeMount that points to the secret
  - The secret data is available to containers in the pod through the volume

```
...
kind: Deployment
...
spec:
  containers:
    - name: demo-container
      image: nginx
      volumeMounts:
          - name: secret-vol
            mountPath: /secret-vol
  volumes:
    - name: secret-vol
      secret:
        secretName: mysecret
```

Other Secret Managers

- Using Kubernetes Secrets is not required
- Any secret manager can always be used
  - HashiCorp Vault
  - Google Secret Manager
  - AWS Secrets Manager
  - Etc.

Chapter Summary

In this chapter, you have:

- Allocated disk space for pods
- Implemented StatefulSet to run and maintain sets of pods
- Used configmaps and secrets to decouple configuration from pods

Helm

Chapter Objectives

In this chapter, you will:

- Install applications with Helm

Helm

- Helm is an open-source package manager for Kubernetes
  - Similar as apt-get and yum are package managers for Linux
- Organizes Kubernetes objects in packages called **charts**
  - A chart manages the deployment of complex applications
  - A chart is like a parameterized YAML template
  - When you install a Helm chart, Helm fills in the parameters you supply and deploys a release

Helm Repos

- A chart repository allows packaged charts to be stored and shared
- A community Helm chart repository is located at Artifact Hub
  - https://artifacthub.io/
- It is also possible to create and run your own chart repository
  - Creating your own is outside the scope of this course
  - For more information: https://helm.sh/docs/topics/chart_repository/
- To use a repo it must be added to the Helm client (initialized)

Using Helm

1. To use Helm, you first need a Kubernetes cluster with `kubectl` configured
2. Install the Helm client on the same system with `kubectl`
   - Helm provides binary releases for a variety of OSes
   - https://github.com/helm/helm/releases
3. Add a chart repository
   - Let's assume we want to install `redis` on a cluster
   - Go to https://artifacthub.io/ and search for `redis`
   - Several charts will be located, click the desired one and it will provide the command to add the repository for that chart
   - For example:

```
helm repo add bitnami https://charts.bitnami.com/bitnami
```

Using Helm (continued)

- Once a repo is initialized, it can be searched for charts it has to install

```
$ helm search repo bitnami
NAME                           CHART VERSION   APP VERSION    DESCRIPTION
bitnami/bitnami-common         0.0.9           0.0.9          DEPRECATED Chart with custom templates used in ...
bitnami/airflow                7.0.3           1.10.13        Apache Airflow is a platform to programmaticall...
bitnami/apache                 8.0.1           2.4.46         Chart for Apache HTTP Server
bitnami/aspnet-core            1.0.0           3.1.9          ASP.NET Core is an open-source framework create...
bitnami/cassandra              7.0.1           3.11.9         Apache Cassandra is a free and open-source dist...
bitnami/common                 1.1.1           1.1.1          A Library Helm Chart for grouping common logic ...
bitnami/consul                 9.0.3           1.9.0          Highly available and distributed service discov...
bitnami/contour                3.0.0           1.10.0         Contour Ingress controller for Kubernetes
bitnami/discourse              2.0.2           2.6.0          A Helm chart for deploying Discourse to Kubernetes
... <Many more> ...
```

Installing Applications with Helm

- To install a chart, you can run `helm install`
  - Provide the chart name

```
helm install my-redis bitnami/redis
helm install my-apache bitnami/apache
```

- To see what is installed, run `helm list`

```
$ helm list
NAME            NAMESPACE  REVISION   UPDATED                                  STATUS     CHART           APP
VERSION
my-apache       default    1          2021-09-26 15:56:46.782062447 +0000 UTC deployed    apache-8.0.1    2.4.46
my-redis        default    1          2021-09-26 15:56:58.910564104 +0000 UTC deployed    redis-12.1.1    6.0.9
```

Updating Applications with Helm

- An installed application can be updated with the `helm upgrade` command
  - Specify the installed application name and the repo

```
helm upgrade my-redis bitnami/redis
```

Deleting Applications with Helm

- To delete an application, use `helm uninstall`
  - This will remove all resources associated with the release as well as the release history
  - If the flag `--keep-history` is provided, release history will be kept
  - You will be able to request information about that release

```
helm uninstall my-redis
helm uninstall my-apache
```

Activity: Using Helm to Install a Database

In this activity, you will:

- Use Helm to install a relational database to your cluster

Chapter Summary

In this chapter, you have:

- Installed applications with Helm

Kubernetes Workloads

Chapter Objectives

In this chapter, you will:

- Perform tasks with Kubernetes Jobs
- Create CronJobs for jobs that run on a schedule
- Leverage DaemonSets to ensure all nodes run a copy of a pod

Chapter Concepts

Jobs

CronJobs

DaemonSets

Kubernetes Jobs

- A job creates one or more pods and ensures that a specified number of them terminate successfully
- Types of jobs
    - Non-parallel
    - Parallel
    - Cron job

