Kubernetes for Developers: Core Concepts

KUBERNETES FROM A DEVELOPER PERSPECTIVE

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Course Overview

- Kubernetes from a Developer Perspective
- Understanding Storage Options
- Creating Pods
- Creating ConfigMaps and Secrets
- Creating Deployments
- Putting It All Together
- Creating Services
- Course Summary

Target Audience

Developers looking to understand Kubernetes core concepts

Course Prereqs

- Comfortable using command-line tools and virtual machines
- General familiarity with software development
- Understanding of Docker containers and how they work

Introduction

Module Overview

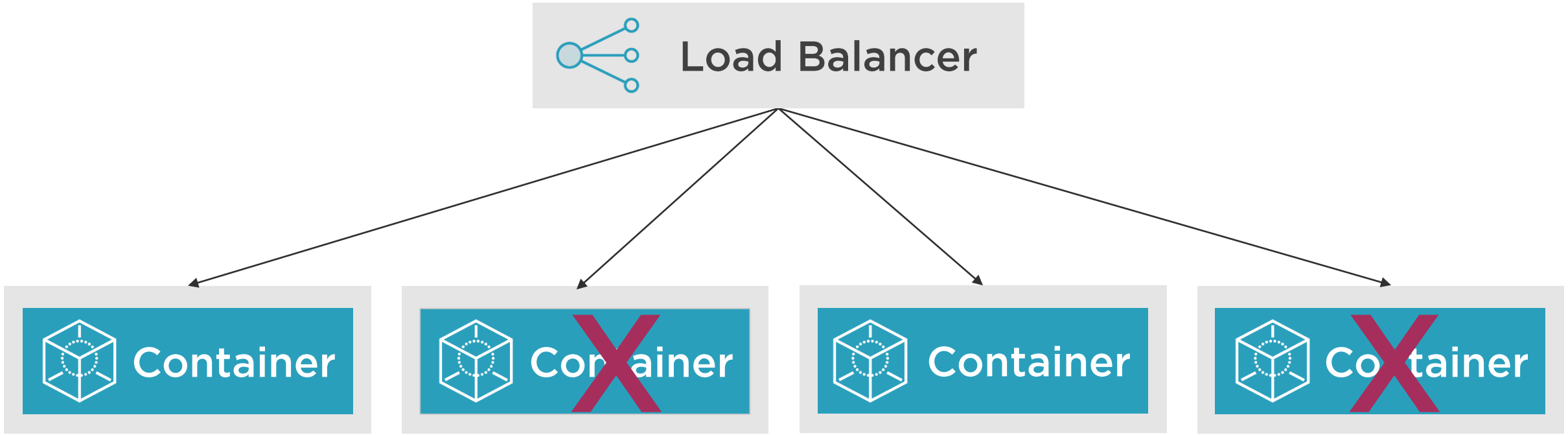
- Kubernetes Overview
- The Big Picture
- Benefits and Use Cases
- Running Kubernetes Locally
- Getting Started with kubectl
- Web UI Dashboard

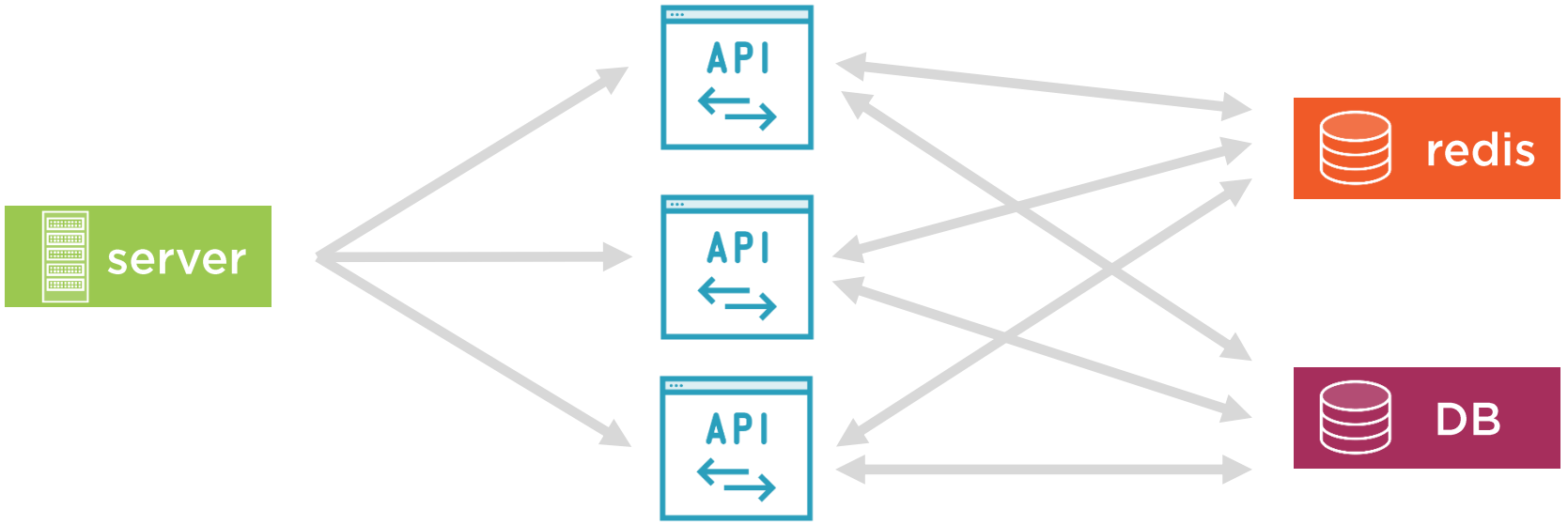
Kubernetes Overview

"Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications."

https://kubernetes.io

How Are You Managing Containers?





How do you manage all of these containers?

It Would Be

Nice if We Could...

- Package up an app and let something else manage it for us
- Not worry about the management of containers
- Eliminate single points of failure
- Scale containers
- Update containers without bringing down the application
- Have robust networking and persistent storage options


Container
Container
Container
Kubernetes
(conductor)

Kubernetes is the conductor of a container orchestra.

Key Kubernetes Features

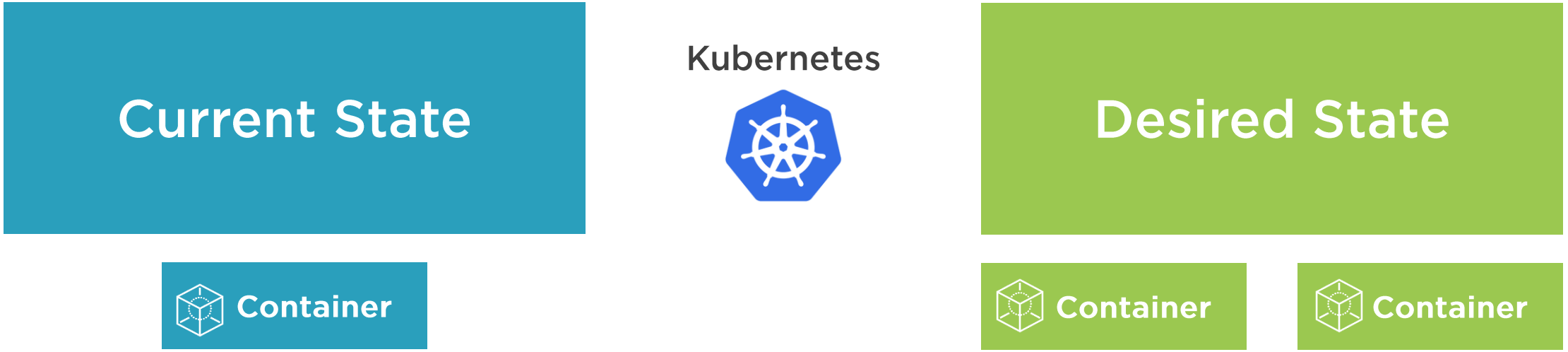
- Service Discovery/ Load Balancing
- Storage Orchestration
- Automate Rollouts/Rollbacks
- Self-healing
- Secret and Configuration Management
- Horizontal Scaling

The Big Picture

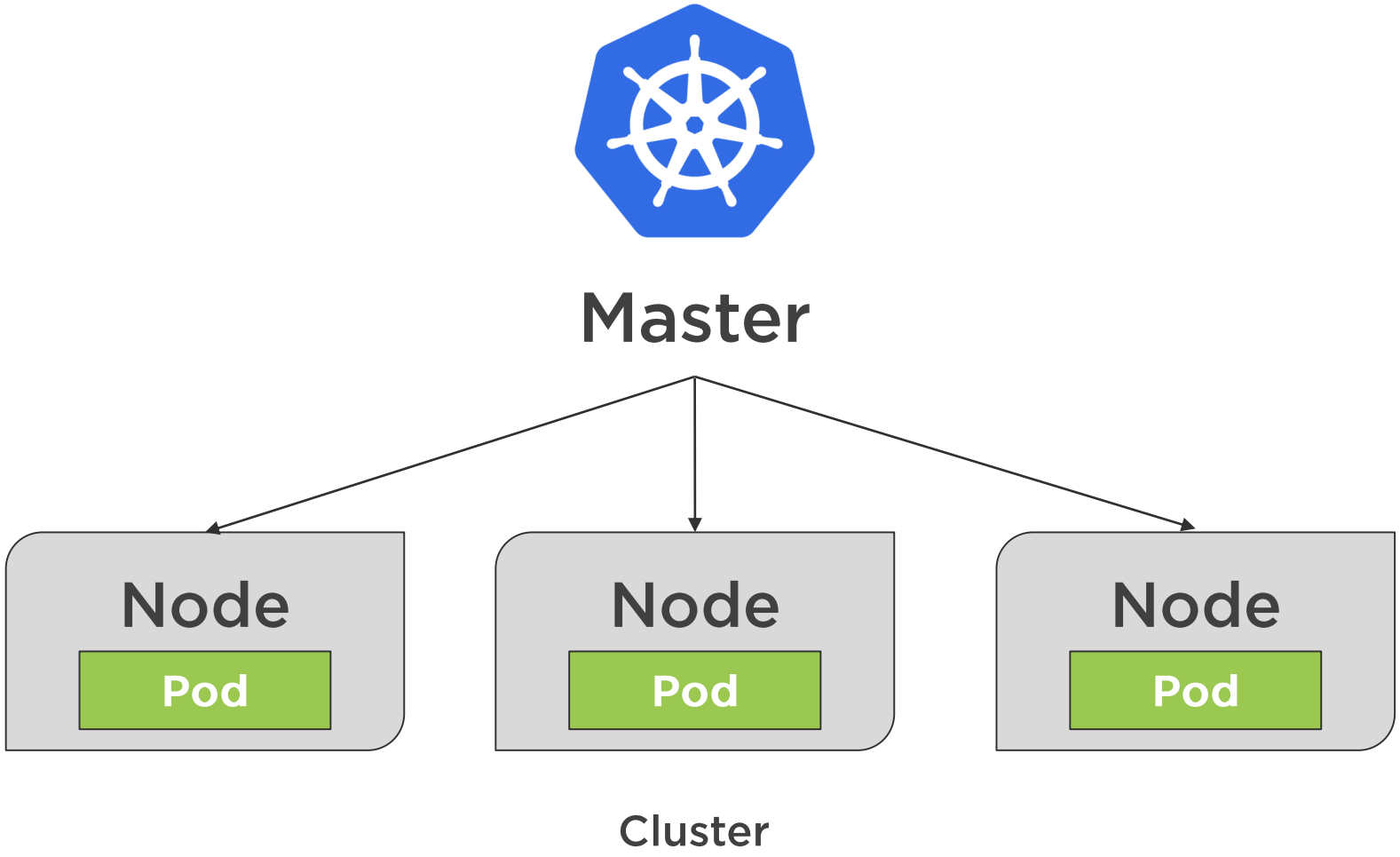
Kubernetes

- Container and cluster management
- Open source project
- Used internally by Google for 15+ years and donated to the Cloud Native Computing Foundation
- Supported by all major cloud platforms
- Provides a "declarative" way to define a cluster's state

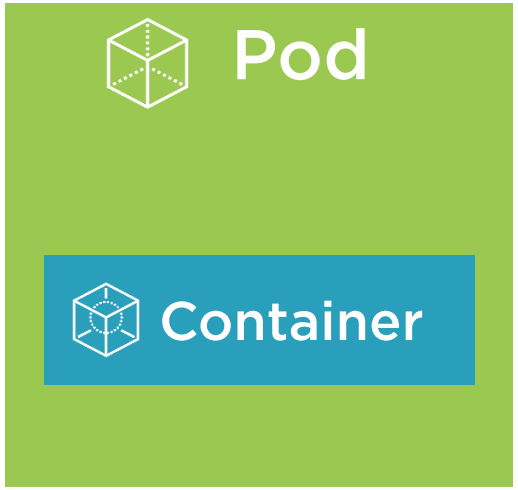
Kubernetes Moves You to a Desired State



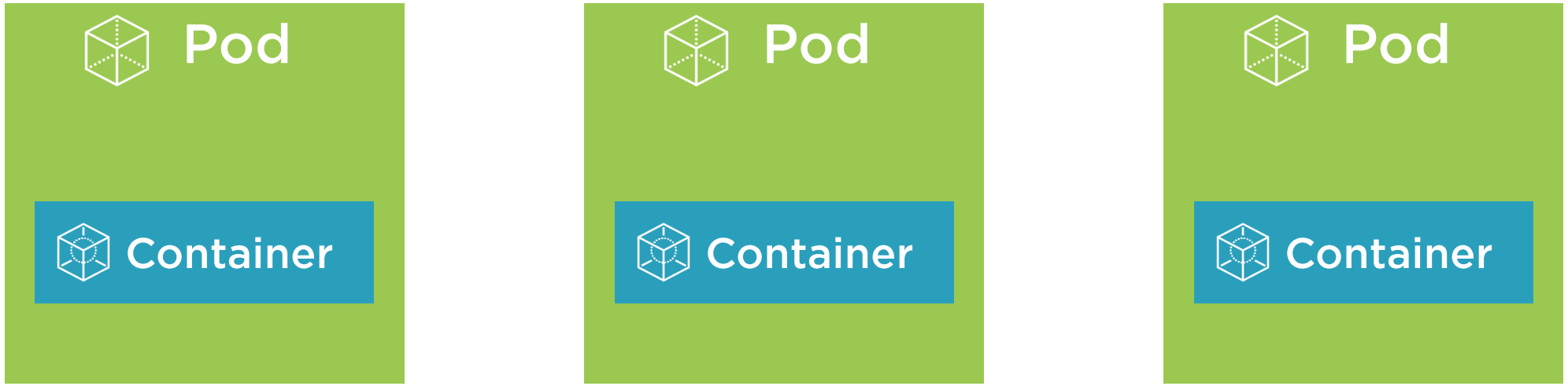
The Big Picture

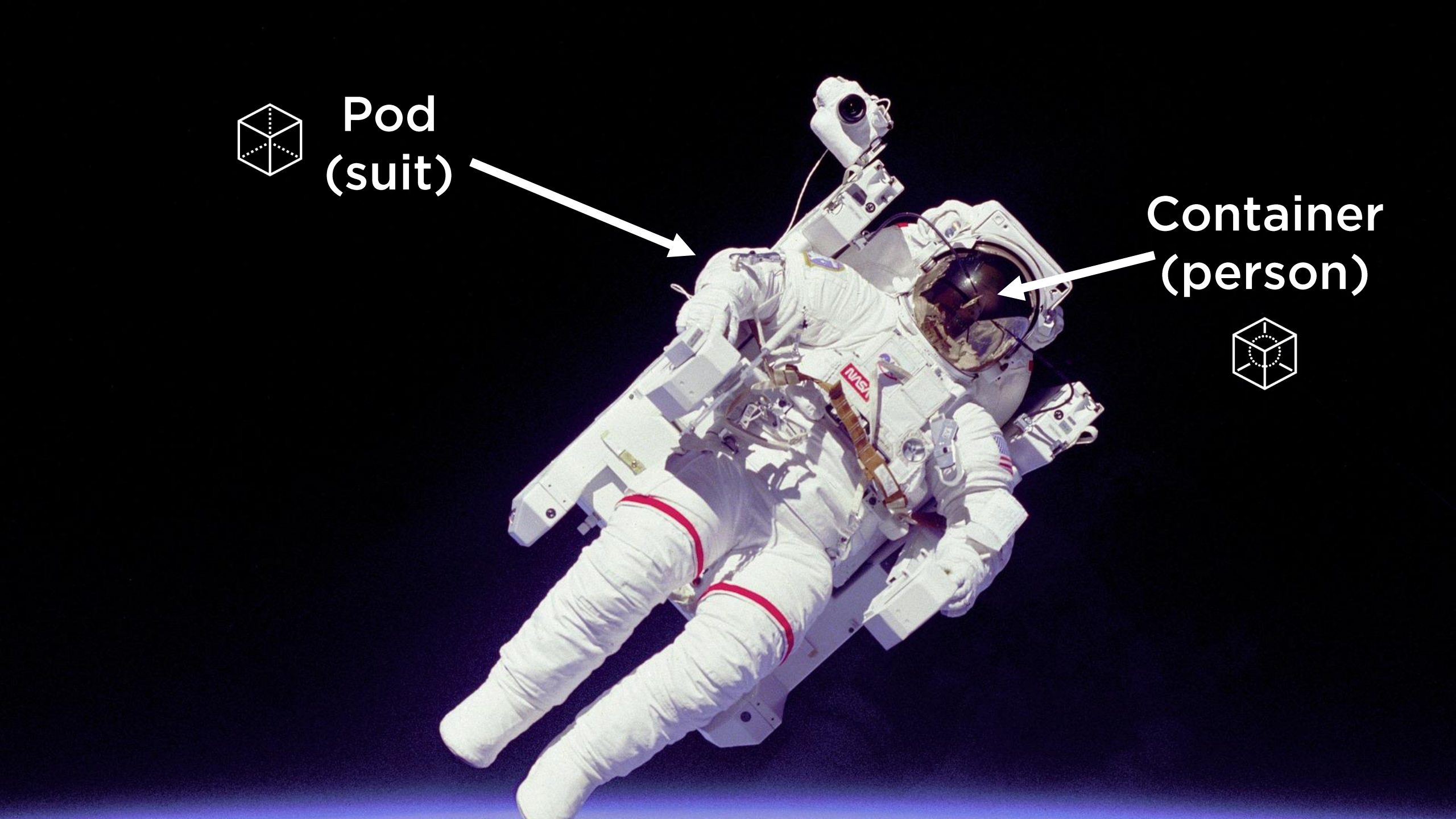


Pods and Containers

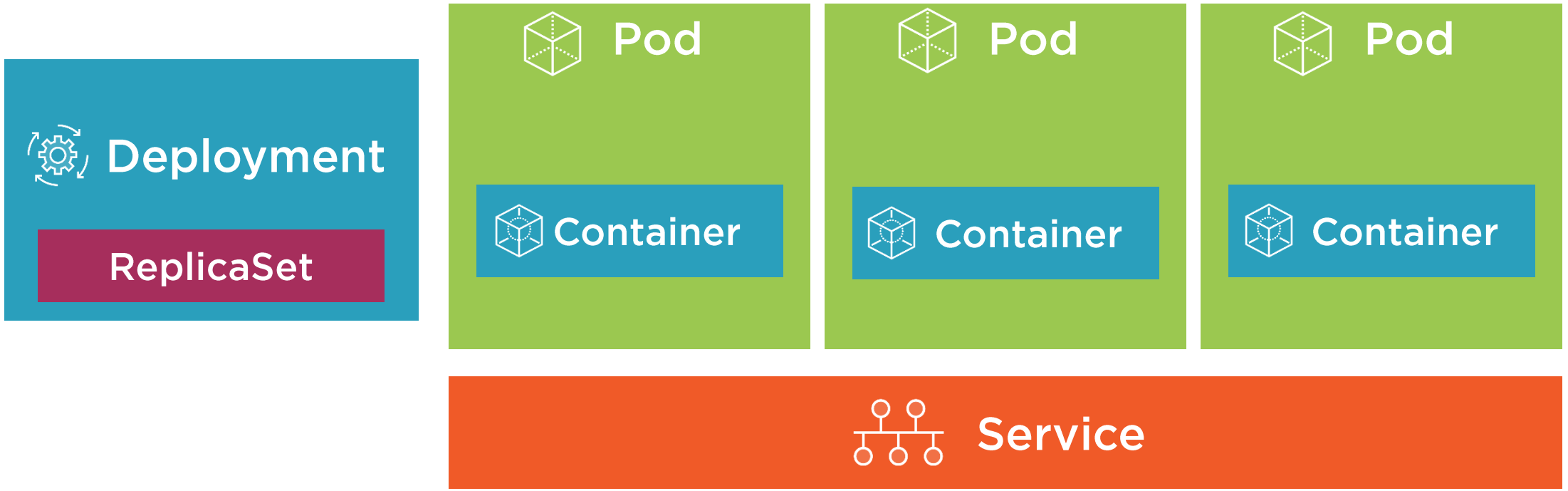


Pods and Containers

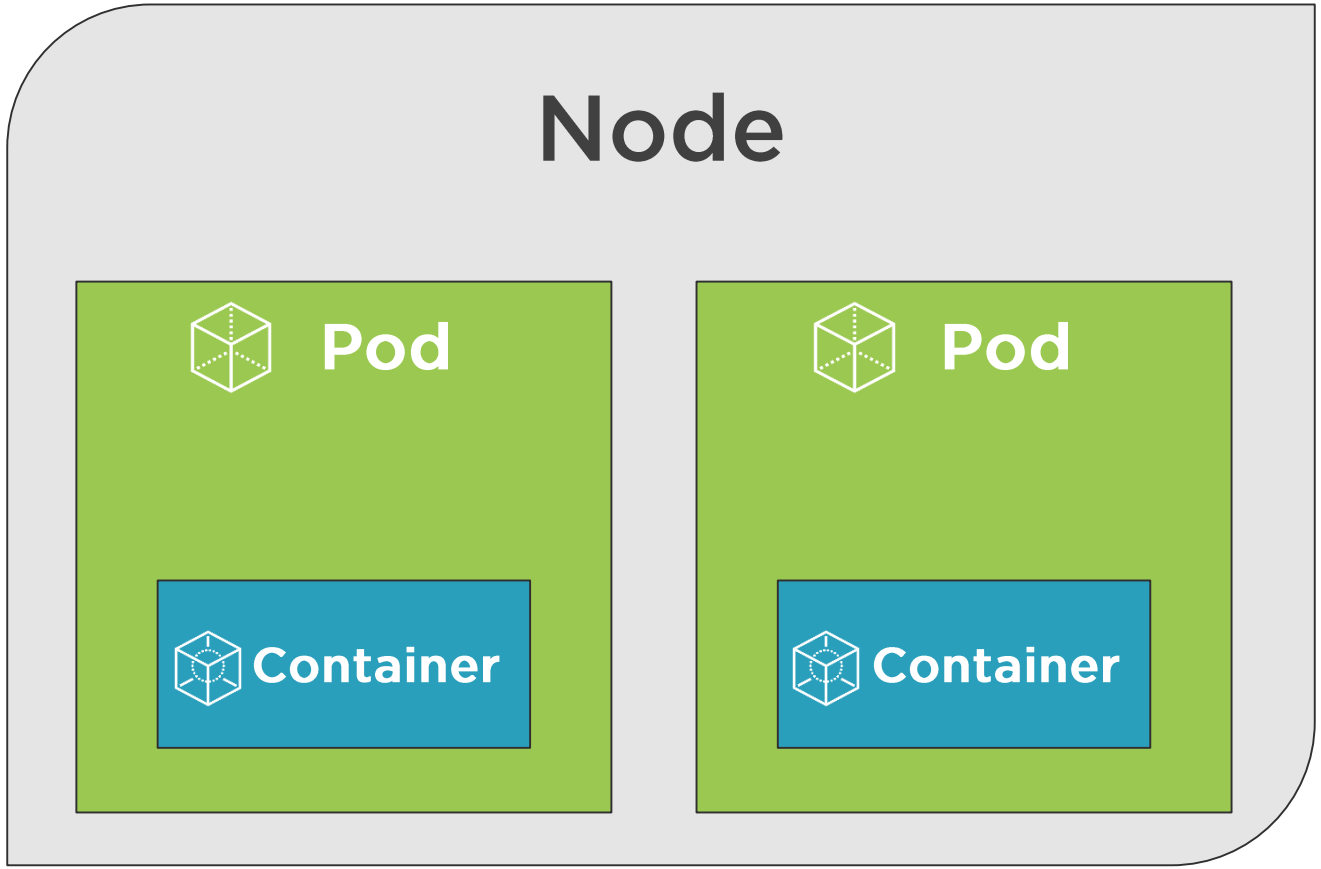



Pod
(suit)
Container
(person)