Jobs Can Be Parallel or Non-Parallel

- Non-parallel jobs create only one pod at a time
  - Will be restarted if it terminates unsuccessfully
  - Job is completed when the pod terminates successfully
  - Or if a completion count is defined, when the required number of completions is completed
- Parallel jobs have multiple pods scheduled to work on the job at the same time
  - Used for tasks that must be completed more than once

Job Examples

```
apiVersion: batch/v1
kind: Job
metadata:
  name: demo-job
spec:
  template:
   spec:
     containers:
     - name: demo
       image: myjob:v1
  ...
```

```
apiVersion: batch/v1
kind: Job
metadata:
  name: demo-job
spec:
  completions: 3
  template:
    spec:
     containers:
     - name: demo
       image: myjob:v1
  ...
```

```
apiVersion: batch/v1
kind: Job
metadata:
  name: demo-parallel-job
spec:
  completions: 7
  parallelism: 2
  template:
    spec:
     containers:
     - name: demo
       image: myjob:v1
  ...
```

Job Retries

- Can specify the number of retries before failing a job
  - Set `backoffLimit` to the number of retries
  - The back-off limit default is six

Job to Compute PI to 2000 Places

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

```
$kubectl apply -f job.yaml
job.batch/pi created
$kubectl get pods
NAME                                  READY    STATUS       RESTARTS    AGE
pi-hxkgp                              0/1      Completed    0           22s
$kubectl logs pi-gm8kj
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421170679821480865132823066470938446095505822317253594081284811174502841027019385211055596446229489549303819644288109756 ......<omitted>
```

Managing Jobs

- You can create, inspect, and delete jobs using the `kubectl` commands:

```
kubectl apply -f <configuration-file>
```

```
kubectl get jobs
```

```
kubectl describe job <name>
```

```
kubectl delete job <name>
```

- When you delete a job, all pods are deleted
  - Use `--cascade false` to retain pods

```
kubectl delete job <name> --cascade false
```

Chapter Concepts

Jobs

CronJobs

DaemonSets

CronJob Is Used to Schedule Jobs

- CronJobs are jobs that are created in a repeatable manner on a defined schedule
  - Uses the standard cron syntax for scheduling

```
Min  Hour Day  Mon  Weekday

*    *    *    *    *   command to be executed

┬    ┬    ┬    ┬    ┬
│    │    │    │    └─  Weekday  (0=Sun .. 6=Sat)
│    │    │    └──────  Month    (1..12)
│    │    └───────────  Day      (1..31)
│    └────────────────  Hour     (0..23)
└─────────────────────  Minute   (0..59)
```

Cron Syntax

- An asterisk (*) stands for the entire range of the possible values
  - Each minute, each hour, etc.
- Any field may contain a list of values separated by commas (e.g., 1,3,7)
  - Or a range of values (e.g., 1-5)
- After an asterisk (*) or a range of values, you can use the slash character (/) to specify that values are repeated over and over with a certain interval between them
- For example, the following cron expression would run once every minute

```
*/1 * * * *
```

CronJob Example

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: crondemo
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            command:
            - /bin/sh
            - -c
            - date; echo Hello from Kubernetes cronjob demo
          restartPolicy: OnFailure
```

CronJob Concurrency

- A CronJob could potentially still be executing at the next scheduled time
  - The job did not finish yet
- Use **`concurrencyPolicy`** to define if concurrent executions are permitted
  - **`Forbid`**
    - If the existing job hasn't finished, the CronJob will not execute a new job
  - **`Replace`**
    - Existing job will be replaced by the new job
  - **`Allow`**

```
apiVersion: batch/v1
kind: CronJob
metadata:
   name: cronjob-demo
spec:
   schedule: "*/1 * * * *"
   concurrencyPolicy: Forbid
   jobtemplate:
...
```

Managing CronJobs

- You can create, inspect, and delete CronJobs using the `kubectl` commands:

```
kubectl apply -f <configuration-file>
```

```
kubectl get cronjobs
```

```
kubectl describe cronjob <name>
```

```
kubectl delete cronjob <name>
```

Chapter Concepts

Jobs

CronJobs

DaemonSets

DaemonSets

- Kubernetes nodes are recreated during upgrade, repair, and scaling
  - What if you want to install something on each node?
- A DaemonSet ensures that all nodes run a copy of a pod
  - If a node is added to the cluster, the DaemonSet Controller adds the pod to that node
  - If a node is replaced for any reason, the daemon set is reinstalled

DaemonSet Uses

- DaemonSets are good for long-running services, such as:
  - Cluster storage daemon on every node
  - Logs collection daemon on every node
  - Node monitoring daemon on every node
- Managed clusters often have several DaemonSets installed by default
  - For example, on GKE:

```
$ kubectl get daemonsets -n kube-system
NAME                         DESIRED    CURRENT    READY    UP-TO-DATE    AVAILABLE    NODE SELECTOR                                                          AGE
fluentbit-gke                3          3          3        3             3            kubernetes.io/os=linux                                                 8h
gke-metrics-agent            3          3          3        3             3            kubernetes.io/os=linux                                                 8h
gke-metrics-agent-windows    0          0          0        0             0            kubernetes.io/os=windows                                               8h
kube-proxy                   0          0          0        0             0            kubernetes.io/os=linux,node.kubernetes.io/kube-proxy-ds-ready=true     8h
metadata-proxy-v0.1          0          0          0        0             0            cloud.google.com/metadata-proxy-ready=true,kubernetes.io/os=linux      8h
nvidia-gpu-device-plugin     0          0          0        0             0            <none>                                                                 8h
pdcsi-node                   3          3          3        3             3            kubernetes.io/os=linux                                                 8h
pdcsi-node-windows           0          0          0        0             0            kubernetes.io/os=windows                                               8h
workload-metrics             3          3          3        3             3            kubernetes.io/os=linux                                                 8h
```

DaemonSet Example

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      containers:
      - name: fluentd-elasticsearch
        image: quay.io/fluentd_elasticsearch/fluentd:v2.5.2
        resources:
          limits:
            memory: 200Mi
            cpu: 500m
        volumeMounts:
        - name: varlog
 ... <remaining volume info omitted> ...
```

Activity: Running Kubernetes Jobs

In this activity, you will:

- Create a job to perform database initialization required for the case study
- Create a CronJob to perform scheduled work

Chapter Summary

In this chapter, you have:

- Performed tasks with Kubernetes Jobs
- Created CronJobs for jobs that run on a schedule
- Leveraged DaemonSets to ensure all nodes run a copy of a pod

Role-Based Access Control and Advanced Networking

Chapter Objectives

In this chapter, you will:

- Control access to Kubernetes resources with role-based access control
- Create pod-level firewall rules with network policies

Chapter Concepts

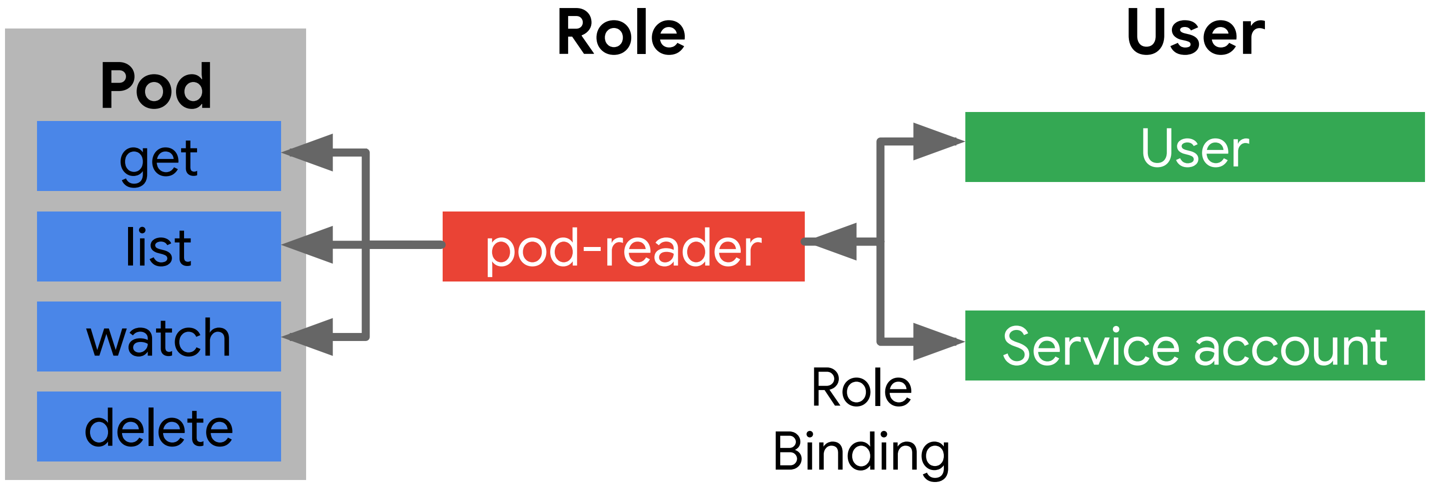
RBAC

Network Policies

Istio Introduction

Role-Based Access Control (RBAC)

- Kubernetes RBAC gives you access control inside your Kubernetes clusters
    - At the cluster level and the namespace level
    - Define who can view or change Kubernetes objects in a cluster
- RBAC allows you to define roles with rules containing a set of permissions
    - The roles can then be bound to users