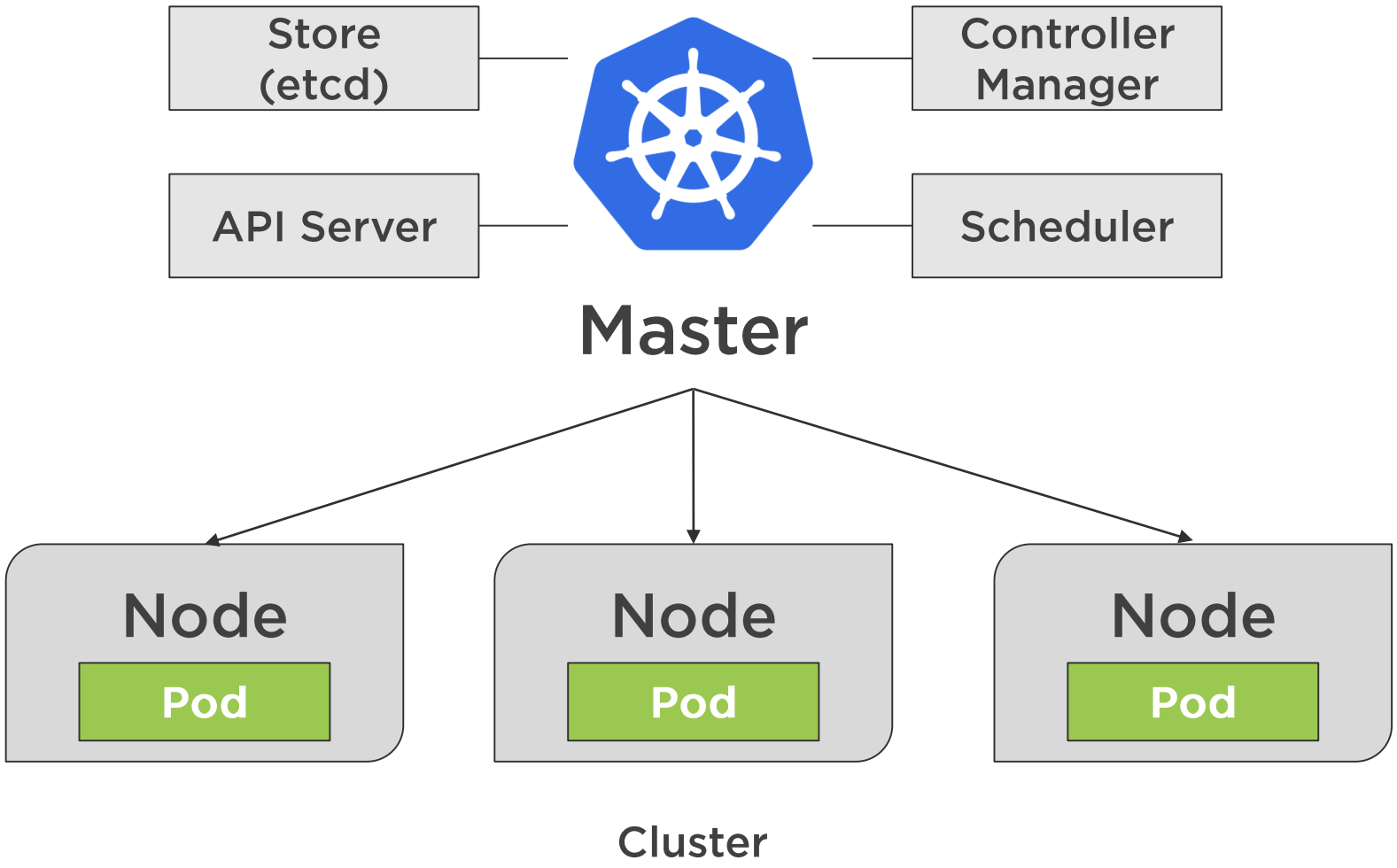
Kubernetes Building Blocks



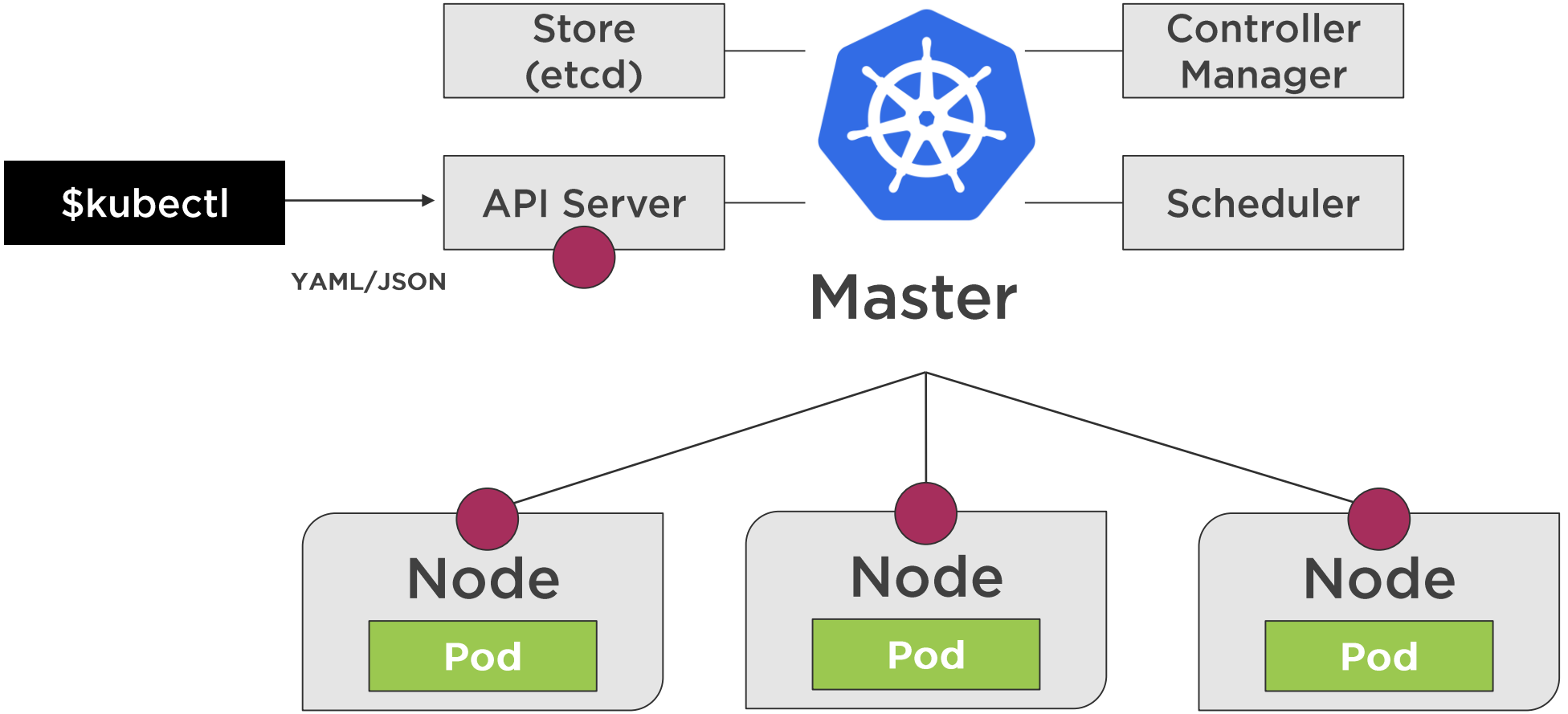
Pods and Nodes



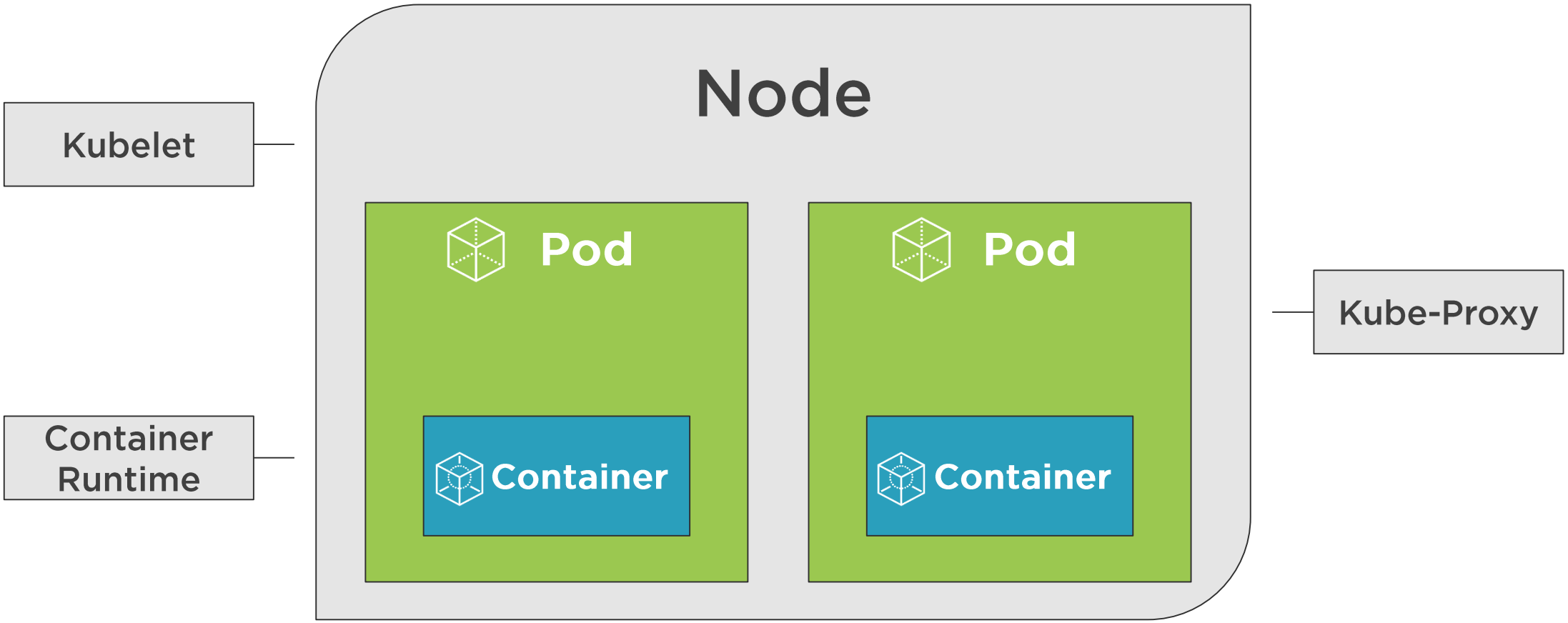
The Master Node



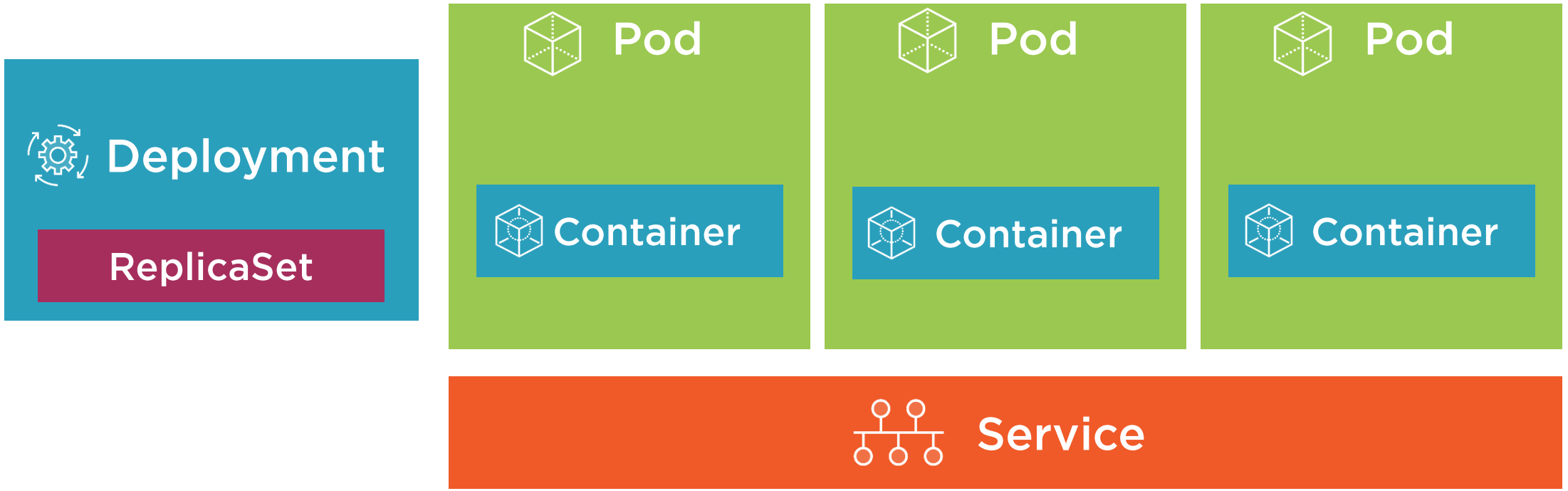
Communicating with kubectl



Kubernetes Nodes



The Big Picture

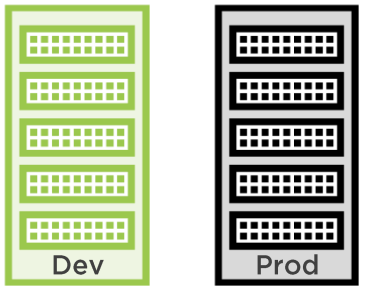


Benefits and Use Cases

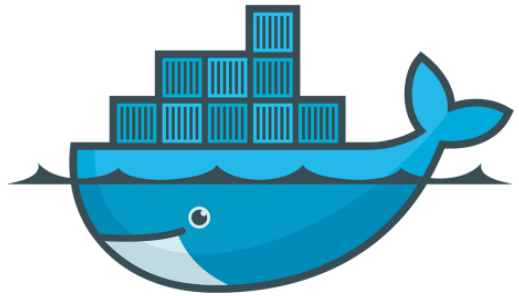
Key Container Benefits

Accelerate Developer Onboarding

Eliminate App Conflicts

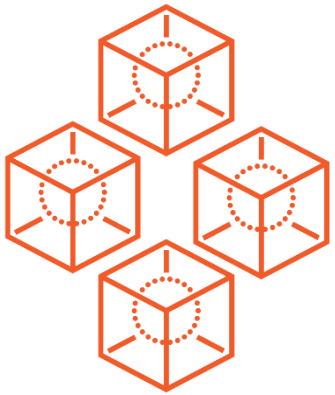


Environment Consistency

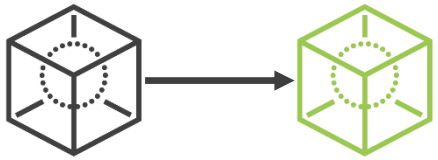


Ship Software Faster

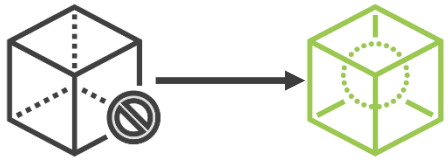
Key Kubernetes Benefits



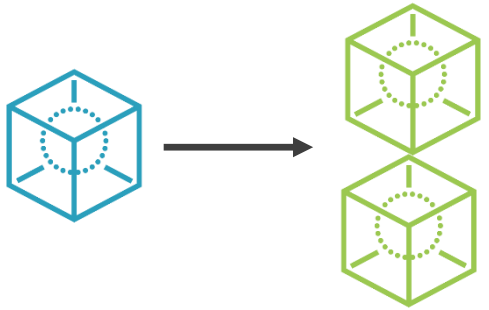
Orchestrate Containers



Zero Downtime Deployments



Self-healing (Superpowers)



Scale Containers

Developer Use Cases

- Emulate production locally
- Move from Docker Compose to Kubernetes
- Create an end-to-end testing environment
- Ensure application scales properly
- Ensure secrets/config are working properly
- Performance testing scenarios
- Workload scenarios (CI/CD and more)
- Learn how to leverage deployment options
- Help DevOps create resources and solve problems

Running Kubernetes Locally

Installing and Running Kubernetes

Minikube

https://github.com/kubernetes/minikube

Docker Desktop

https://www.docker.com/products/docker-desktop

Installing and Running Kubernetes

kind

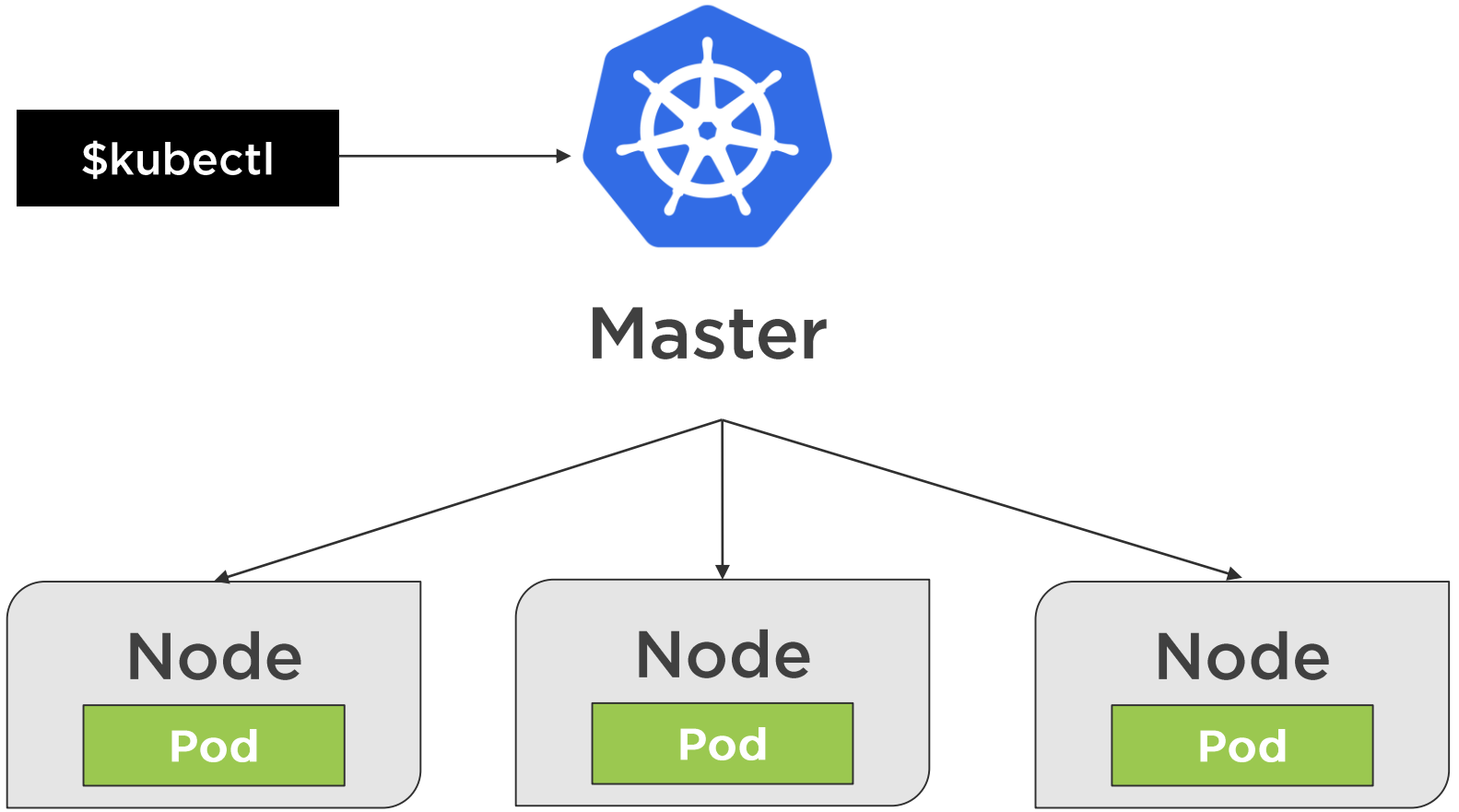
https://kind.sigs.k8s.io

kubeadm

https://kubernetes.io/docs/reference/setup-tools/kubeadm/kubeadm

Getting Started with kubectl

Using kubectl



Getting Started with kubectl Commands

```
kubectl version

kubectl cluster-info

kubectl get all

kubectl run [container-name]
  --image=[image-name]

kubectl port-forward [pod] [ports]

kubectl expose …

kubectl create [resource]

kubectl apply [resource]
```

- Check Kubernetes version
- View cluster information
- Retrieve information about Kubernetes Pods, Deployments, Services, and more
- Simple way to create a Deployment for a Pod
- Forward a port to allow external access
- Expose a port for a Deployment/Pod
- Create a resource
- Create or modify a resource

Aliasing kubectl (to save on typing)

```
# PowerShell

Set-Alias -Name k -Value kubectl
```