Kubernetes Users

- There are two types of users in Kubernetes
  - Service accounts
    - Managed by Kubernetes
    - Each namespace has a default service account
  - Normal users
    - Users can also be used in groups
- Clusters must be configured to use authentication modules that identify normal users
  - Managed cloud-based clusters handle this for you
  - For example: when using GKE, all users are defined in Google IAM

Role and ClusterRole

- Role or ClusterRole define rules that represent a set of permissions
  - Permissions are purely additive (there are no “deny” rules)
  - A Role always sets permissions within a particular namespace
  - ClusterRole sets permissions cluster wide
- A RoleBinding (or ClusterRoleBinding) grants the permissions in the Role (or ClusterRole) to a set of users
  - Contains a list of the users, and a reference to the Role (or ClusterRole) being granted to those users

Not All Resources Are Namespaced

- When creating Roles and ClusterRoles, it's important to know whether a resource is associated with a namespace or defined at the cluster level
  - Use the `kubectl api-resources` to list which resources are namespaced

```
$ kubectl api-resources
NAME                           SHORTNAMES   APIVERSION          NAMESPACED   KIND
bindings                                    v1                  true         Binding
componentstatuses              cs           v1                  false        ComponentStatus
configmaps                     cm           v1                  true         ConfigMap
endpoints                      ep           v1                  true         Endpoints
events                         ev           v1                  true         Event
limitranges                    limits       v1                  true         LimitRange
namespaces                     ns           v1                  false        Namespace
nodes                          no           v1                  false        Node
persistentvolumeclaims         pvc          v1                  true         PersistentVolumeClaim
persistentvolumes              pv           v1                  false        PersistentVolume
pods                           po           v1                  true         Pod
podtemplates                                v1                  true         PodTemplate
replicationcontrollers         rc           v1                  true         ReplicationContro
...
```

Defining a Role (or ClusterRole)

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: dev
  name: pod-reader
rules:
- apiGroups: [""] # "" indicates core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
```

- Example **Role** in the “dev” namespace that can be used to grant read access to pods

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

- Example **ClusterRole** to grant read access to secrets in any particular namespace, or across all namespaces
    - Depending on how it’s bound

How to Refer to Resources

- The Role rules specify which resources the role applies to
  - Can reference resources several ways

```
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "list", "watch"]
```

```
rules:
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["get", "list", "watch"]
```

```
rules:
- apiGroups: [""]
  resources: ["pods"]
  resourceNames: ["demo-pod"]
  verbs: ["patch", "update"]
```

Binding a Role

- This Role binding assigns “john@roitraining.com” the pod-reader Role in the “dev” namespace

```
kind: RoleBinding # must be RoleBinding or ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: dev
subjects:
- kind: User
  name: john@roitraining.com
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role # must be Role or ClusterRole
  name: pod-reader # must match a Role or ClusterRole name
  apiGroup: rbac.authorization.k8s.io
```

How to Refer to Subjects

- The Role rules specify which resources the Role applies to
  - Can references resources several ways

```
subjects
- kind: User
  name: "john@roitraining.com"
  apiGroup: rbac.authorization...
```

```
subjects
- kind: Group
  name: system.serviceaccounts:dev
  apiGroup: rbac.authorization...
```

```
subjects
- kind: Group
  name: "Developers"
  apiGroup: rbac.authorization...
```

```
subjects
- kind: Group
  name: system.authenticated
  apiGroup: rbac.authorization...
```

Service Accounts

- Containers inside pods can also contact the control plane apiserver
  - When they do, they are authenticated as a service account
- Every namespace has a default service account called default
  - Additional ServiceAccounts can be created
- When a pod is created, it is being assigned a service account
  - Pods are automatically assigned the default service account in the same namespace if no service account is specified
  - You cannot update the service account of an existing pod
- You can use authorization plugins to set permissions on service accounts
  - Outside the scope of this course

Activity: Using Kubernetes RBAC

In this activity, you will:

- Control cluster permissions with RBAC
- Limit what a user can access on a cluster

Chapter Concepts

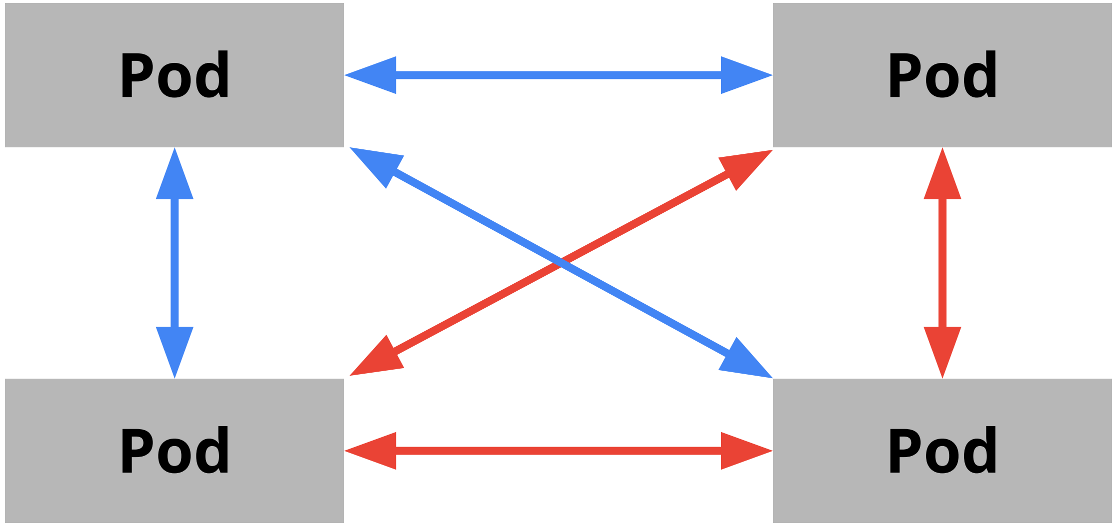
RBAC

Network Policies

Istio Introduction

Restrict Traffic Among Pods

- By default, all pods in a cluster can communicate with each other
  - Even if in different namespaces
- Pod-to-pod communication should be controlled for your applications
  - Make it more difficult for attackers to move laterally within your cluster
  - Lock down traffic to allow only legitimate network traffic



Network Policies Can Restrict Network Traffic

- Network policies are pod-level firewall rules
  - Restrict access based on connection direction, source, destination, port number, namespace, etc.
- Network policies use selectors to select pods
  - If any NetworkPolicy selects a particular pod, any connections not allowed by any NetworkPolicy are dropped
  - Other pods that are not selected by any NetworkPolicy will continue to allow connections

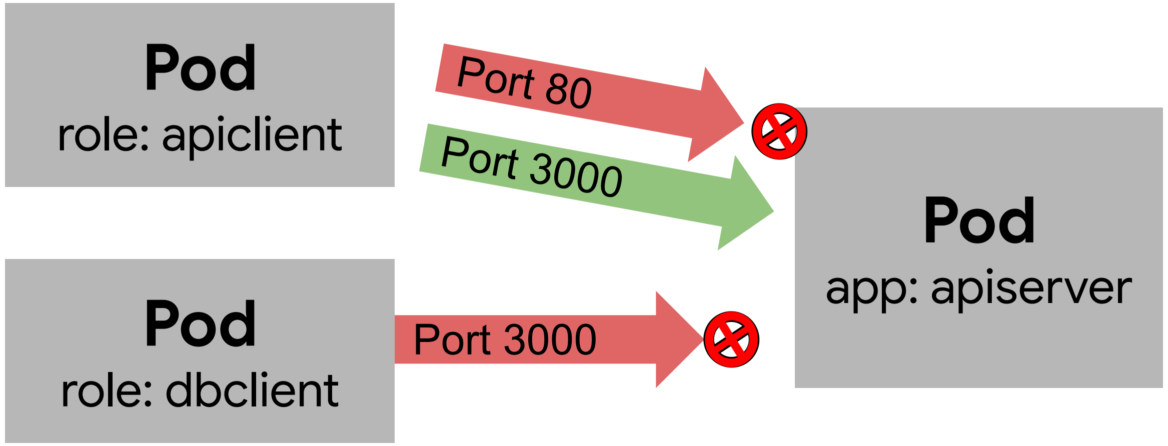
Network Policies Can Restrict Network Traffic (continued)

- Network policies are not enabled on clusters by default
  - Are implemented by the network plugin
- Project Calico is a network policy engine for Kubernetes
  - https://www.tigera.io/project-calico/
  - Can be installed on any Kubernetes cluster
- GKE clusters manage the Calico network policy engine
  - Simply enable it in the cluster properties
  - GKE Dataplane V2 will have network policies built in

Allow Traffic Only to Certain Port

- Allow traffic to pods labeled `app:apiserver` on port 3000
  - Only from pods labeled `app:apiclient`

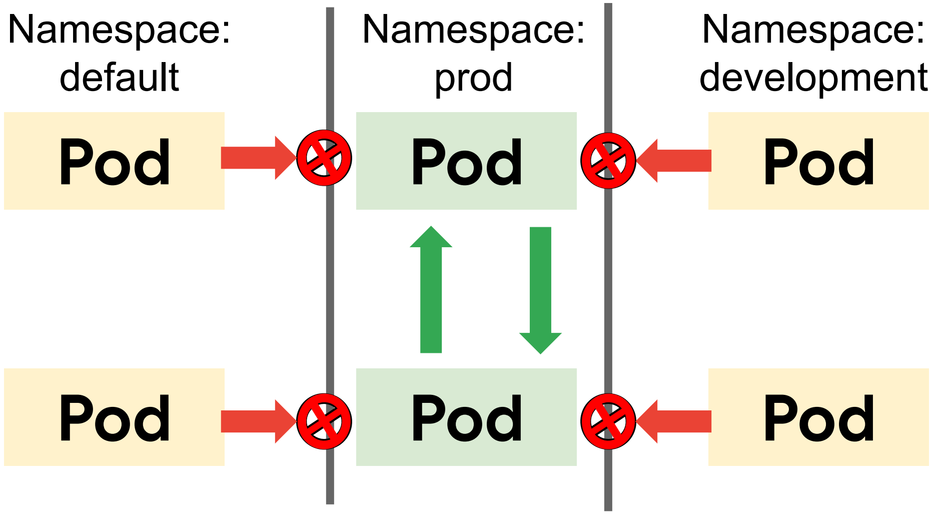
```yaml
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-port-3000
spec:
  podSelector:
    matchLabels:
      app: apiserver
  ingress:
  - ports:
    - port: 3000
    from:
    - podSelector:
        matchLabels:
          role: apiclient
```



Deny Traffic from Other Namespaces

- The following policy will deny all the traffic from other namespaces
  - While allowing all the traffic coming from the same namespace (prod)

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: prod
  name: deny-from-other-namespaces
spec:
  podSelector:
    matchLabels:
  ingress:
  - from:
    - podSelector: {}
```