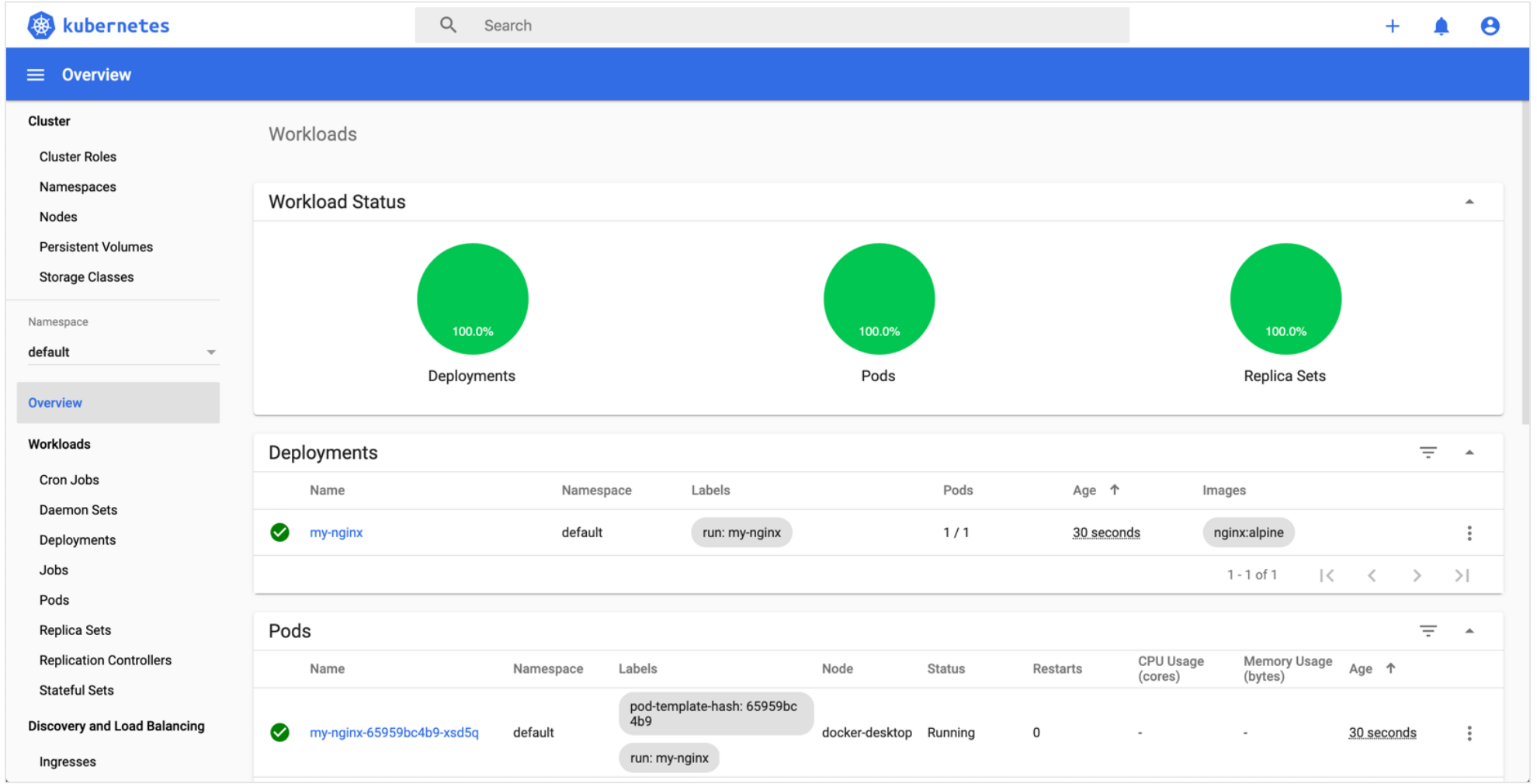
◀ Create alias for PowerShell

```
# Mac/Linux

alias k="kubectl"
```

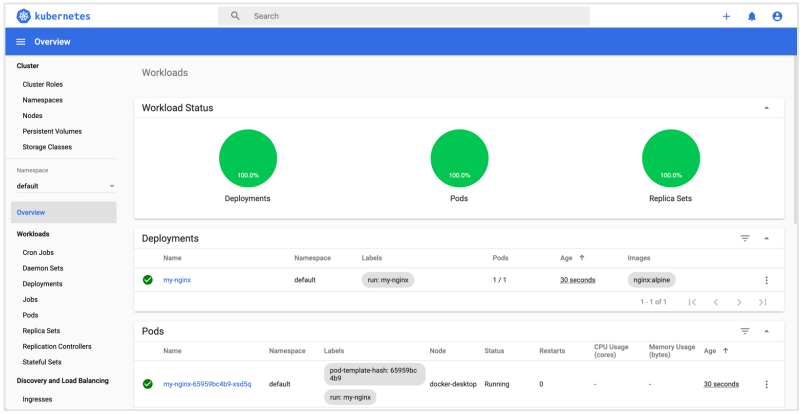
◀ Create alias for Mac/Linux shell

Web UI Dashboard

The Web UI Dashboard



Enabling the Web UI Dashboard



Web UI dashboard provides a user interface to view Kubernetes resources

Steps to enable the UI Dashboard:

- kubectl apply [dashboard-yaml-url]
- kubectl describe secret -n kube-system
- Locate the kubernetes.io/service-account-token and copy the token
- kubectl proxy
- Visit the dashboard URL and login using the token

Summary

Kubernetes provides container orchestration capabilities

Use for production, emulating production, testing, and more

Several options are available to run Kubernetes locally

Interact with Kubernetes using kubectl

Creating Pods

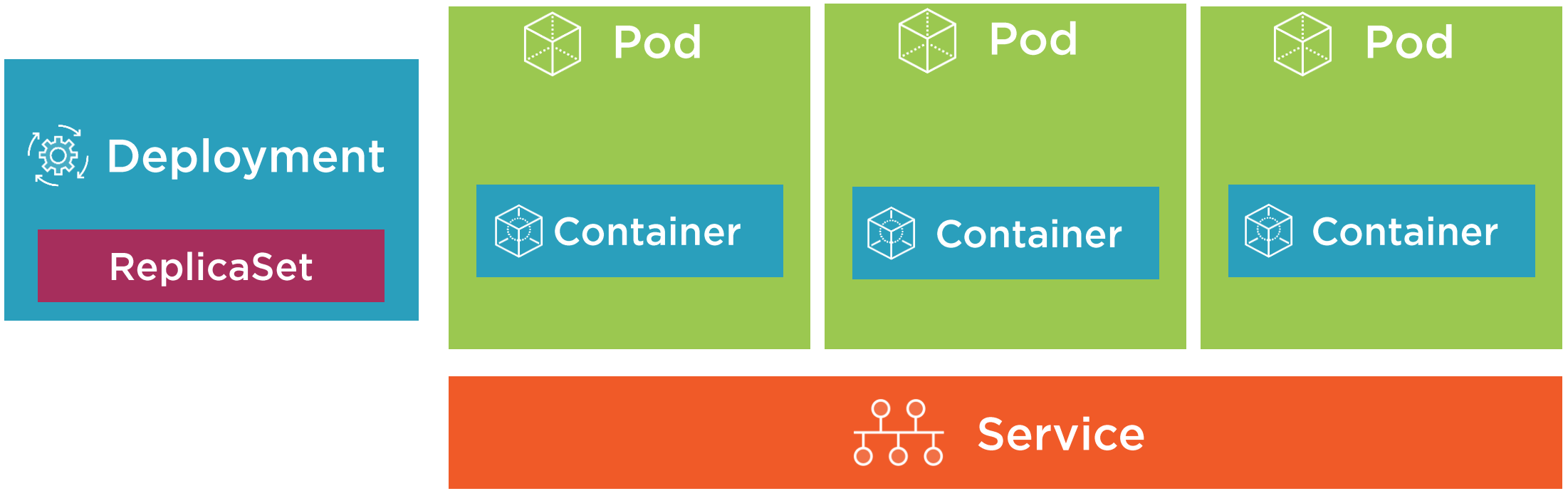
Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

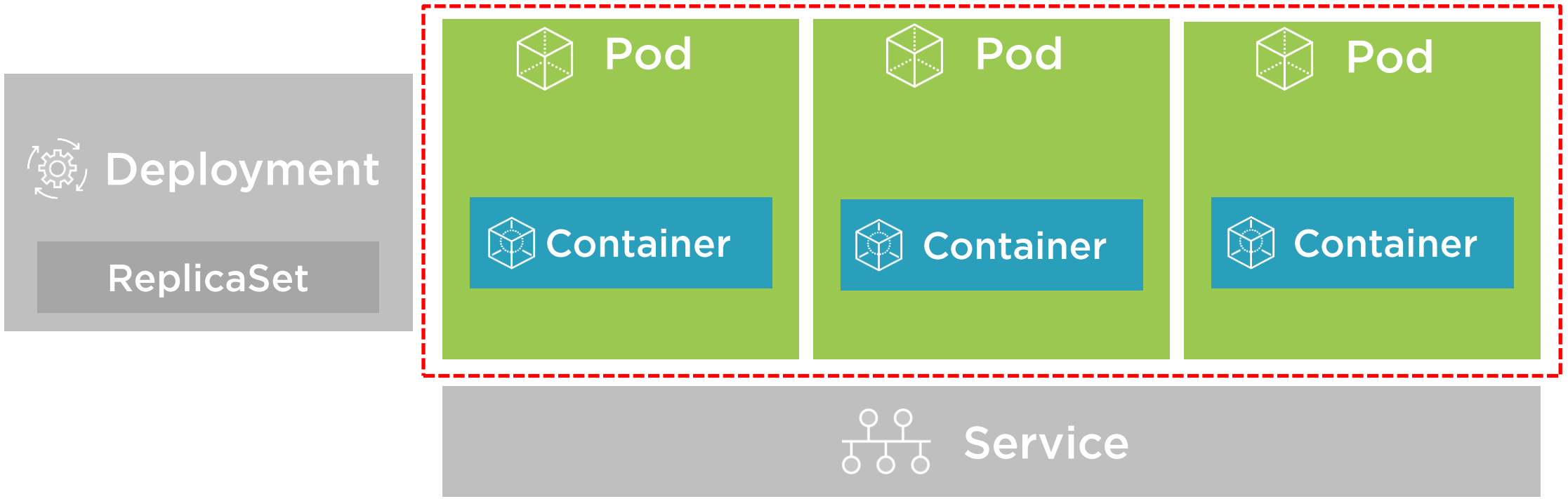
Module Overview

- Pod Core Concepts
- Creating a Pod
- kubectl and Pods
- YAML Fundamentals
- Defining a Pod with YAML
- Pod Health

You Are Here


Deployment
ReplicaSet
Pod
Container
Pod
Container
Pod
Container
Service

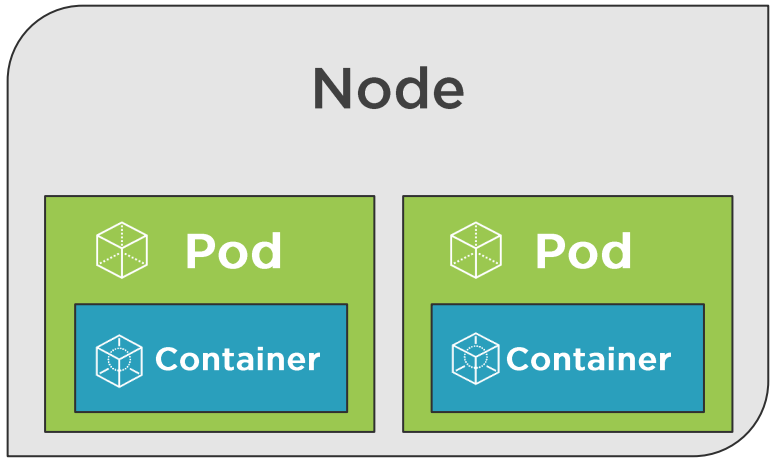
You Are Here


Deployment
ReplicaSet
Pod
Container
Pod
Container
Pod
Container
Service

Pod Core Concepts

A Pod is the basic execution unit of a Kubernetes application–the smallest and simplest unit in the Kubernetes object model that you create or deploy.

Kubernetes Pods



Smallest object of the Kubernetes object model

Environment for containers

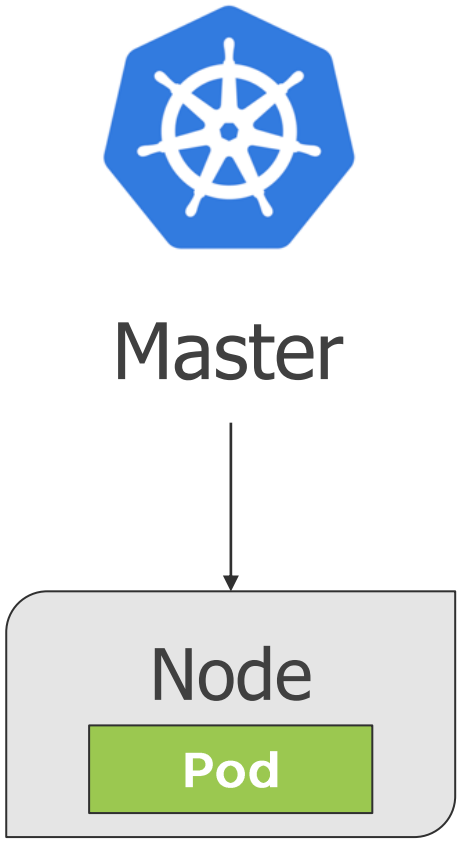
Organize application "parts" into Pods (server, caching, APIs, database, etc.)

Pod IP, memory, volumes, etc. shared across containers

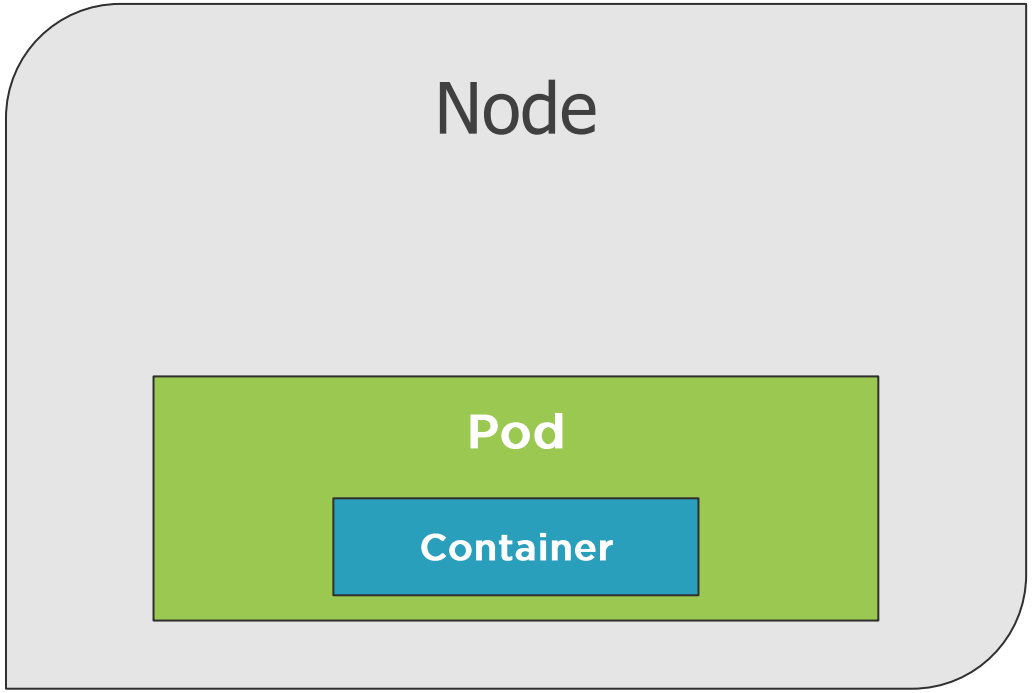
Scale horizontally by adding Pod replicas

Pods live and die but never come back to life

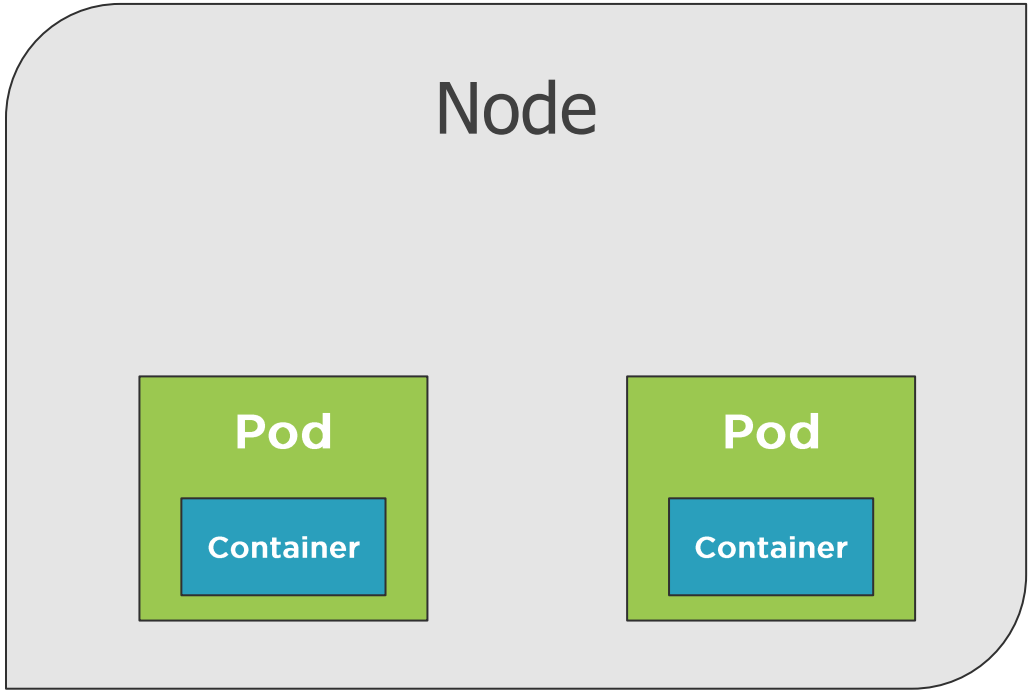
The Role of Pods



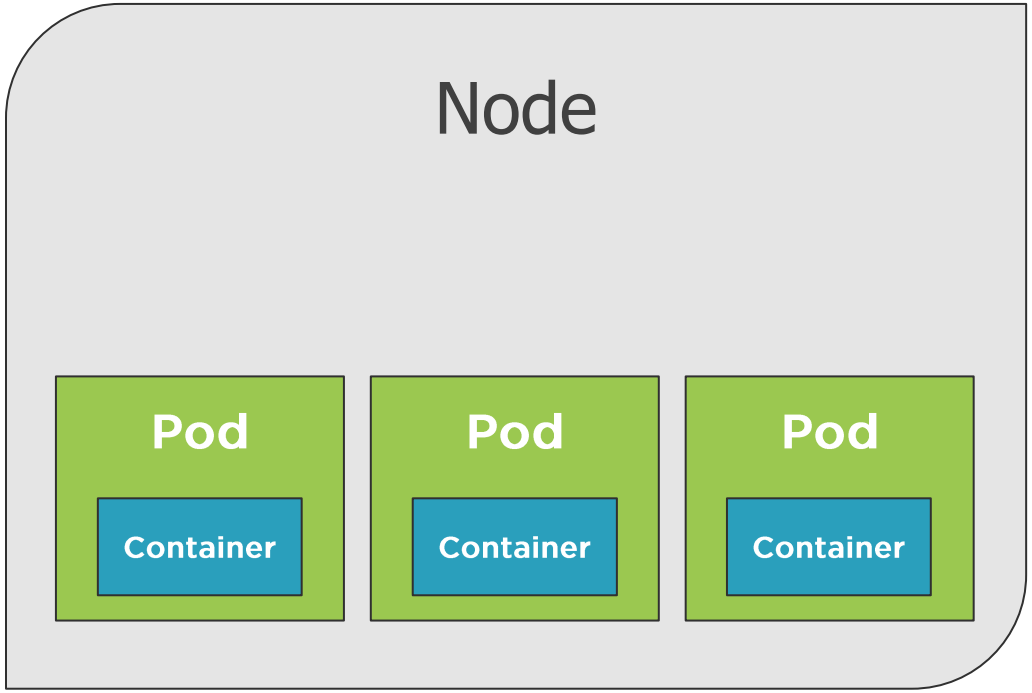
The Role of Pods



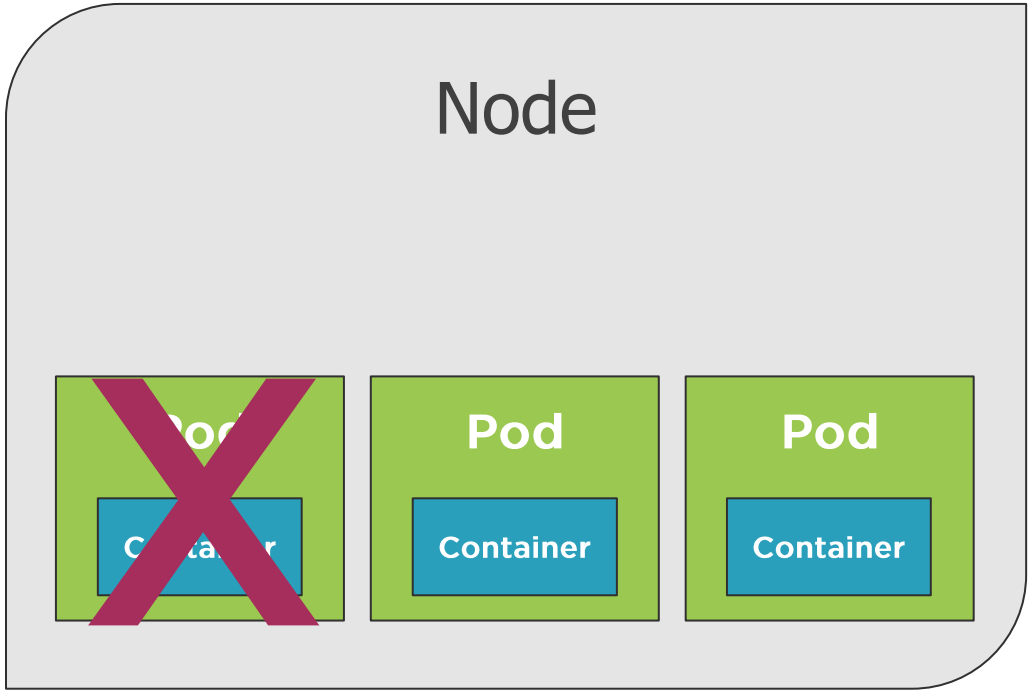
The Role of Pods



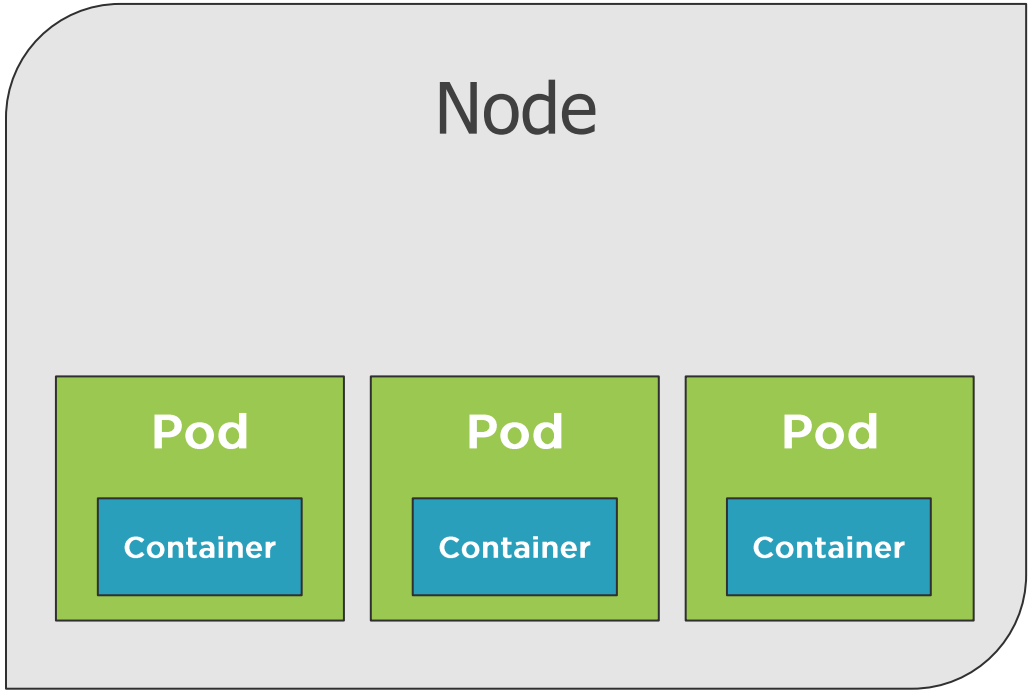
The Role of Pods



The Role of Pods



The Role of Pods



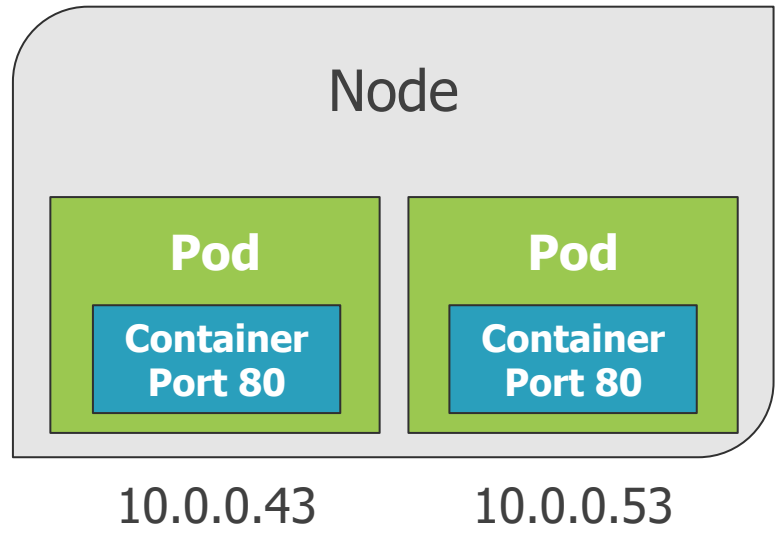
Pod containers share the same Network namespace (share IP/port)