Activity: Network Policies

In this activity, you will:

- Implement pod-level firewall rules with a network policy

Chapter Concepts

RBAC

Network Policies

Istio Introduction

Service Mesh

- A service mesh is a mesh of Layer 7 proxies that microservices can use to completely abstract the network away
  - Solve many challenges developers face when talking to remote endpoints
- A service mesh consists of:
  - Network proxies paired with each service in an application
    - Called the data plane
  - And a set of task management processes
    - Called the control plane
- Basically, a layer that lives on top of your workloads
  - Does not modify them
  - Allows you to manage network interactions in a consistent way

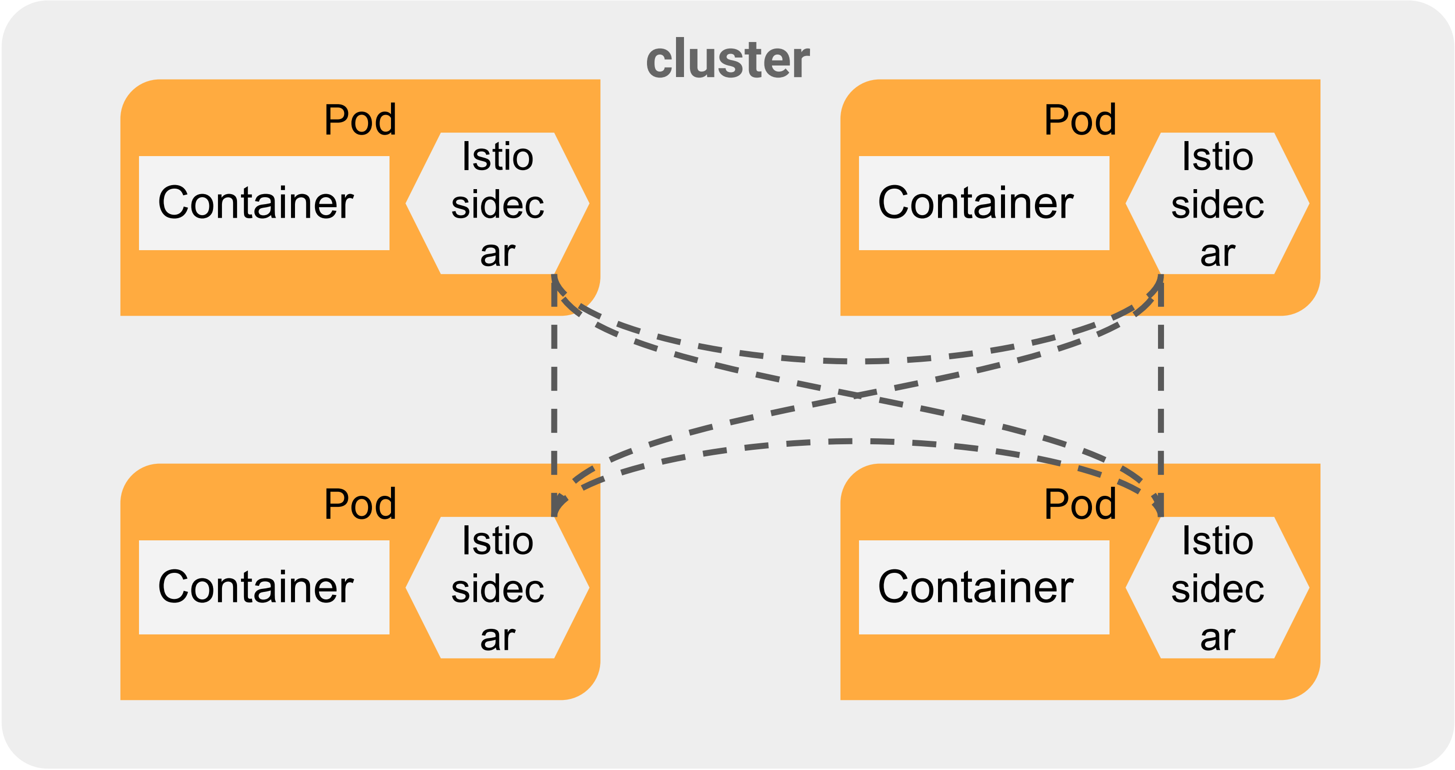
Istio

- Open-source service mesh
  - Designed to make it easier to connect, manage, and secure traffic between microservices running in containers
  - And obtain telemetry about the microservices running in containers

Istio Data Plane

- Typically, composed of Envoy proxies that are deployed as sidecars in an extra container within the Kubernetes pods
  - These proxies take on the task of establishing connections to other services and managing the communication between them
- The sidecar injection into the application pods happens automatically
  - Manual injection is also possible
- Traffic is directed from the application services to and from these sidecars without developers needing to worry about it
- Once the applications are connected to the Istio service mesh, developers can start using and reaping the benefits of all that the service mesh has to offer

Istio Data Plane (continued)



= Envoy Proxy

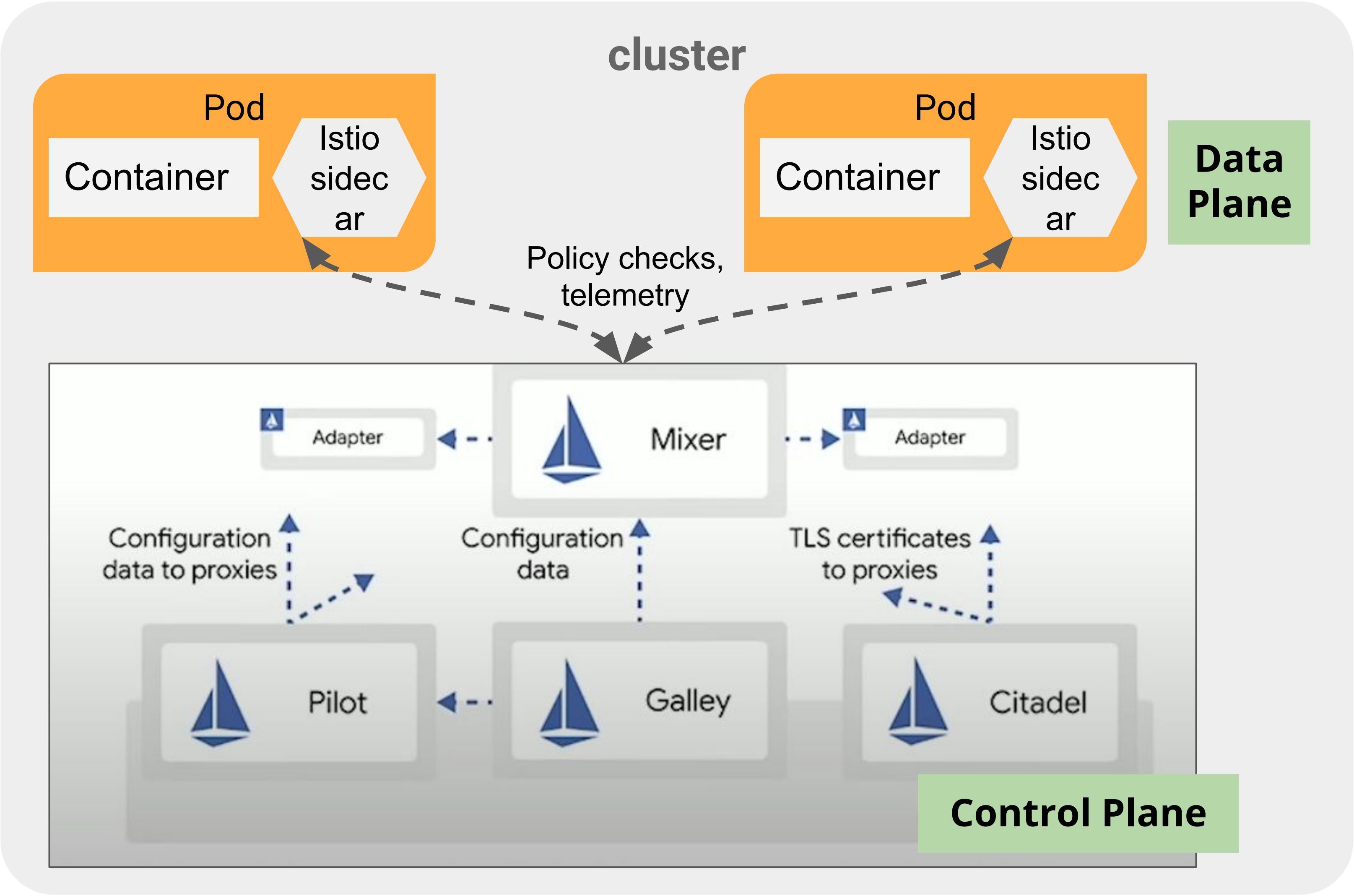
Istio Control Plane

- Istio control plane is deployed to Kubernetes
  - Installs its API as Custom Resource Definitions (CRDs)
  - Can interact with Istio policies, rules, etc. using Kubernetes native tooling such as `kubectl`