Pod containers have the same loopback network interface (localhost)

Container processes need to bind to different ports within a Pod

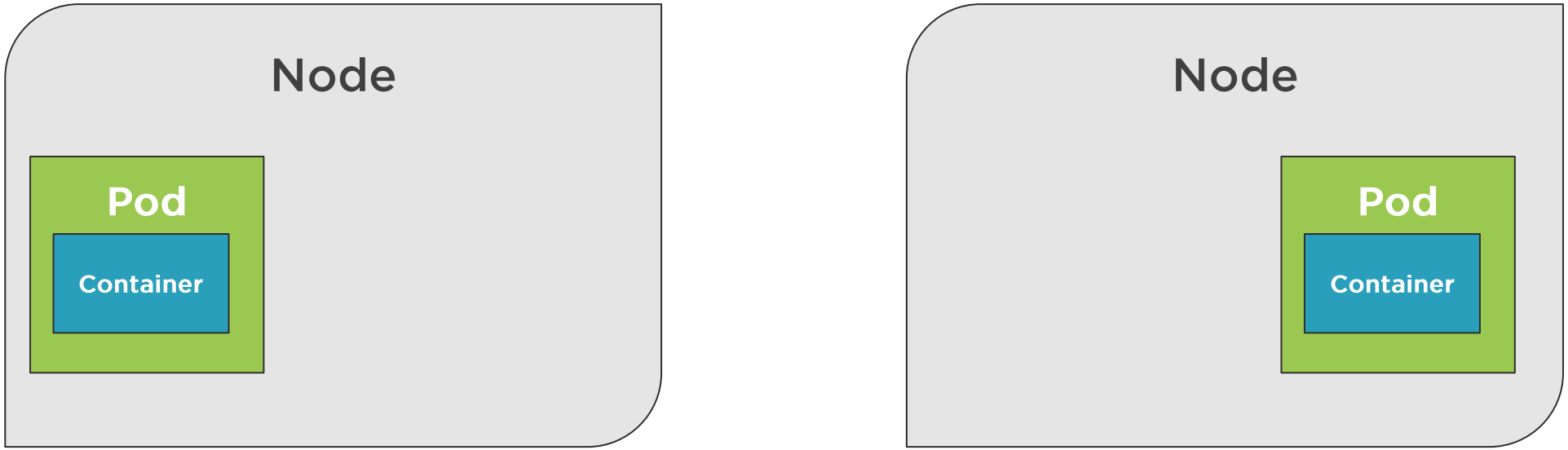
Ports can be reused by containers in separate Pods

Pods, IPs, and Ports



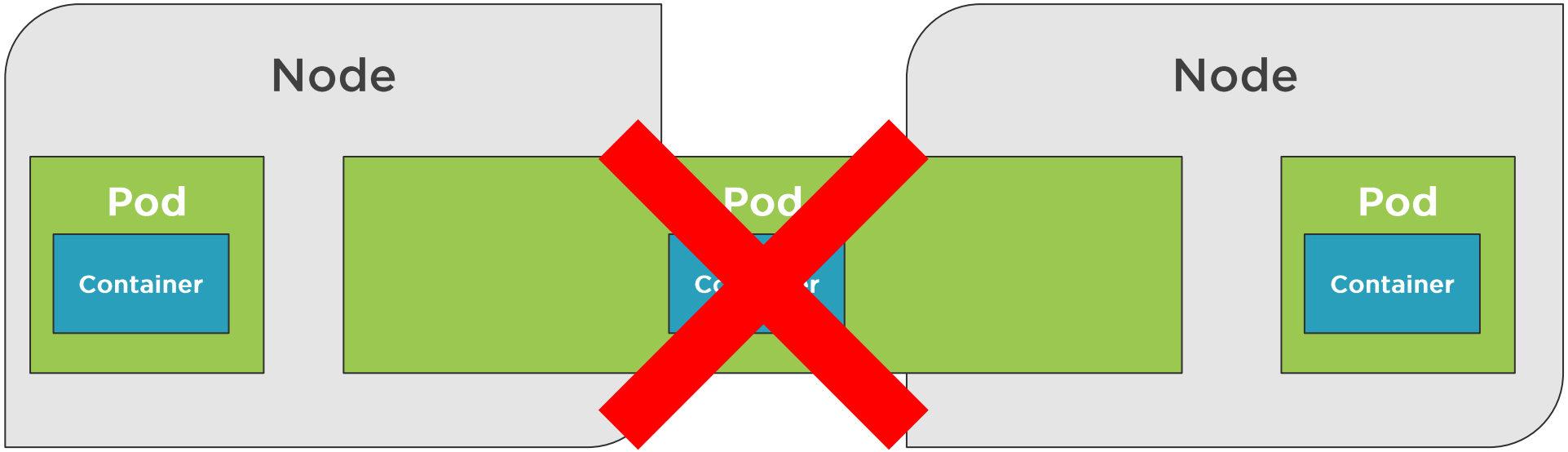
Nodes and Pods

Pods do not span nodes



Nodes and Pods

Pods do not span nodes



Creating a Pod

Running a Pod

There are several different ways to schedule a Pod:

- **kubectl run command**
- **kubectl create/apply command with a yaml file**

```
# Run the nginx:alpine container in a Pod
kubectl run [podname] --image=nginx:alpine
```

```
# List only Pods
kubectl get pods

# List all resources
kubectl get all
```

Get Information about a Pod

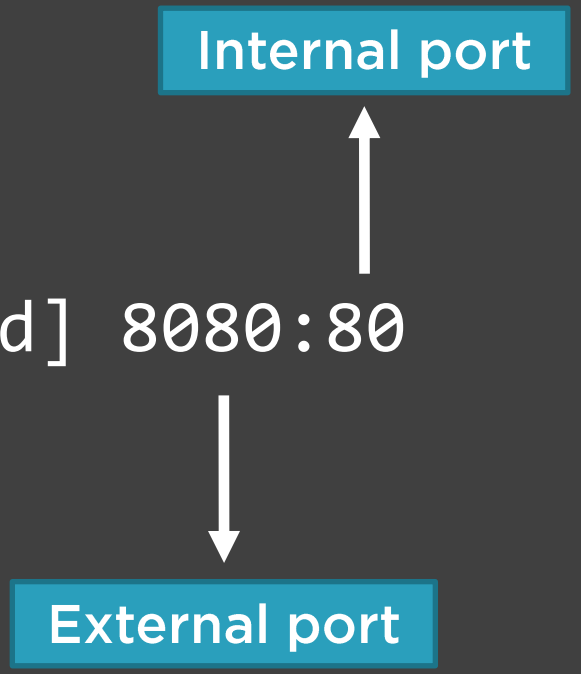
The kubectl get command can be used to retrieve information about Pods and many other Kubernetes objects

Expose a Pod Port

Pods and containers are only accessible within the Kubernetes cluster by default

One way to expose a container port externally: kubectl port-forward

```
# Enable Pod container to be
# called externally
kubectl port-forward [name-of-pod] 8080:80
```



```
# Will cause pod to be recreated
kubectl delete pod [name-of-pod]


# Delete Deployment that manages the Pod
kubectl delete deployment [name-of-deployment]
```

Deleting a Pod

Running a Pod will cause a Deployment to be created

To delete a Pod use kubectl delete pod or find the deployment and use kubectl delete deployment

kubectl and Pods

```
kubectl run [pod-name] --image=nginx:alpine

kubectl get pods

kubectl port-forward [pod-name] 8080:80

kubectl delete pod [pod-name]
```

Working with Pods using kubectl

Different kubectl commands can be used to run, view, and delete Pods

YAML Fundamentals

YAML Review

YAML files are composed of maps and lists

Indentation matters (be consistent!)

Always use spaces

Maps:

- name: value pairs
- Maps can contain other maps for more complex data structures

Lists:

- Sequence of items
- Multiple maps can be defined in a list

Introduction to YAML

```
key: value
complexMap:
  key1: value
  key2:
    subKey: value
items:
  - item1
  - item2
itemsMap:
  - map1: value
    map1Prop: value
  - map2: value
    map2Prop: value
```

- YAML maps define a key and value
- More complicated map structures can be defined using a key that references another map
- YAML lists can be used to define a sequence of items
- YAML lists can define a sequence maps

Note:
- Indentation matters
- Use spaces NOT tabs

Defining a Pod with YAML

Defining a Pod with YAML

Pod

+ kubectl =

nginx.pod.yml

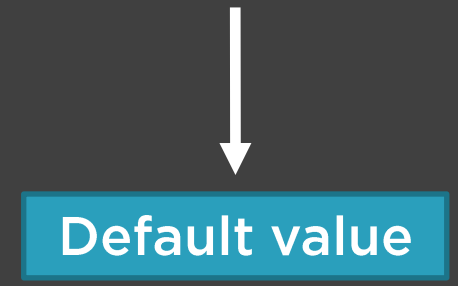
```
apiVersion: v1
kind: Pod
metadata:
  name: my-nginx
spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
```

- Kubernetes API version
- Type of Kubernetes resource
- Metadata about the Pod
- The spec/blueprint for the Pod
- Information about the containers that will run in the pod

Creating a Pod Using YAML

To create a pod using YAML use the kubectl create command along with the --filename or -f switch

```
# Perform a "trial" create and also validate the YAML
kubectl create -f file.pod.yml --dry-run --validate=true


# Create a Pod from YAML
# Will error if Pod already exists
kubectl create -f file.pod.yml
```



Creating or Applying Changes to a Pod

To create or apply changes to a pod using YAML use the kubectl apply command along with the --filename or -f switch

```
# Alternate way to create or apply changes to a
# Pod from YAML
kubectl apply -f file.pod.yml

# Use --save-config when you want to use
# kubectl apply in the future
kubectl create –f file.pod.yml --save-config
```

Using kubectl create --save-config

```
apiVersion: v1

kind: Pod

metadata:

  annotations:
    kubectl.kubernetes.io/
    last-applied-configuration:
   {"apiVersion":"v1","kind":"Pod",
    "metadata":{
      "name": "my-nginx"
     ...
    }

   }

...
```

- **--save-config** causes the resource's configuration settings to be saved in the annotations
- Example of saved configuration
- Having this allows in-place changes to be made to a Pod in the future using **kubectl apply**

In-place/non-disruptive changes can also be made to a Pod using **kubectl edit** or **kubectl patch.**

Deleting a Pod

To delete a Pod use kubectl delete

```
# Delete Pod
kubectl delete pod [name-of-pod]

# Delete Pod using YAML file that created it
kubectl delete -f file.pod.yml
```

kubectl and YAML

```
kubectl create –f nginx.pod.yml --save-config
kubectl describe pod [pod-name]
kubectl apply –f nginx.pod.yml
kubectl exec [pod-name] -it sh
kubectl edit -f nginx.pod.yml
kubectl delete –f nginx.pod.yml
```

Creating and Inspecting Pods with kubectl

Several different commands can be used to create and modify Pods

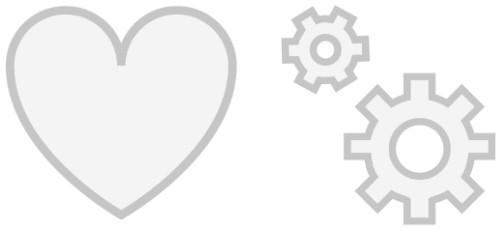
Pod Health

Kubernetes relies on Probes to determine the health of a Pod container.

A Probe is a diagnostic performed periodically by the kubelet on a Container.

Types of Probes

Types of Probes



Liveness probes determine if a Pod is healthy and running as expected

Readiness probes determine if a Pod should receive requests

Failed Pod containers are recreated by default (**restartPolicy** defaults to **Always**)

ExecAction – Executes an action inside the container

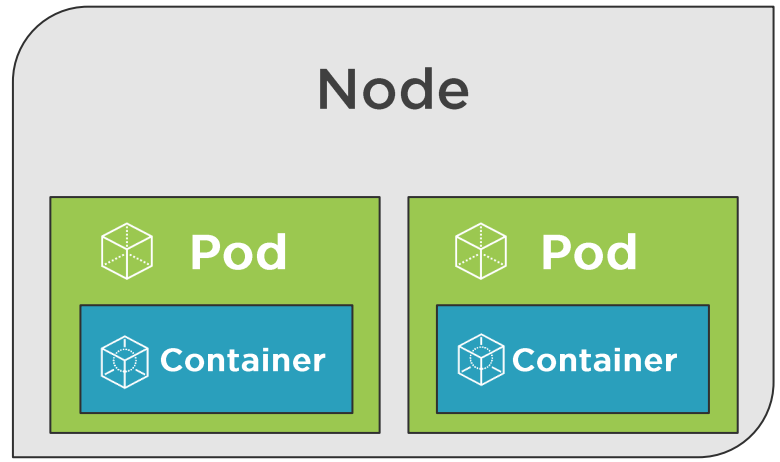
TCPSocketAction – TCP check against the container's IP address on a specified port

HTTPGetAction – HTTP GET request against container

Probes can have the following results:

- Success
- Failure
- Unknown

Probe Types



Defining an HTTP Liveness Probe

```
apiVersion: v1

kind: Pod

...

spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
    livenessProbe:
      httpGet:
        path: /index.html
        port: 80
      initialDelaySeconds: 15
      timeoutSeconds: 2
      periodSeconds: 5
      failureThreshold: 1
```

- Define liveness probe
- Check /index.html on port 80
- Wait 15 seconds
- Timeout after 2 seconds
- Check every 5 seconds
- Allow 1 failure before failing Pod

Defining an ExecAction Liveness Probe

```
apiVersion: v1
kind: Pod
...
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30;
      rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

- Define args for container
- Define liveness probe
- Define action/command to execute

Defining a Readiness Probe

```
apiVersion: v1

kind: Pod

...

spec:
  containers:
  - name: my-nginx
    image: nginx:alpine
    readinessProbe:
      httpGet:
        path: /index.html
        port: 80
      initialDelaySeconds: 2
      periodSeconds: 5
```

- Define readiness probe
- Check /index.html on port 80

- Wait 2 seconds
- Check every 5 seconds

Readiness Probe:

When should a container start receiving traffic?

Liveness Probe:

When should a container restart?

Pod Health in Action

Summary

- Pods are the smallest unit of Kubernetes
- Containers run within Pods and share a Pod's memory, IP, volumes, and more
- Pods can be started using different kubectl commands
- YAML can be used to create a Pod
- Health checks provide a way to notify Kubernetes when a Pod has a problem

Creating Deployments

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

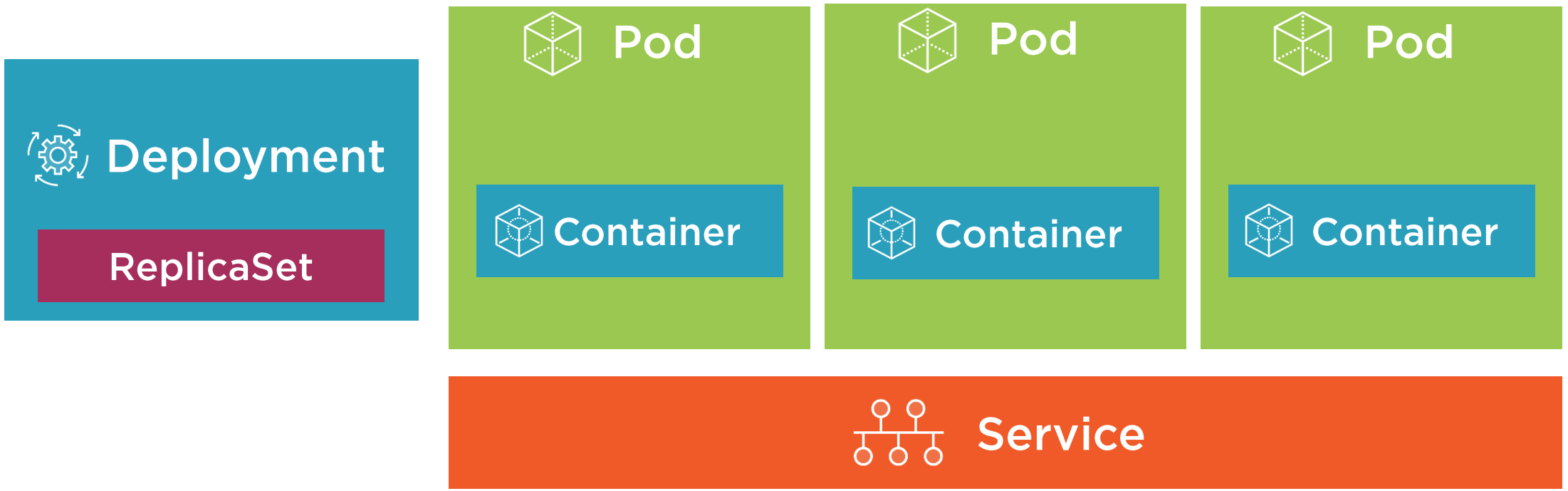
Deployments Core Concepts

Creating a Deployment

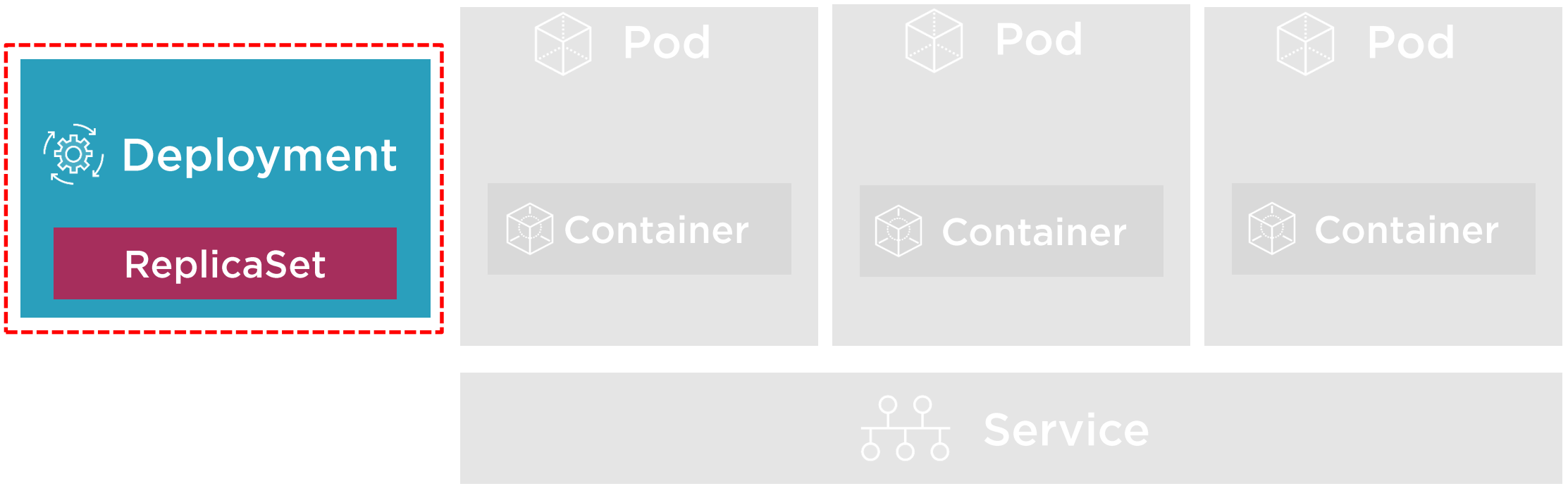
kubectl and Deployments

Deployment Options

You Are Here


Deployment
ReplicaSet
Pod
Container
Pod
Container
Pod
Container
Service

You Are Here


Deployment
ReplicaSet
Pod
Container
Pod
Container
Pod
Container
Service

Deployments Core Concepts

A ReplicaSet is a declarative way to manage Pods

A Deployment is a declarative way to manage Pods using a ReplicaSet

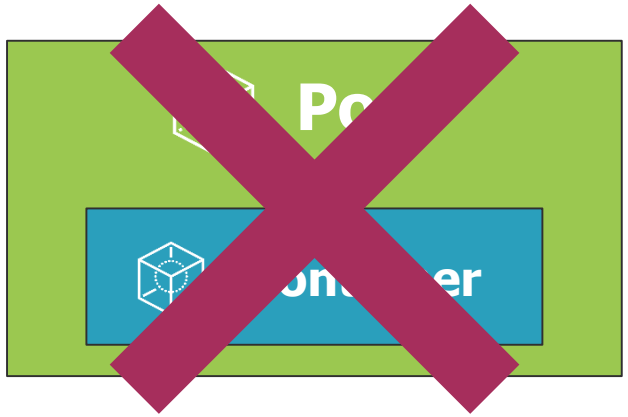
Pods, Deployments, and ReplicaSets

Pods represent the most basic resource in Kubernetes

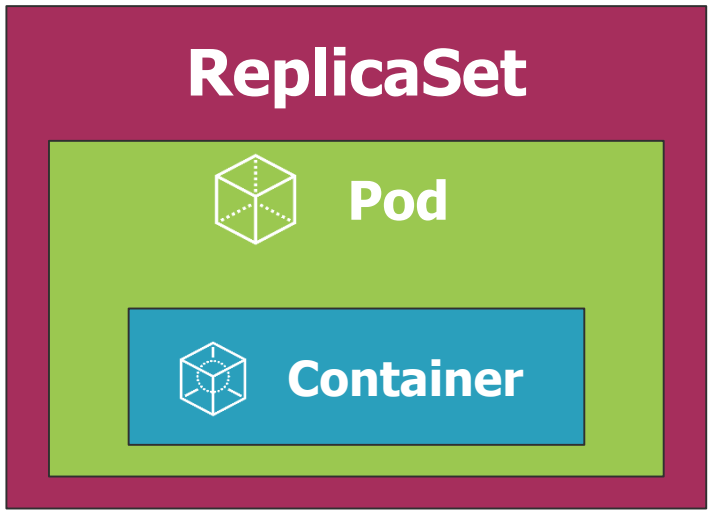
Can be created and destroyed but are never re-created

What happens if a Pod is destroyed?

Deployments and ReplicaSets ensure Pods stay running and can be used to scale Pods



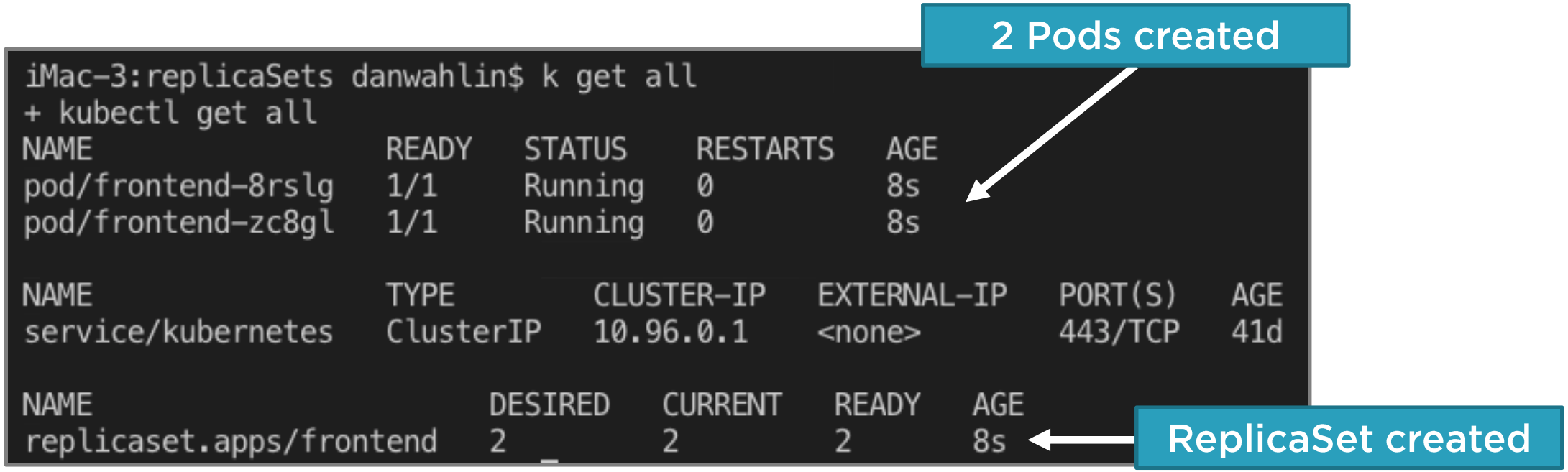
The Role of ReplicaSets



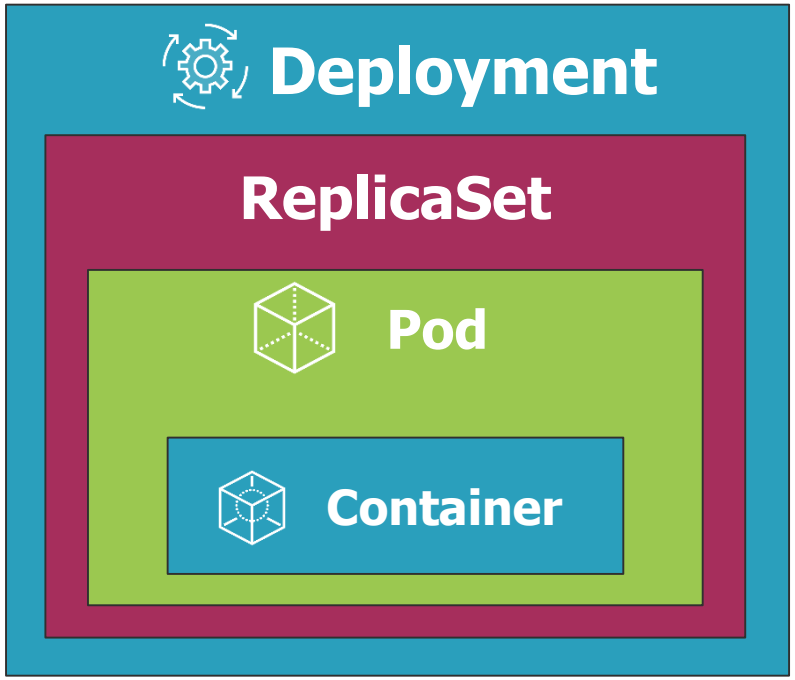
ReplicaSets act as a Pod controller:

- Self-healing mechanism
- Ensure the requested number of Pods are available
- Provide fault-tolerance
- Can be used to scale Pods
- Relies on a Pod template
- No need to create Pods directly!
- Used by Deployments

Result of Creating a ReplicaSet



The Role of Deployments



A Deployment manages Pods:

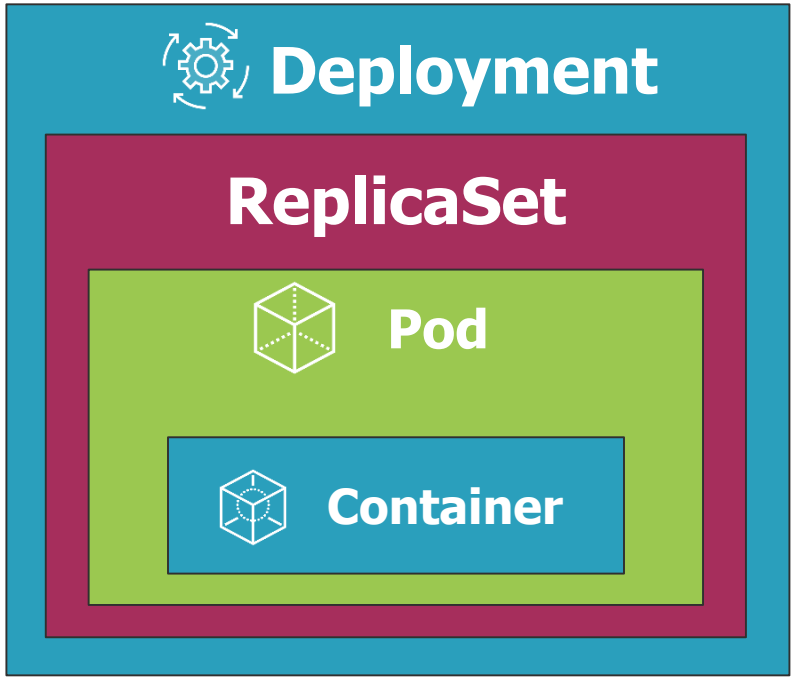
- Pods are managed using ReplicaSets
- Scales ReplicaSets, which scale Pods
- Supports zero-downtime updates by creating and destroying ReplicaSets
- Provides rollback functionality
- Creates a unique label that is assigned to the ReplicaSet and generated Pods
- YAML is very similar to a ReplicaSet

Creating a Deployment

Defining a Deployment with YAML

Deployment

+ kubectl =



Defining a Deployment (From a High-Level)

```
apiVersion: apps/v1
kind: Deployment
metadata:
spec:
  selector:
  template:
    spec:
      containers:
      - name: my-nginx
        image: nginx:alpine
```

- Kubernetes API version and resource type (Deployment)
- Metadata about the Deployment
- Select Pod template label(s)
- Template used to create the Pods
- Containers that will run in the Pod

Defining a Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: my-nginx
    tier: frontend
spec:
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: my-nginx
        image: nginx:alpine
```

- Kubernetes API version and resource type (Deployment)
- Metadata about the Deployment
- The selector is used to "select" the template to use (based on labels)
- Template to use to create the Pod/Containers (note that the selector matches the label)

Defining Probes in a Deployment

```
apiVersion: apps/v1

kind: Deployment

...

template:
  spec:
    containers:
    - name: my-nginx
      image: nginx:alpine
      livenessProbe:
        httpGet:
          path: /index.html
          port: 80
        initialDelaySeconds: 15
        timeoutSeconds: 2
        periodSeconds: 5
        failureThreshold: 1
```

- Define liveness probe (readiness probes can also be defined)
- Check /index.html on port 80

kubectl and Deployments

```
# Create a Deployment
kubectl create -f file.deployment.yml
```

Creating a Deployment

Use the kubectl create command along with the --filename or -f switch

Creating or Applying Changes

Use the kubectl apply command along with the --filename or -f switch

```
# Alternate way to create or apply changes to a
# Deployment from YAML
kubectl apply -f file.deployment.yml

# Use --save-config when you want to use
# kubectl apply in the future
kubectl create –f file.deployment.yml --save-config
```

```
kubectl get deployments
```

Getting Deployments

List all Deployments

```
# List all Deployments and their labels
kubectl get deployment --show-labels

# Get all Deployments with a specific label
kubectl get deployment -l app=nginx
```

Deployments and Labels

List the labels for all Deployments using the --show-labels switch

To get information about a Deployment with a specific label, use the -l switch

Deleting a Deployment

To delete a Deployment use kubectl delete

Will delete the Deployment and all associated Pods/Containers

```
# Delete Deployment
kubectl delete deployment [deployment-name]
```

```
# Scale the Deployment Pods to 5
kubectl scale deployment [deployment-name] --replicas=5

# Scale by refencing the YAML file
kubectl scale –f file.deployment.yml --replicas=5
```

Scaling Pods Horizontally

Update the YAML file or use the kubectl scale command

```
spec:
  replicas: 3
  selector:
    tier: frontend
```

kubectl Deployments in Action

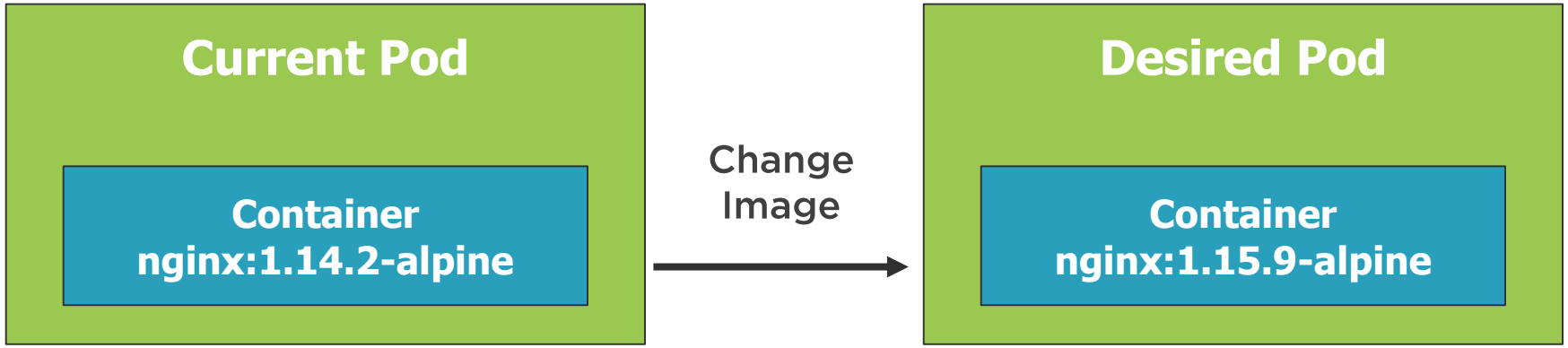
```
kubectl create –f nginx.deployment.yml --save-config

kubectl describe [pod | deployment] [pod-name | deployment-name]

kubectl apply –f nginx.pod.yml

kubectl get deployments --show-labels

kubectl get deployments –l app=my-nginx

kubectl scale -f nginx.deployment.yml --replicas=4
```

kubectl Deployments Commands

Several different kubectl commands can be used to create and work with Deployments

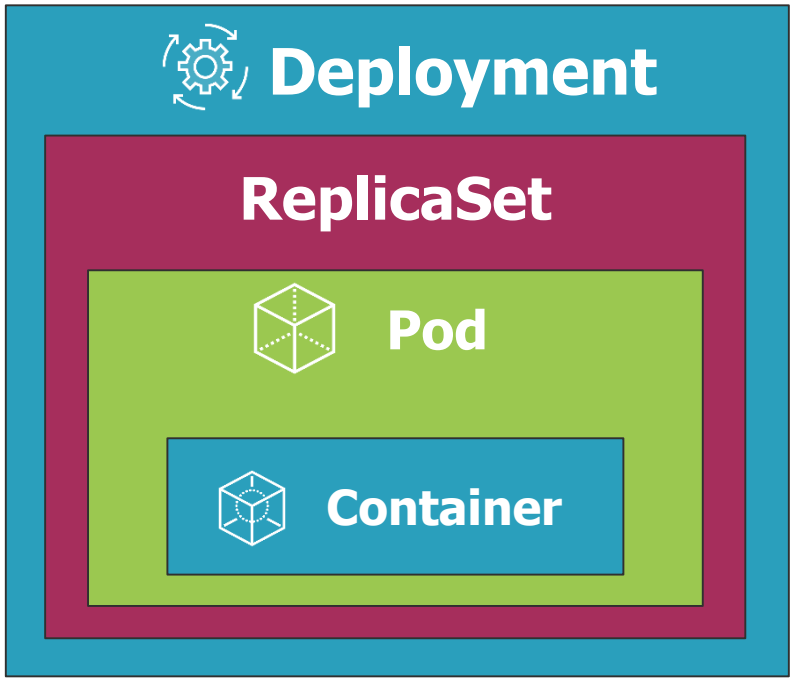
Deployment Options

How Do You Update Existing Pods?



Zero downtime deployments allow software updates to be deployed to production without impacting end users

Deployment Options



One of the strengths of Kubernetes is zero downtime deployments

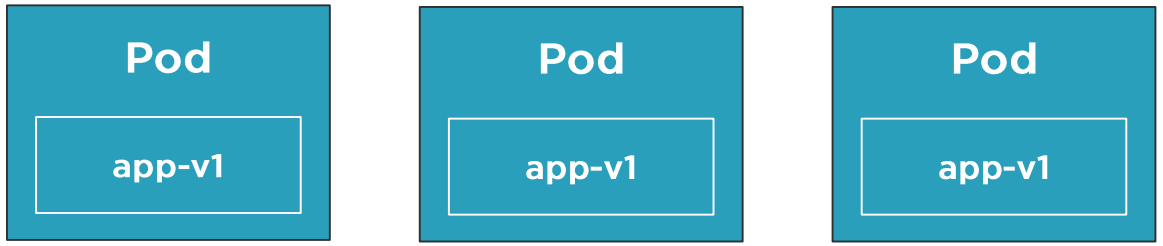
Update an application's Pods without impacting end users

Several options are available:

- Rolling updates
- Blue-green deployments
- Canary deployments
- Rollbacks

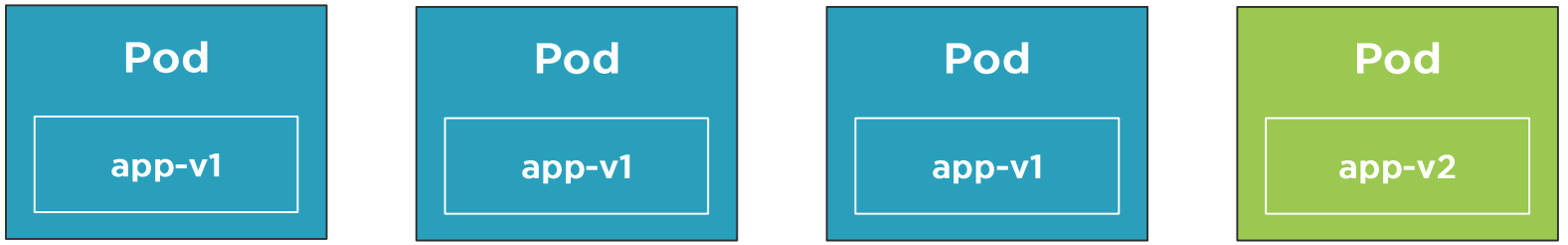
Rolling Deployments

Initial Pod State



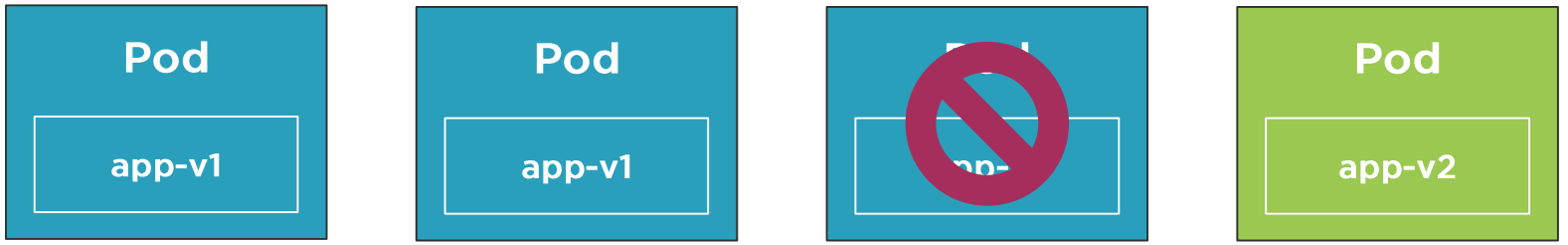
Rolling Deployments

Rollout New Pod



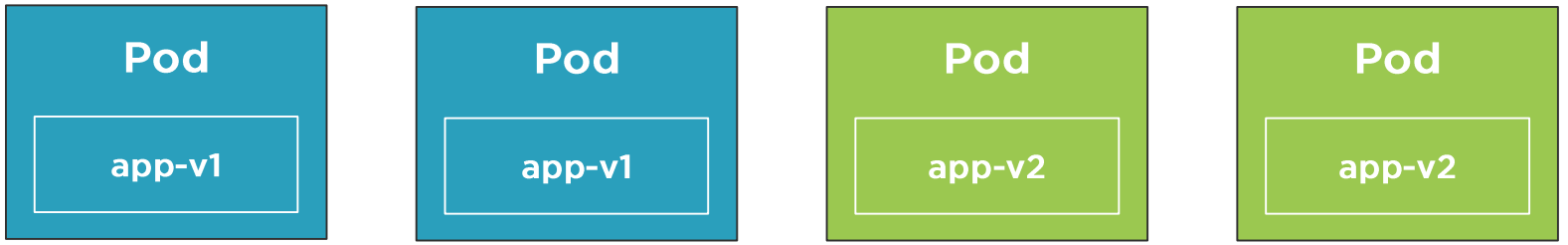
Rolling Deployments

Delete Pod

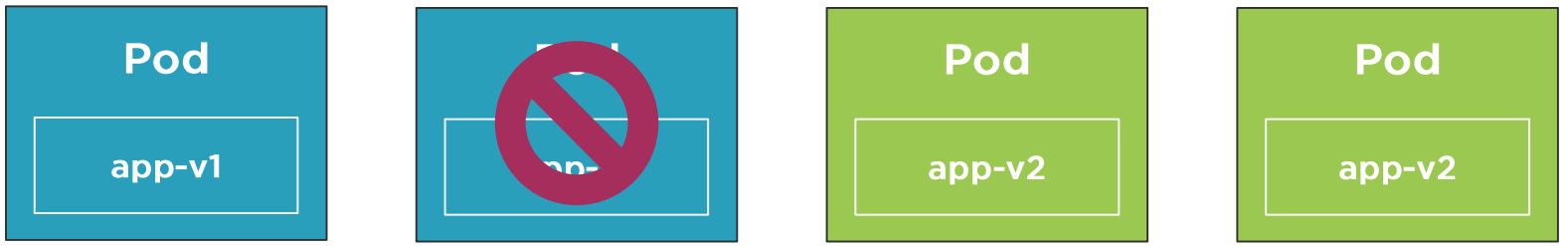


Rolling Deployments

Rollout New Pod

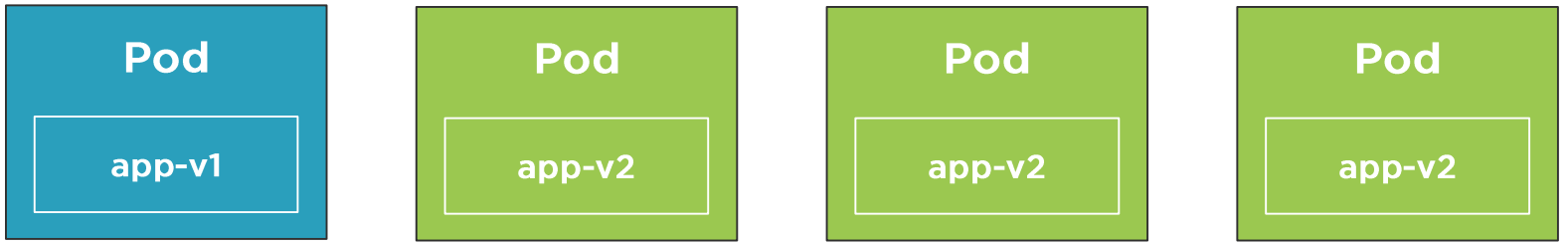


Rolling Deployments

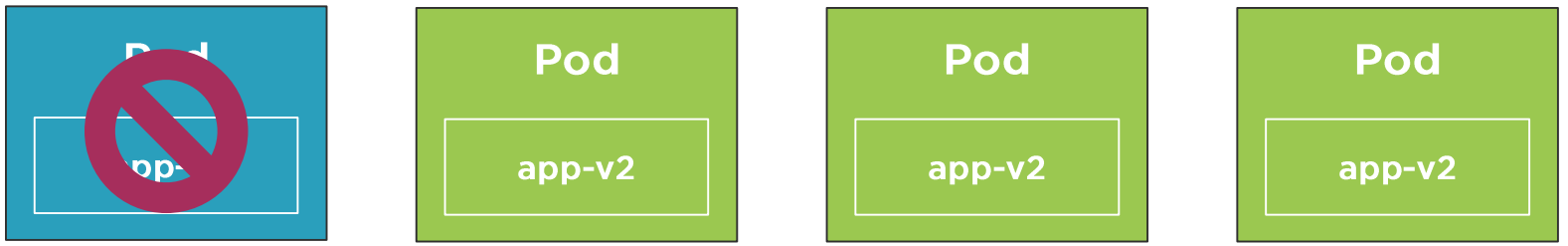


Rolling Deployments

Rollout New Pod

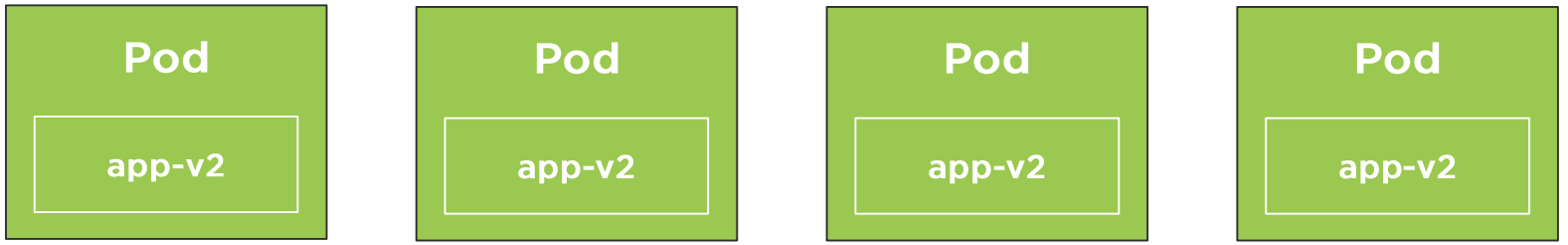


Rolling Deployments



Rolling Deployments

Rollout New Pod



Updating a Deployment

Update a deployment by changing the YAML and applying changes to the cluster with kubectl apply

```
# Apply changes made in a YAML file
kubectl apply -f file.deployment.yml
```