Istio Control Plane (continued)

- Mixer
    - Enforces access control and usage policies across the service mesh, and collects telemetry data from the Envoy proxy and other services
- Pilot
    - Provides service discovery for the Envoy sidecars, traffic management capabilities for intelligent routing and resiliency
- Citadel
    - Provides strong service-to-service and end-user authentication with built-in identity and credential management
- Galley
    - Validates user authored Istio API configuration on behalf of the other Istio control plane components

Istio Architecture



What Does Istio Do for You?

- Understand network interactions between services
  - I.e., which microservices is calling which
- Traffic inspection between services
  - I.e., make decision based on things like HTTP headers
- Granular percentage-based routing
  - I.e., very specific canary release routing (95%, 5%)
- Automation across all deployed services
  - I.e., turn on mTLS end-to-end encryption for entire mesh with a single rule
- Decouple the network from application code
  - I.e., firewall rules and retry logic can be managed in a centralized way

Chapter Summary

In this chapter, you have:

- Controlled access to Kubernetes resources with role-based access control
- Created pod-level firewall rules with network policies

Multi-Cloud Support

Chapter Objectives

In this chapter, we will:

- Deploy to Kubernetes on AWS, Google Cloud, or Microsoft Azure

Kubernetes Is Open Source

- Can run in virtually any environment
  - Public cloud, private cloud
  - On-premises data centers
  - Virtual machines or physical hosts
  - Even your laptop
- Wide support
  - Google Cloud Google Kubernetes Engine (GKE)
  - Microsoft Azure Kubernetes Service (AKS)
  - AWS Elastic Kubernetes Service (EKS)
  - Red Hat OpenShift
  - Meso and Pivotal Cloud Foundry support Kubernetes

Minor Differences

- During this course, a GKE cluster was used
- The core functionality of Kubernetes is generally the same on all providers
  - Every Kubernetes lab and activity should work the same on any Kubernetes cluster
- ***Note:*** features that each cloud provider offers can be different
  - For example, GKE has many fully automated cluster features
    - Installing components like Istio and Network Policy engine is simply a flag
    - Upgrading clusters is automatic on GKE
    - These may be manual on other providers
- Pricing on different cloud providers can also be different
  - Be sure to check their pricing calculators

MiniKube

- You can even continue your learning using just your local computer
  - Install MiniKube
  - Follow steps here: https://minikube.sigs.k8s.io/docs/start/
- The course case study should deploy to MiniKube just fine

Demonstration: Deploying the Class Case Study to Another Cloud Provider

In this demonstration, your instructor will:

- Create a Kubernetes cluster on another cloud provider
- Configure `kubectl` for that cluster
- Deploy the class case study to the new cluster

Chapter Summary

In this chapter, we have:

- Deployed to Kubernetes on AWS, Google Cloud, and Microsoft Azure

Course Summary

Course Summary

In this course, you have learned how to:

- Build, run, and deploy container applications using Docker
- Configure Kubernetes clusters in the cloud
- Automate application management using the `kubectl` CLI and configuration
- Deploy scalable, fault-tolerant applications using Kubernetes Deployments, Services, Jobs, CronJobs, and DaemonSets
- Migrate to new versions of services safely with zero downtime
- Save data in Kubernetes using Persistent Volumes, StatefulSets, ConfigMaps, and Secrets
- Simplify Kubernetes deployments with Helm
- Manage Kubernetes security with Role-Based Access Control (RBAC)
- Enhance Kubernetes networking with network policies and Istio

ROI's Training Curricula

- Agile Development
- Amazon Web Services (AWS)
- Azure
- Big Data and Data Analytics
- Business Analysis
- Cloud Computing and Virtualization
- Excel and VBA
- Google Cloud
- ITIL® and IT Service Management
- Java
- Leadership and Management Skills
- Machine Learning and Neural Networks
- Microsoft Exchange
- .NET and Visual Studio
- Networking and IPv6
- Oracle and SQL Server Databases
- OpenStack and Docker
- Project Management
- Python and Perl Programming
- Security
- SharePoint
- Software Analysis and Design
- Software Engineering
- UNIX and Linux
- Web and Mobile Apps
- Windows and Windows Server

Our full course list is available at www.ROItraining.com