Zero Downtime Deployments in Action

Summary

Pods are deployed, managed, and scaled using deployments and ReplicaSets

Deployments are a higher-level resource that define one or more Pod templates

The kubectl *create* or *apply* commands can be used to run a deployment

Kubernetes supports zero downtime deployments

Creating Services

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

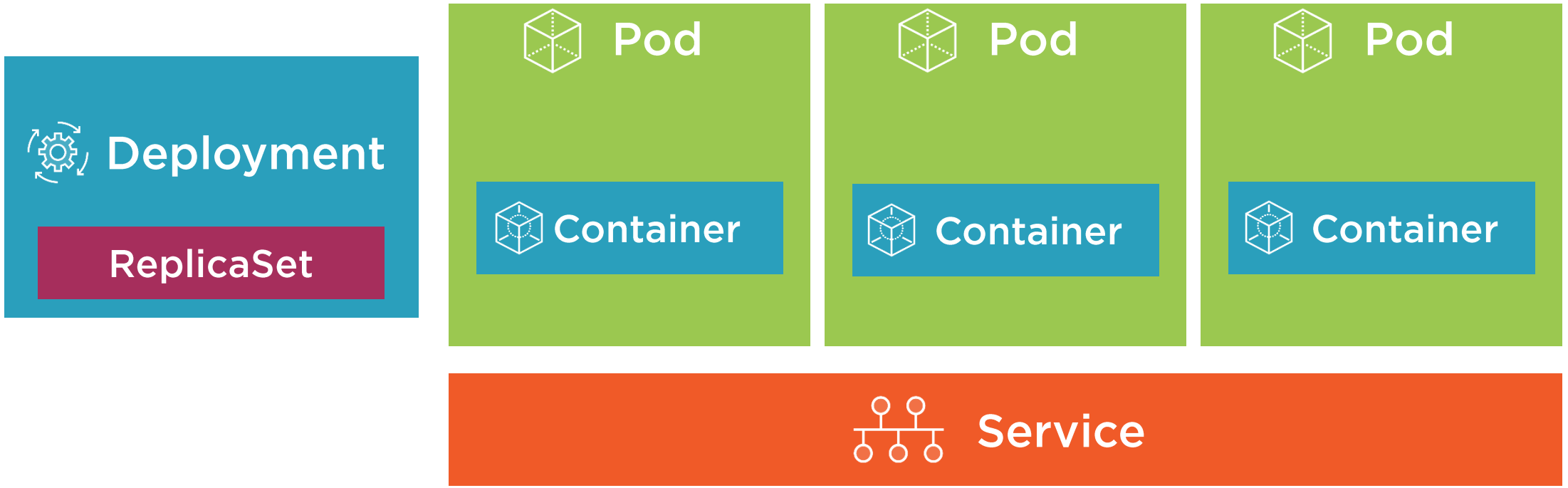
Services Core Concepts

Service Types

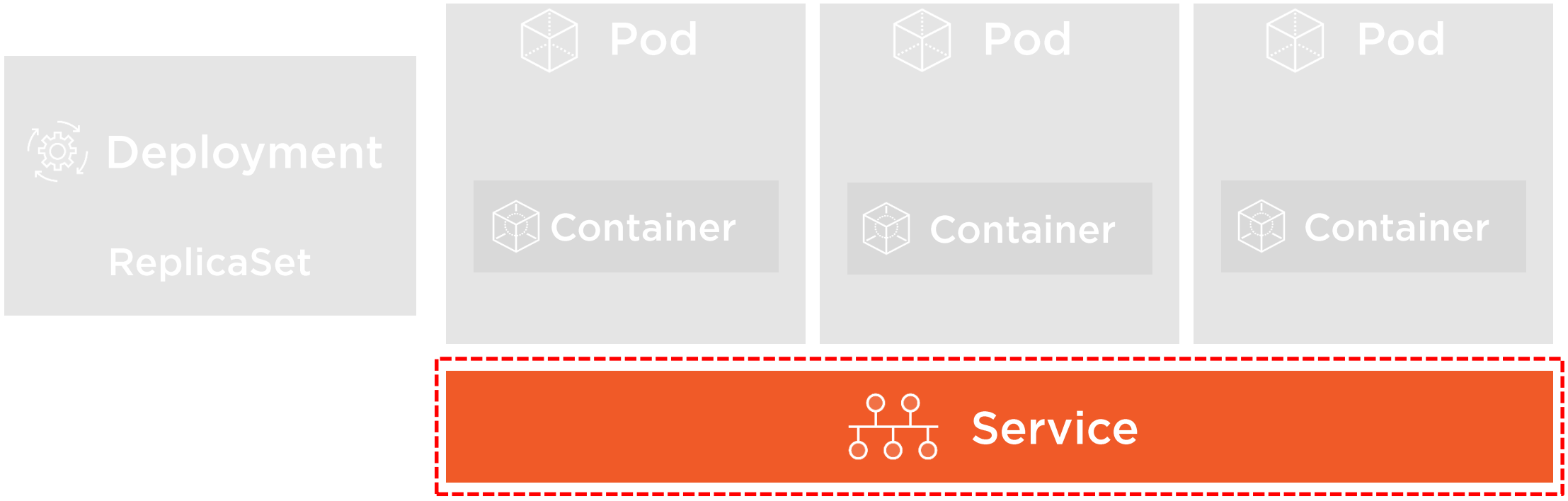
Creating a Service with kubectl

Creating a Service with YAML

You Are Here


Deployment
ReplicaSet
Pod
Container
Pod
Container
Pod
Container
Service

You Are Here

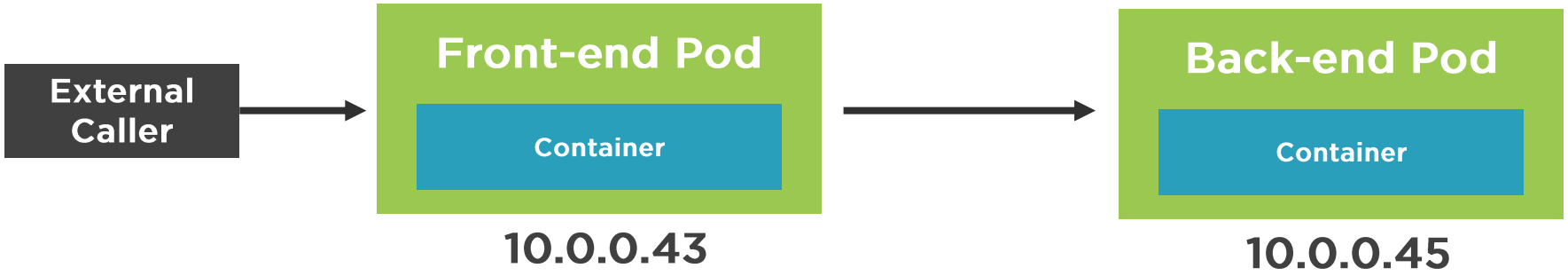

Deployment
ReplicaSet
Pod
Container
Pod
Container
Pod
Container
Service

Services Core Concepts

A Service provides a single point of entry for accessing one or more Pods

Review Question:

Since Pods live and die, can you rely on their IP?



Answer:

No! That's why we need Services - IPs change a lot!

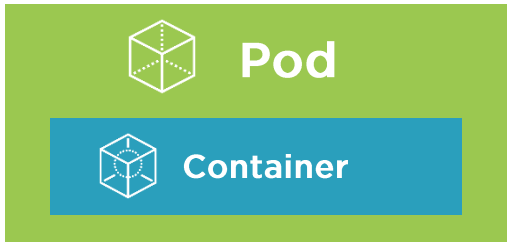
Pods are "mortal" and may only live a short time (ephemeral)

You can't rely on a Pod IP address staying the same

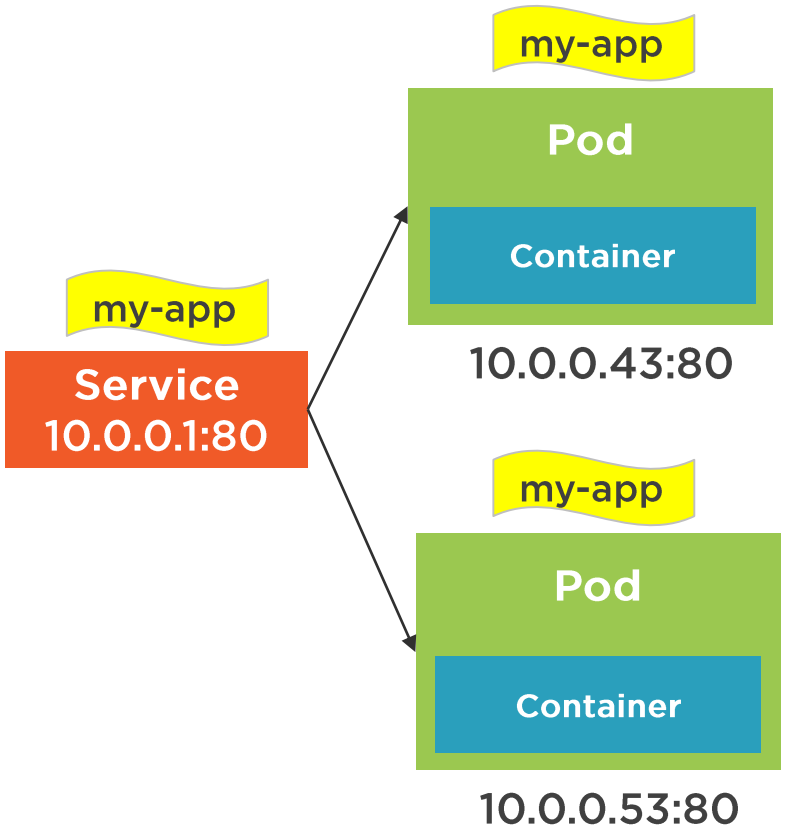
Pods can be horizontally scaled so each Pod gets its own IP address

A Pod gets an IP address after it has been scheduled (no way for clients to know IP ahead of time)

The Life of a Pod

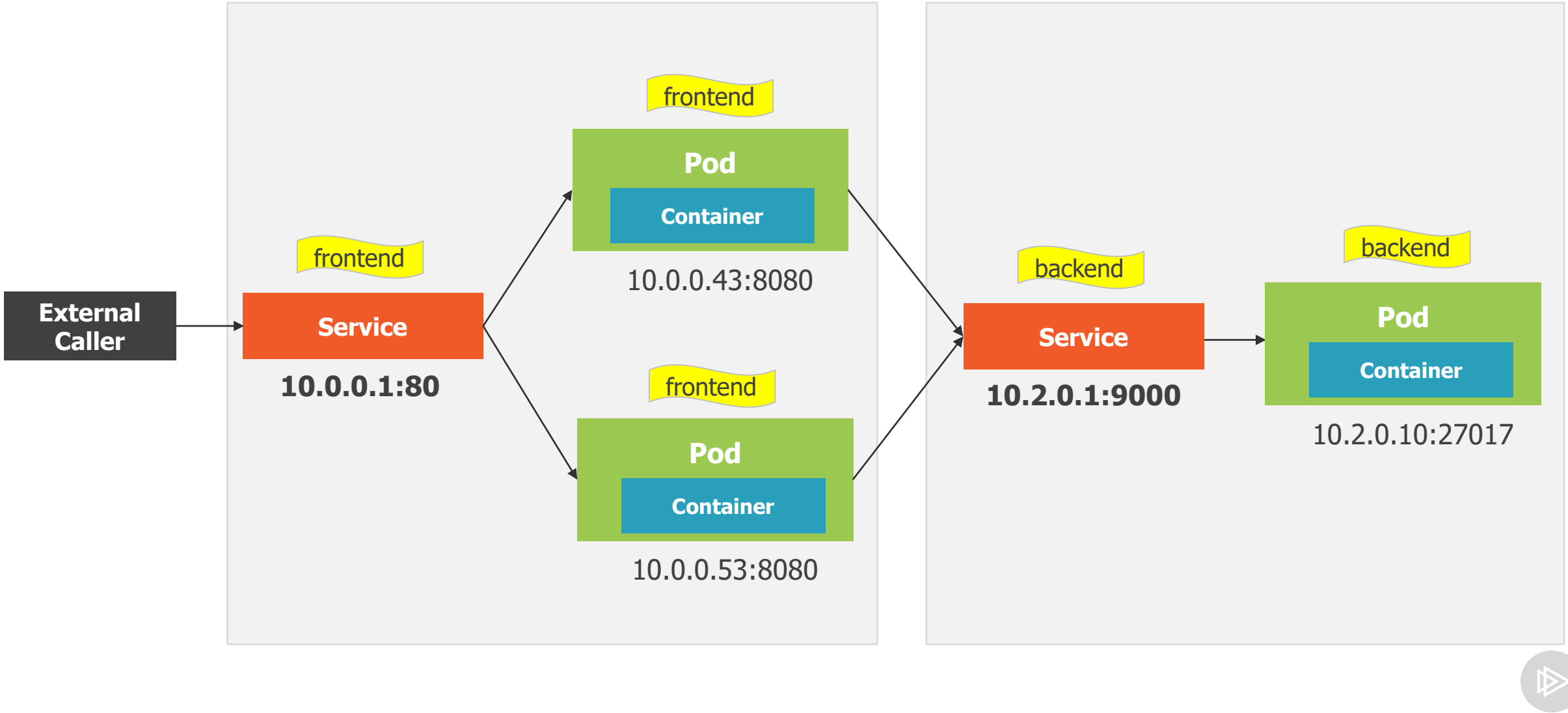


The Role of Services

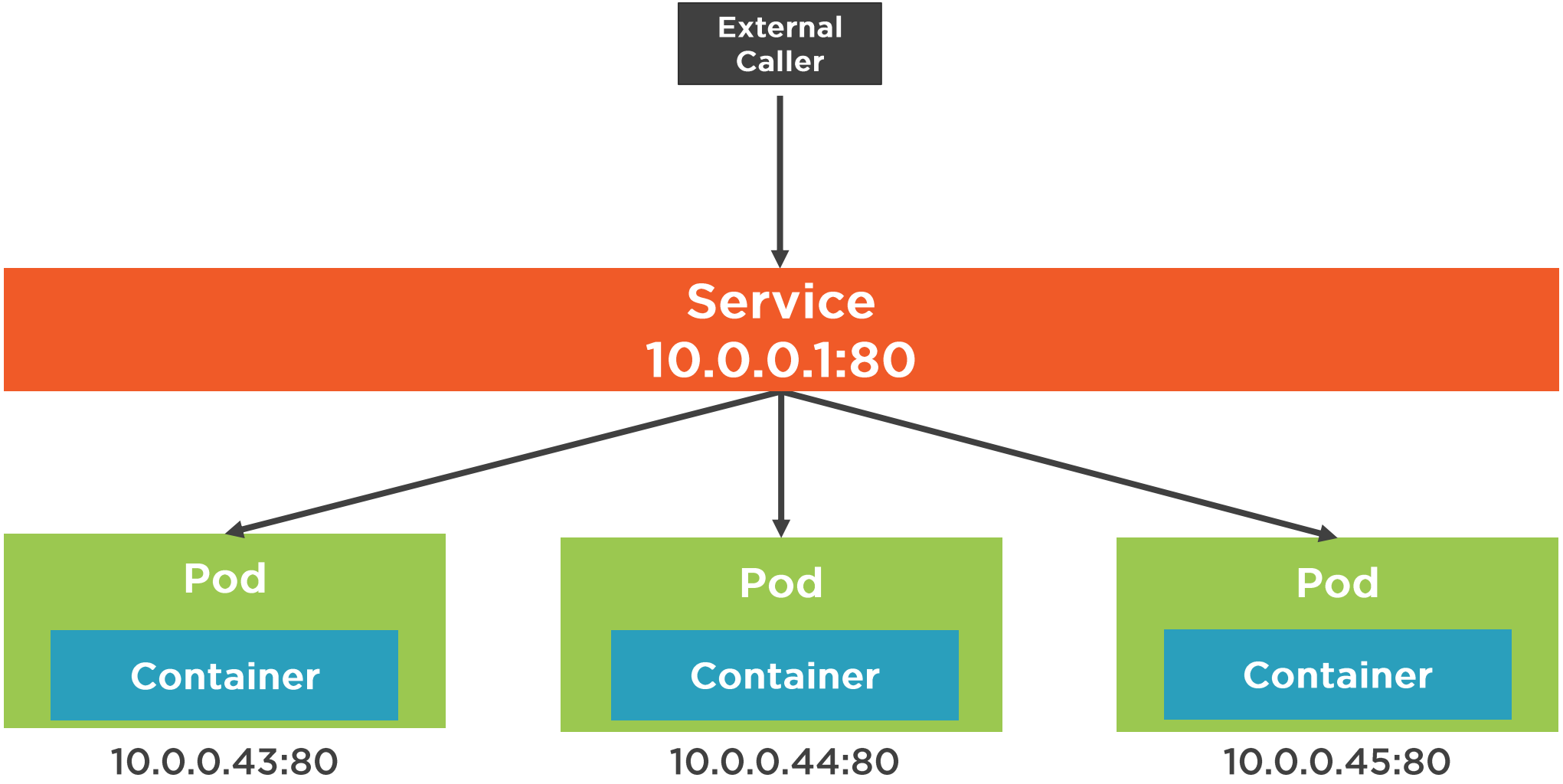


- Services abstract Pod IP addresses from consumers
- Load balances between Pods
- Relies on labels to associate a Service with a Pod
- Node's kube-proxy creates a virtual IP for Services
- Layer 4 (TCP/UDP over IP)
- Services are not ephemeral
- Creates endpoints which sit between a Service and Pod

Calling Services



Services and Pod Load Balancing



Service Types

Service Types

Services can be defined in different ways:

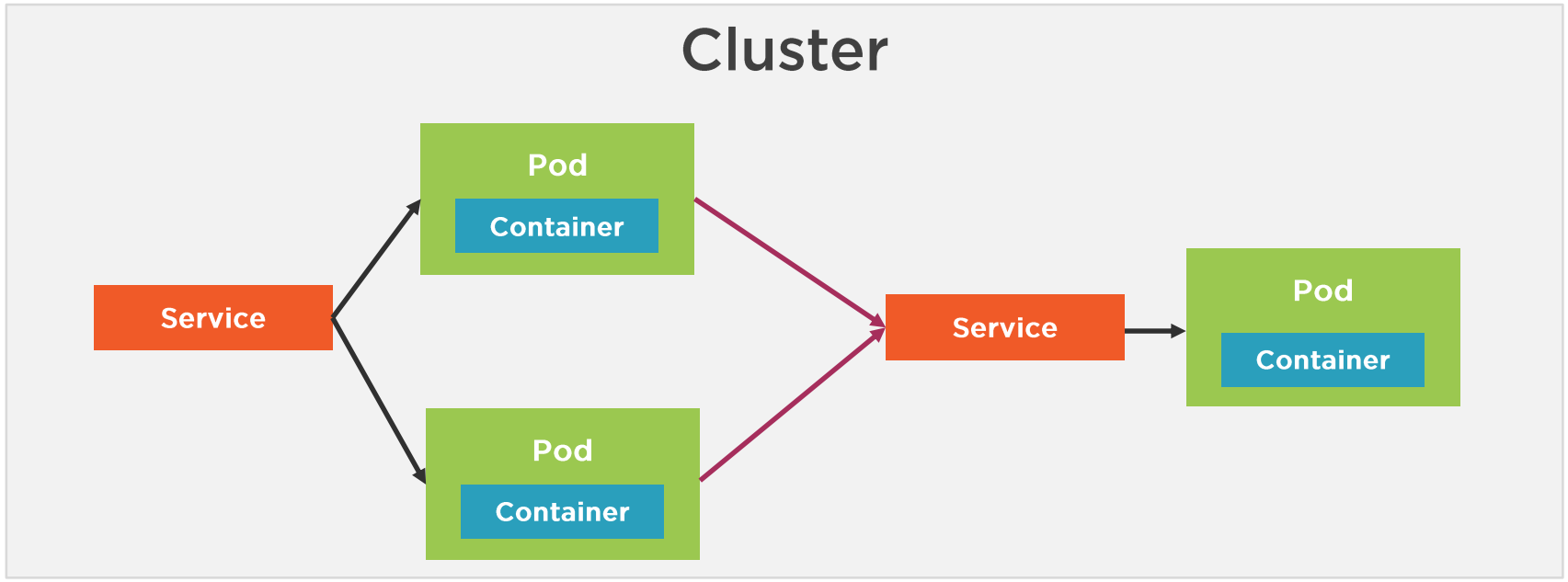
- ClusterIP – Expose the service on a cluster-internal IP (default)
- NodePort – Expose the service on each Node's IP at a static port.
- LoadBalancer – Provision an external IP to act as a load balancer for the service
- ExternalName – Maps a service to a DNS name

ClusterIP Service

Service IP is exposed internally within the cluster

Only Pods within the cluster can talk to the Service

Allows Pods to talk to other Pods

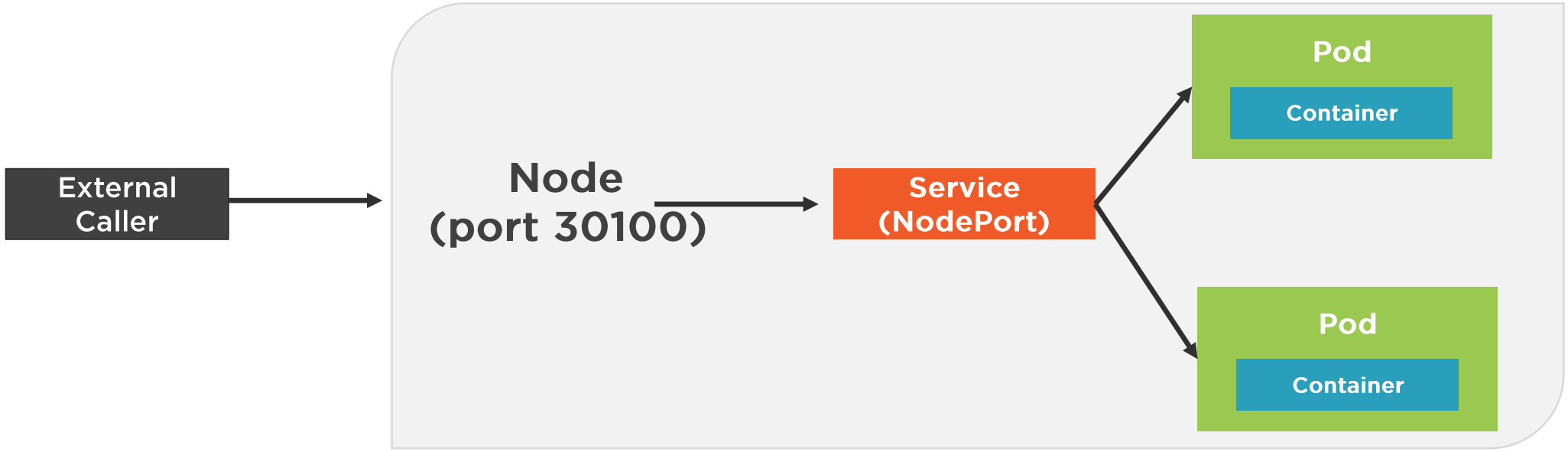


NodePort Service

Exposes the Service on each Node's IP at a static port

Allocates a port from a range (default is 30000-32767)

Each Node proxies the allocated port



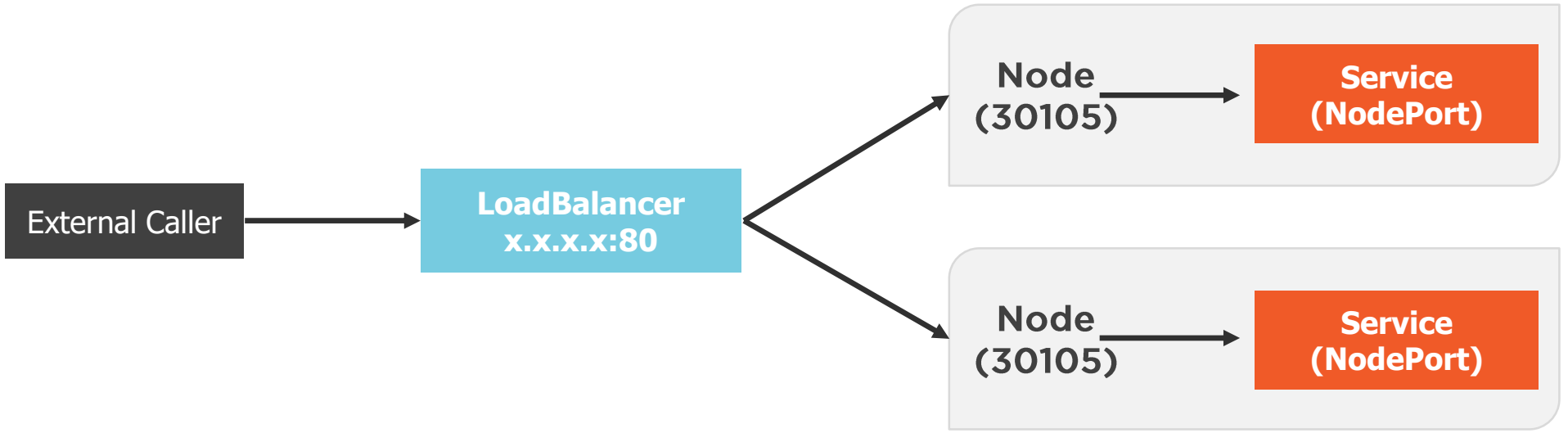
LoadBalancer Service

Exposes a Service externally

Useful when combined with a cloud provider's load balancer

NodePort and ClusterIP Services are created

Each Node proxies the allocated port

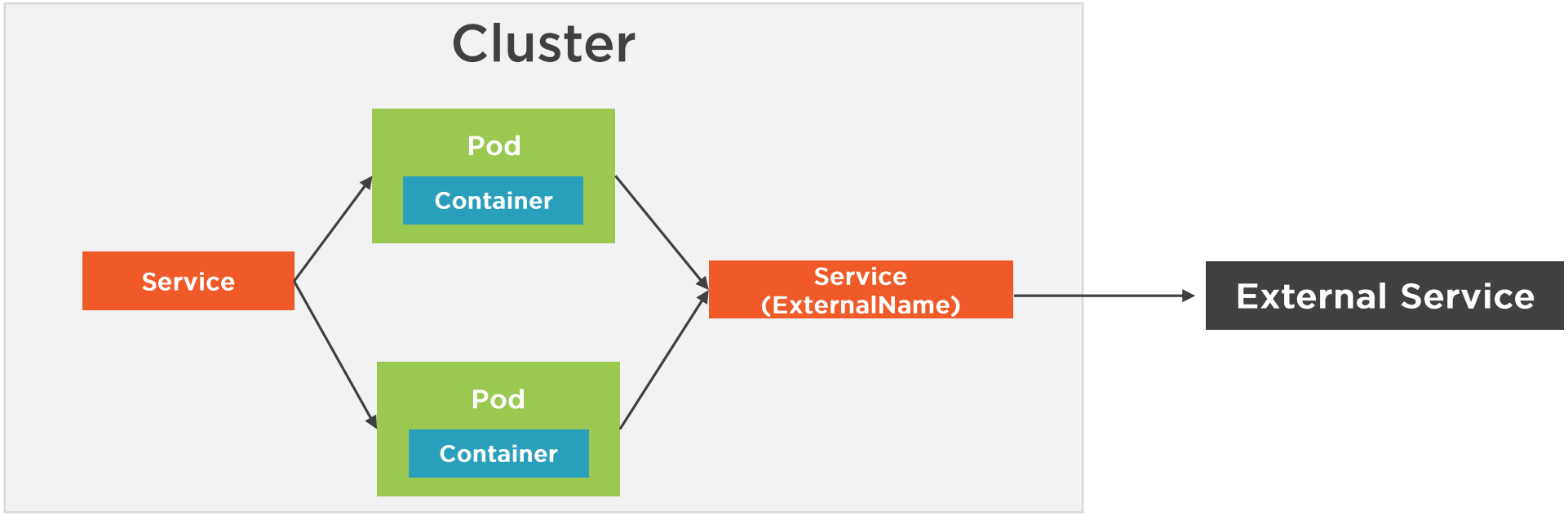


ExternalName Service

Service that acts as an alias for an external service

Allows a Service to act as the proxy for an external service

External service details are hidden from cluster (easier to change)



Creating a Service with kubectl

Port Forwarding

Q. How can you access a Pod from outside of Kubernetes?

A. Port forwarding

Use the kubectl port-forward to forward a local port to a Pod port

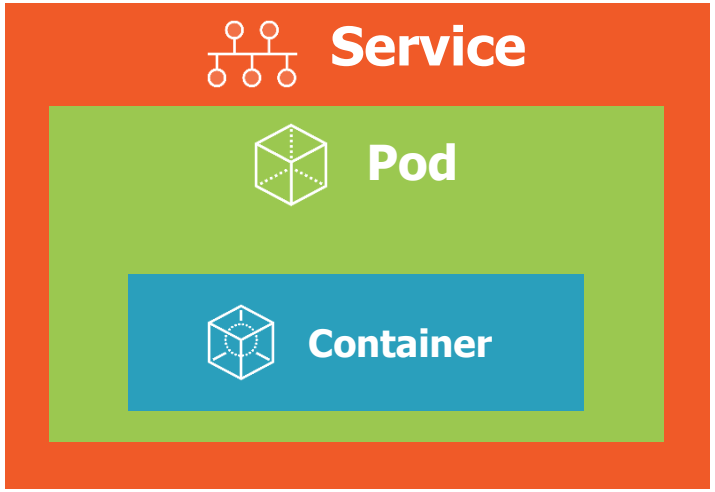
```
# Listen on port 8080 locally and forward to port 80 in Pod
kubectl port-forward pod/[pod-name] 8080:80

# Listen on port 8080 locally and forward to Deployment's Pod
kubectl port-forward deployment/[deployment-name] 8080

# Listen on port 8080 locally and forward to Service's Pod
kubectl port-forward service/[service-name] 8080
```

Creating a Service with YAML

Defining a Service with YAML

Service

+ kubectl =



Service Overview

```
apiVersion: v1
kind: Service
metadata:
spec:
  type:

  selector:

  ports:
```

- Kubernetes API version and resource type (Service)
- Metadata about the Service
- Type of service (ClusterIP, NodePort, LoadBalancer) – defaults to ClusterIP
- Select Pod template label(s) that service will apply to
- Define container target port and the port for the service

Defining a Service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  selector:
    app: nginx
  ports:
  - name: http
    port: 80
    targetPort: 80
```

- Kubernetes API version and resource type (Service)
- Metadata about the Service
- Service will apply to resources with a label of app: nginx
- Define container target port(s) and the port(s) for the Service

Connecting to a Service by It's DNS Name

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  ...

apiVersion: v1
kind: Service
metadata:
  name: backend
  ...
```

- Name of Service (each Service gets a DNS entry)

- A frontend Pod can access a backend Pod using **backend:port**

Creating a NodePort Service

```
apiVersion: v1
kind: Service
metadata:
  ...
spec:
  type: NodePort
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
    nodePort: 31000
```

- Set Service *type* to NodePort
- Optionally set NodePort value (defaults between 30000-32767)

Creating a LoadBalancer Service

```
apiVersion: v1
kind: Service
metadata:
  ...
spec:
  type: LoadBalancer
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
```

- Set Service *type* to LoadBalancer (normally used with cloud providers)

Creating an ExternalName Service

```
apiVersion: v1
kind: Service
metadata:
  name: external-service
spec:
  type: ExternalName
  externalName: api.acmecorp.com
  ports:
  - port: 9000
```

- Other Pods can use this FQDN to access the external service
- Set type to ExternalName
- Service will proxy to FQDN

kubectl and Services

Creating a Service

Use the kubectl create command along with the --filename or -f switch

```
# Create a Service
kubectl create -f file.service.yml
```

```
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes        ClusterIP   10.96.0.1      <none>        443/TCP    55d
nginx-clusterip   ClusterIP   10.102.26.70   <none>        8080/TCP   6s
```

```
# Update a Service
# Assumes --save-config was used with create
kubectl apply -f file.service.yml
```

Updating or Creating a Service

Use the kubectl apply command along with the --filename or -f switch

Deleting a Service

Use the kubectl delete command along with the --filename or -f switch

```
# Delete a Service
kubectl delete -f file.service.yml
```

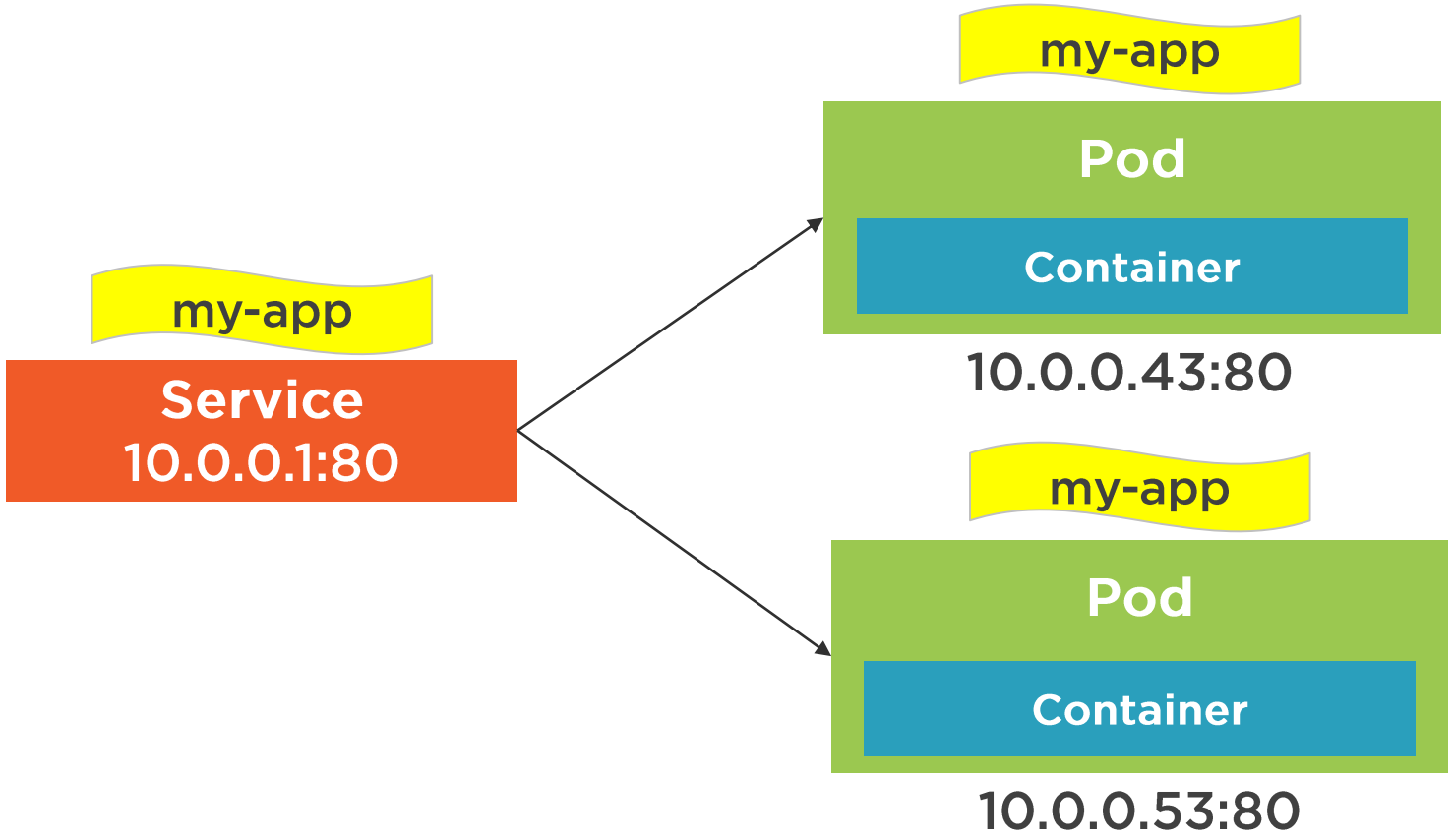
```
# Shell into a Pod and test a URL. Add -c [containerID]
# in cases where multiple containers are running in the Pod
kubectl exec [pod-name] -- curl -s http://podIP

# Install and use curl (example shown is for Alpine Linux)
kubectl exec [pod-name] -it sh
> apk add curl
> curl -s http://podIP
```

Testing a Service and Pod with curl

How can you quickly test if a Service and Pod is working?

Use **kubectl exec** to shell into a Pod/Container

The Role of Services



kubectl Services in Action

Summary

Pods live and die so their IP address can change

Services abstract Pod IP addresses from consumers

Labels associate a Service with a Pod

Service types:

- ClusterIP (internal to cluster – default)
- NodePort (exposes Service on each's Node's IP)
- LoadBalancer (exposes a Service externally)
- ExternalName (proxies to an external service)

Understanding Storage Options

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

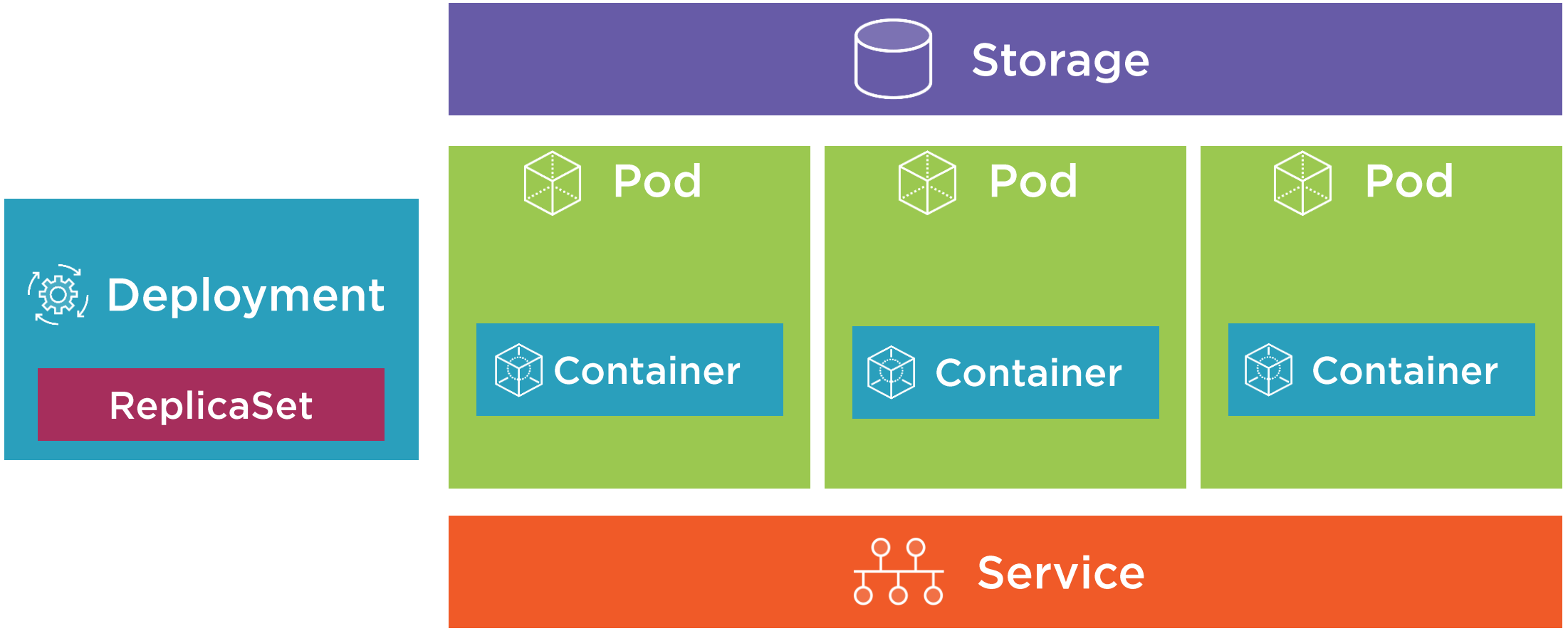
Storage Core Concepts

Volumes

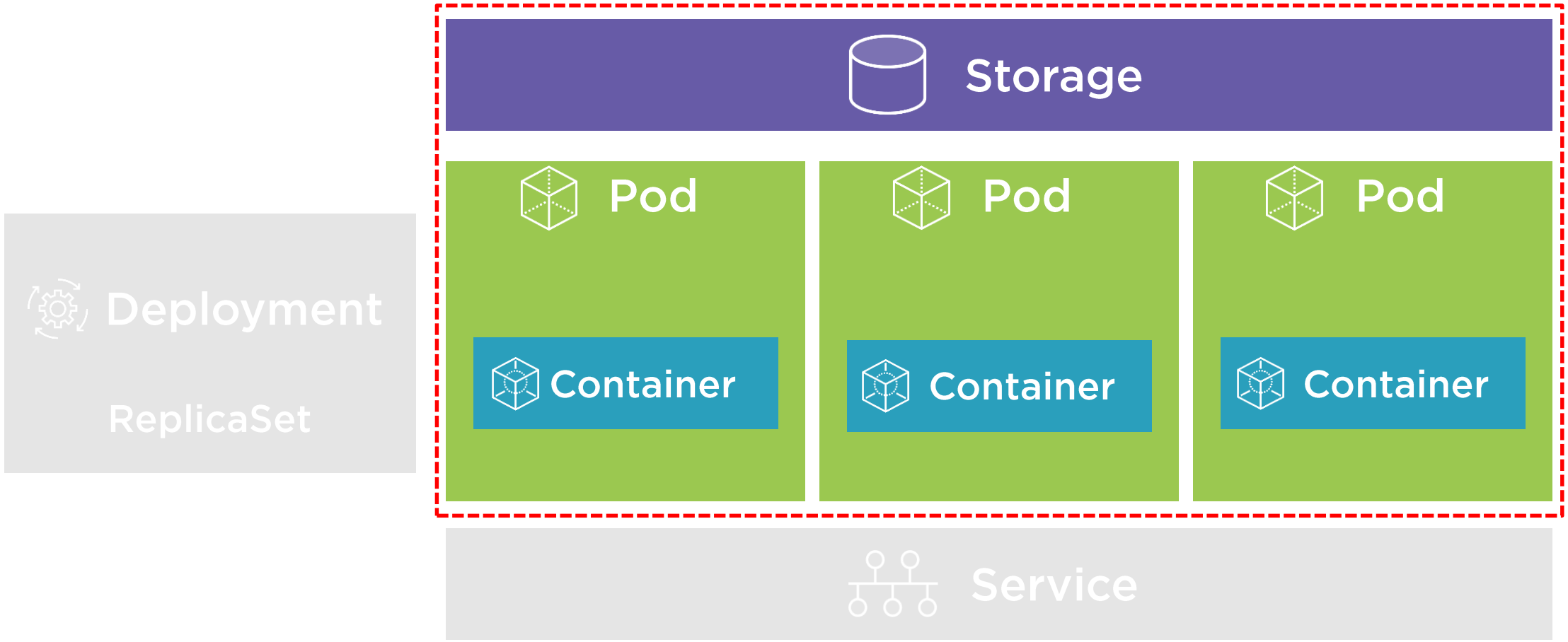
PersistentVolumes and PersistentVolumeClaims

StorageClasses

You Are Here


Storage
Pod
Pod
Pod
Deployment
ReplicaSet
Container
Container
Container
Service

You Are Here


Deployment
ReplicaSet
Storage
Pod
Container
Pod
Container
Pod
Container
Service

Storage Core Concepts

Question:

How do you store application state/data and exchange it between Pods with Kubernetes?

Answer:

Volumes (although other data storage options exist)

A Volume can be used to hold data and state for Pods and containers.

Pod State and Data

Pods live and die so their file system is short-lived (ephemeral)

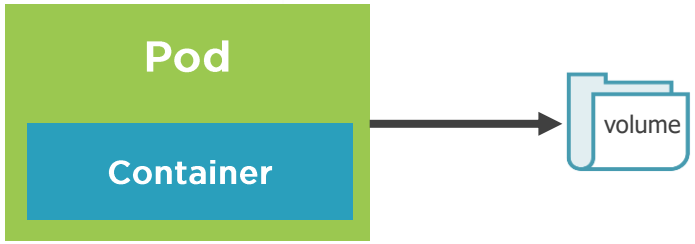
Volumes can be used to store state/data and use it in a Pod

A Pod can have multiple Volumes attached to it

Containers rely on a mountPath to access a Volume

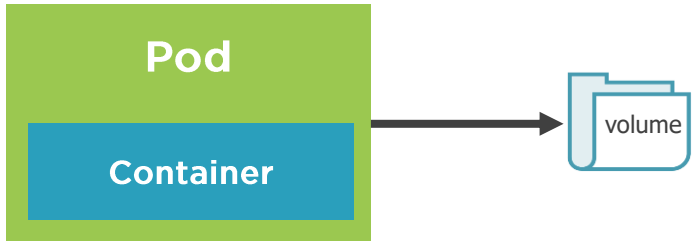
Kubernetes supports:

- Volumes
- PersistentVolumes
- PersistentVolumeClaims
- StorageClasses



Volumes

Volumes and Volume Mounts



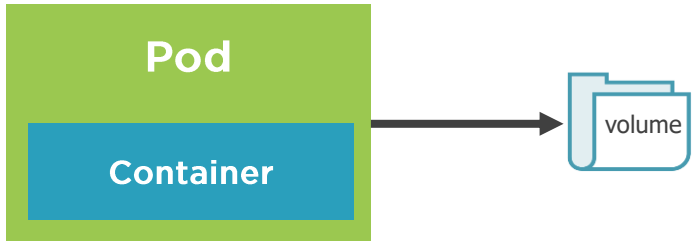
A Volume references a storage location

Must have a unique name

Attached to a Pod and may or may not be tied to the Pod's lifetime (depending on the Volume type)

A Volume Mount references a Volume by name and defines a mountPath

Volumes Type Examples



emptyDir – Empty directory for storing "transient" data (shares a Pod's lifetime) useful for sharing files between containers running in a Pod

hostPath – Pod mounts into the node's filesystem

nfs – An NFS (Network File System) share mounted into the Pod

configMap/secret – Special types of volumes that provide a Pod with access to Kubernetes resources

persistentVolumeClaim – Provides Pods with a more persistent storage option that is abstracted from the details

Cloud – Cluster-wide storage

Volume Types

awsElasticBlockStore	azureDisk	azureFile	cephfs	configMap
csi	downwardAPI	emptyDir	fc	flexVolume
flocker	gcePersistentDisk	glusterfs	hostPath	iscsi
local	nfs	persistentVolumeClaim	projected	portworxVolume
quobyte	rbd	scaleIO	secret	storageos
vsphereVolume				

Defining an emptyDir Volume

```
apiVersion: v1
kind: Pod
spec:
 volumes:
     - name: html
       emptyDir: {}
  containers:
  - name: nginx
    image: nginx:alpine
    volumeMounts:
      - name: html
        mountPath: /usr/share/nginx/html
        readOnly: true
  - name: html-updater
    image: alpine
    command: ["/bin/sh", "-c"]
    args:
      - while true; do date >> /html/index.html;
          sleep 10; done
    volumeMounts:
      - name: html
        mountPath: /html
```

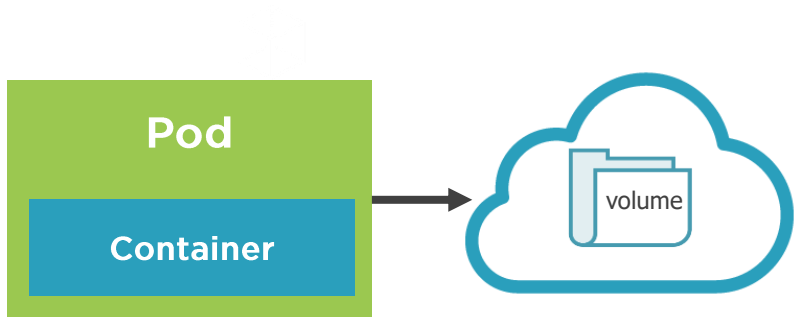
- Define initial Volume named "html" that is an empty directory (lifetime of the Pod)
- Reference "html" Volume and define a mountPath
- Update file in Volume mount /html path with latest date every 10 seconds
- Reference "html" Volume (defined above) and define a mountPath

Defining a hostPath Volume

```
apiVersion: v1
kind: Pod
spec:
 volumes:
   - name: docker-socket
     hostPath:
       path: /var/run/docker.sock
       type: Socket
  containers:
  - name: docker
    image: docker
    command: ["sleep"]
    args: ["100000"]
    volumeMounts:
      - name: docker-socket
        mountPath: /var/run/docker.sock
```

- Define a socket volume on host that points to /var/run/docker.sock
- Reference "docker-socket" Volume and define mountPath

Cloud Volumes



Cloud providers (Azure, AWS, GCP, etc.) support different types of Volumes:

- Azure – Azure Disk and Azure File
- AWS – Elastic Block Store
- GCP – GCE Persistent Disk

Defining an Azure File Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  volumes:
  - name: data
    azureFile:
      secretName: <azure-secret>
      shareName: <share-name>
      readOnly: false
  containers:
  - image: someimage
    name: my-app
    volumeMounts:
    - name: data
      mountPath: /data/storage
```

- Define initial Volume named "data" that is Azure File storage
- Reference "data" Volume and define a mountPath

Defining an AWS Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  volumes:
  - name: data
    awsElasticBlockStore:
      volumeID: <volume_ID>
      fsType: ext4
  containers:
  - image: someimage
    name: my-app
    volumeMounts:
    - name: data
      mountPath: /data/storage
```

- Define initial Volume named "data" that is a awsElasticBlockStore
- Reference "data" Volume and define a mountPath

Defining a Google Cloud gcePersistentDisk Volume

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  volumes:
  - name: data
    gcePersistentDisk:
      pdName: datastorage
      fsType: ext4
  containers:
  - image: someimage
    name: my-app
    volumeMounts:
    - name: data
      mountPath: /data/storage
```

- Define initial Volume named "data" that is a gcePersistentDisk
- Reference "data" Volume and define a mountPath

Viewing a Pod's Volumes

Several different techniques can be used to view a Pod's Volumes

```
# Describe Pod
kubectl describe pod [pod-name]
```

```
Volumes:
  html:
    Type:    EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
```

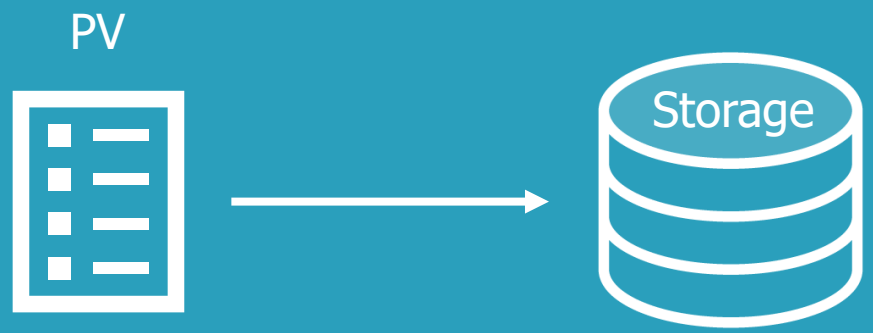
```
# Get Pod YAML
kubectl get pod [pod-name] -o yaml
```

```
volumeMounts:
- mountPath: /html
  name: html
```

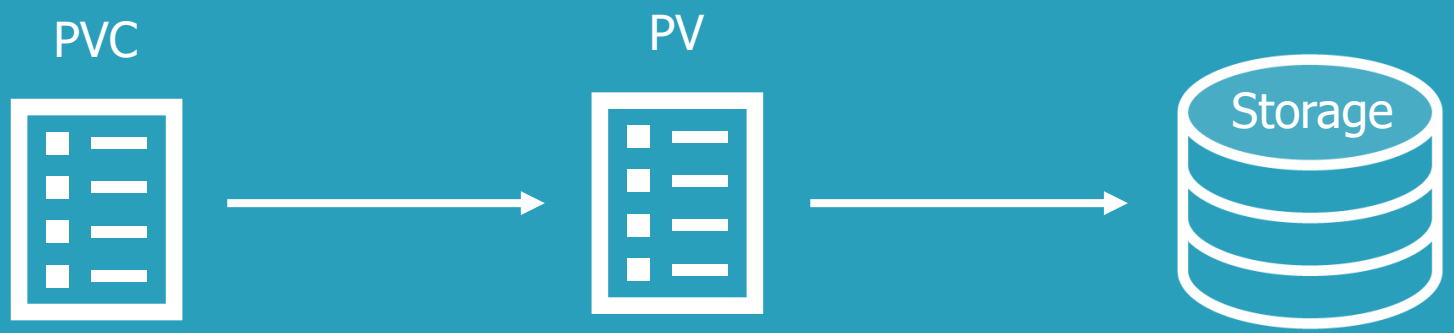
Volumes in Action

PersistentVolumes and PersistentVolumeClaims

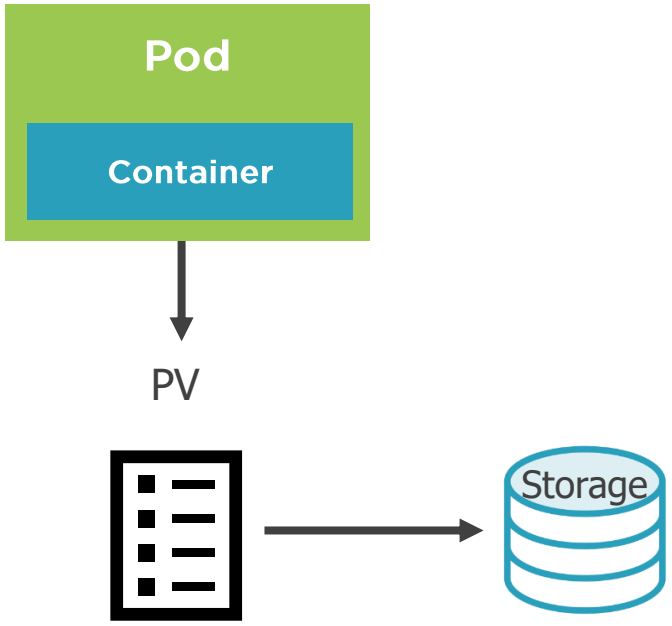
A PersistentVolume (PV) is a cluster-wide storage unit provisioned by an administrator with a lifecycle independent from a Pod.



A PersistentVolumeClaim (PVC) is a request for a storage unit (PV).



PersistentVolume



A PersistentVolume is a cluster-wide storage resource that relies on network-attached storage (NAS)

Normally provisioned by a cluster administrator

Available to a Pod even if it gets rescheduled to a different Node

Rely on a storage provider such as NFS, cloud storage, or other options

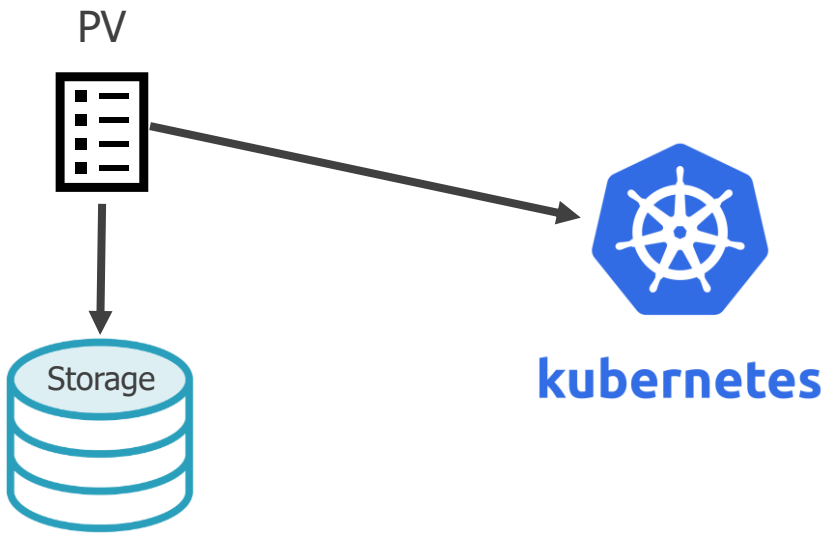
Associated with a Pod by using a PersistentVolumeClaim (PVC)

PersistentVolume Workflow

1. **Create network storage resource (NFS, cloud, etc.)**

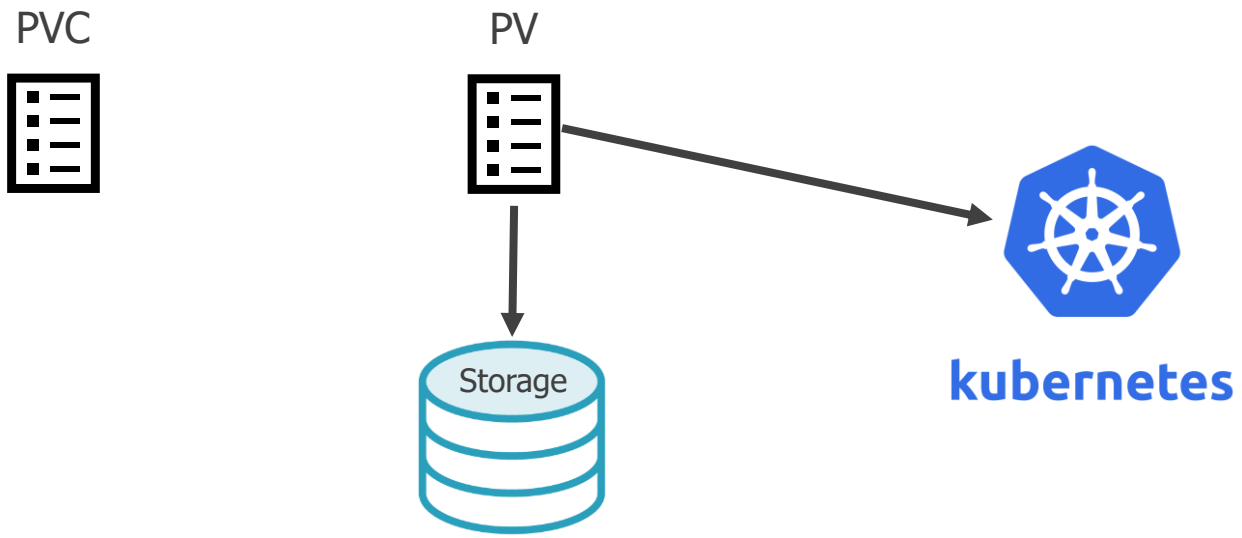
PersistentVolume Workflow

2 **Define a Persistent Volume (PV) and send to the Kubernetes API**



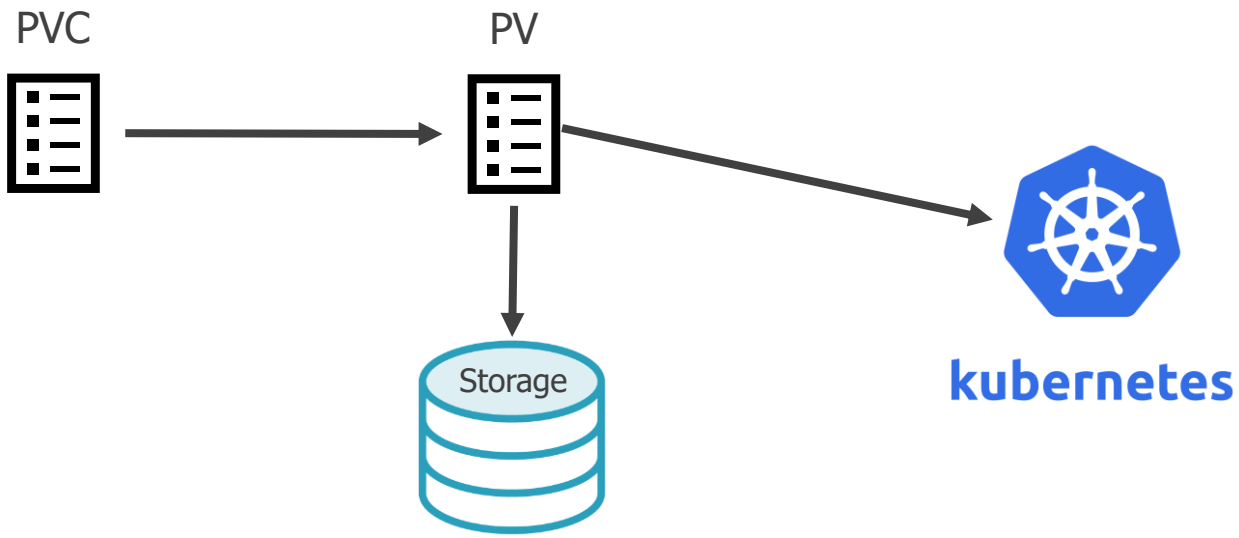
PersistentVolume Workflow

3 **Create a PersistentVolumeClaim (PVC)**



PersistentVolume Workflow

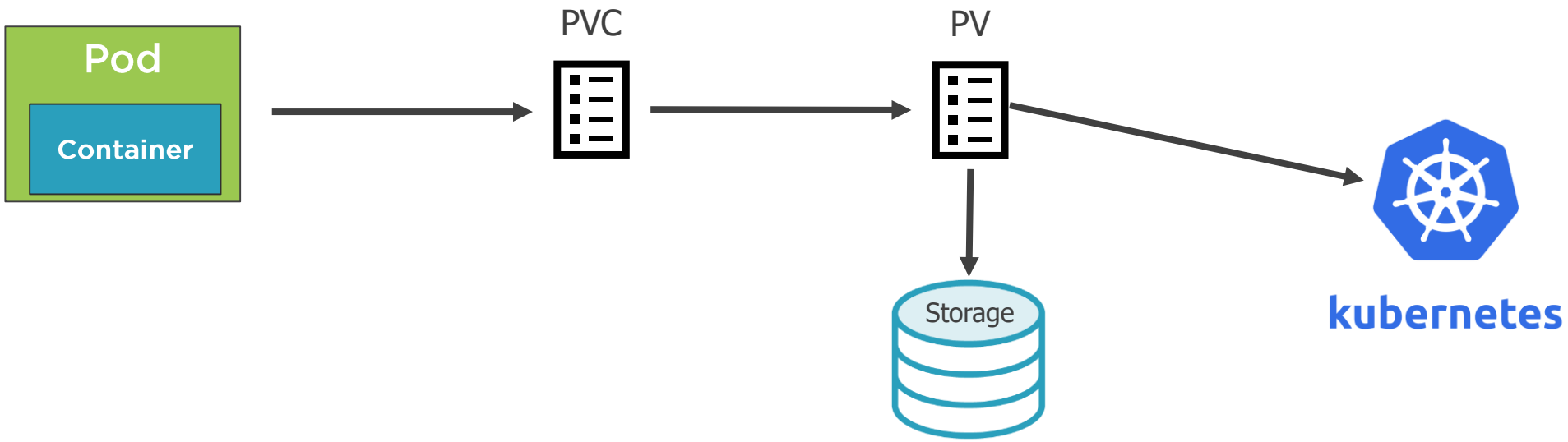
4 **Kubernetes binds the PVC to the PV**



PersistentVolume Workflow

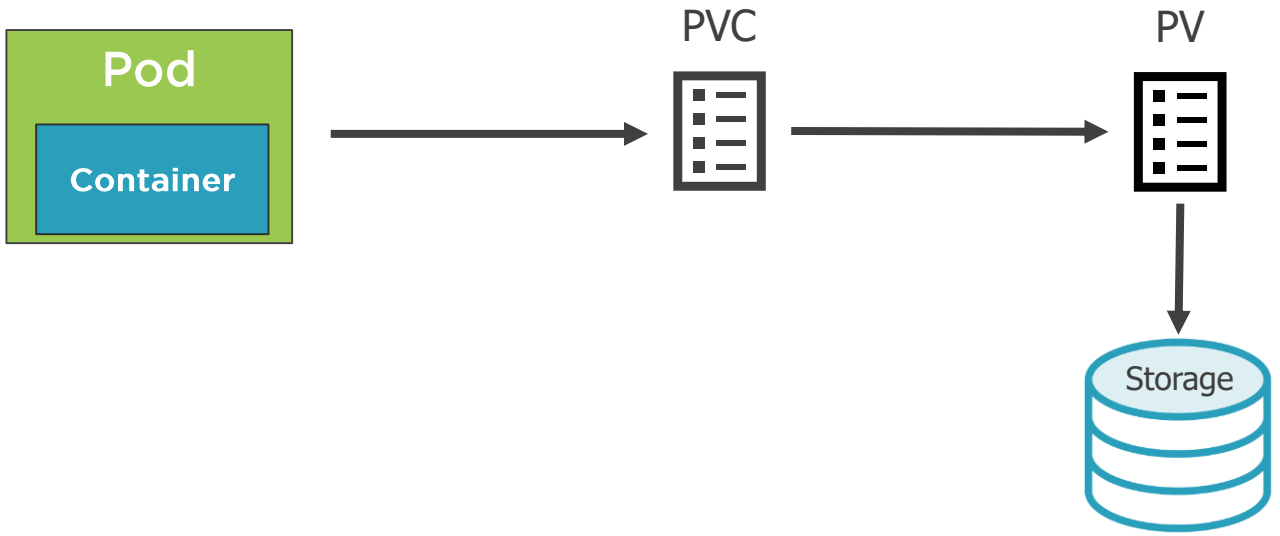
5

Pod Volume references the PVC

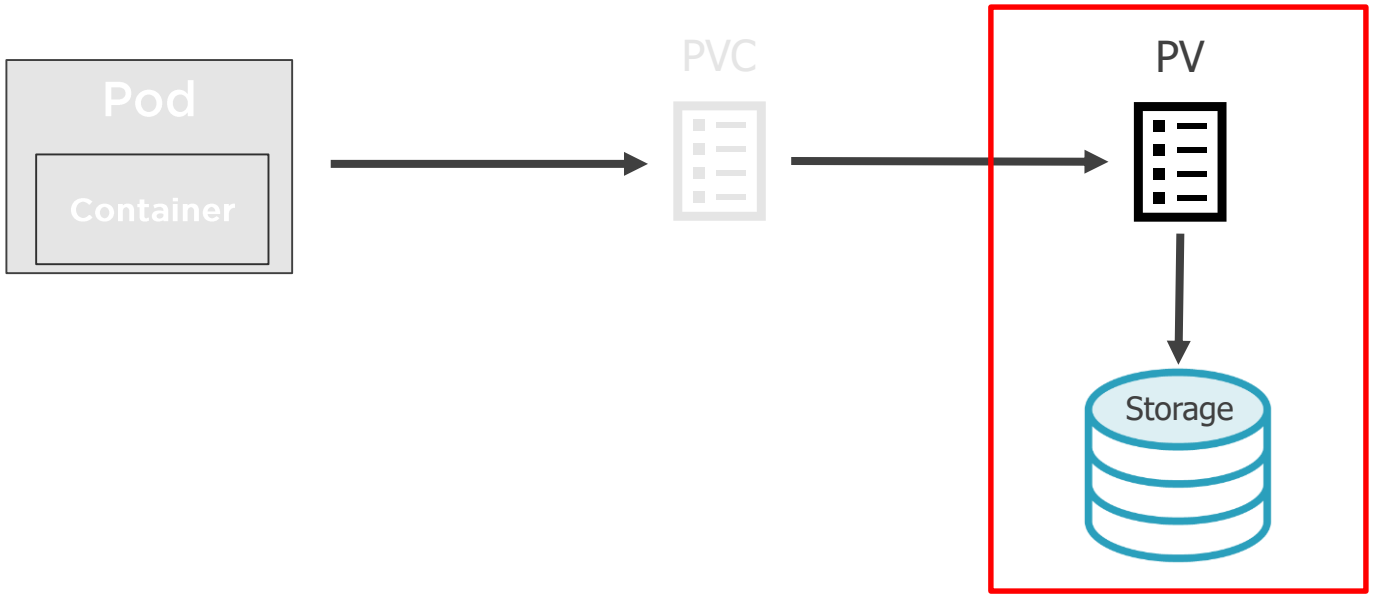


PersistentVolume and PersistentVolumeClaim YAML

Defining a PV and PVC



Creating a PersistentVolume



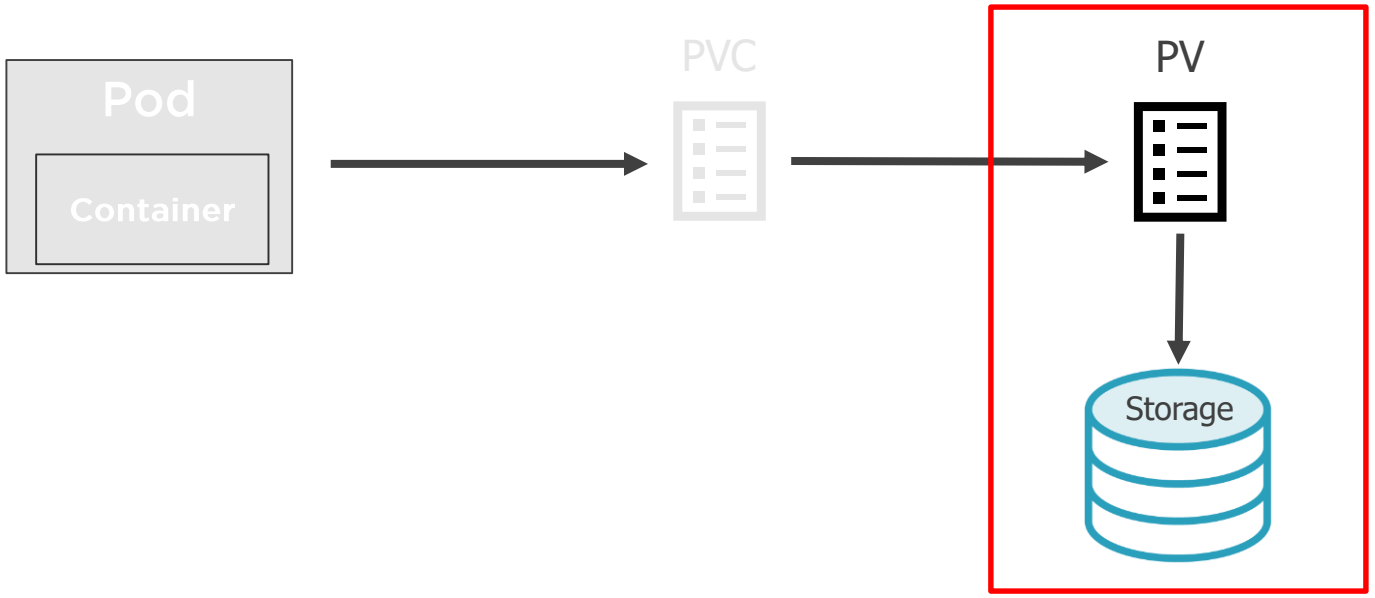
Defining a PersistentVolume for Azure

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity: 10Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  persistentVolumeRelaimPolicy: Retain
  azureFile:
    secretName: <azure-secret>
    shareName: <name_from_azure>
    readOnly: false
```

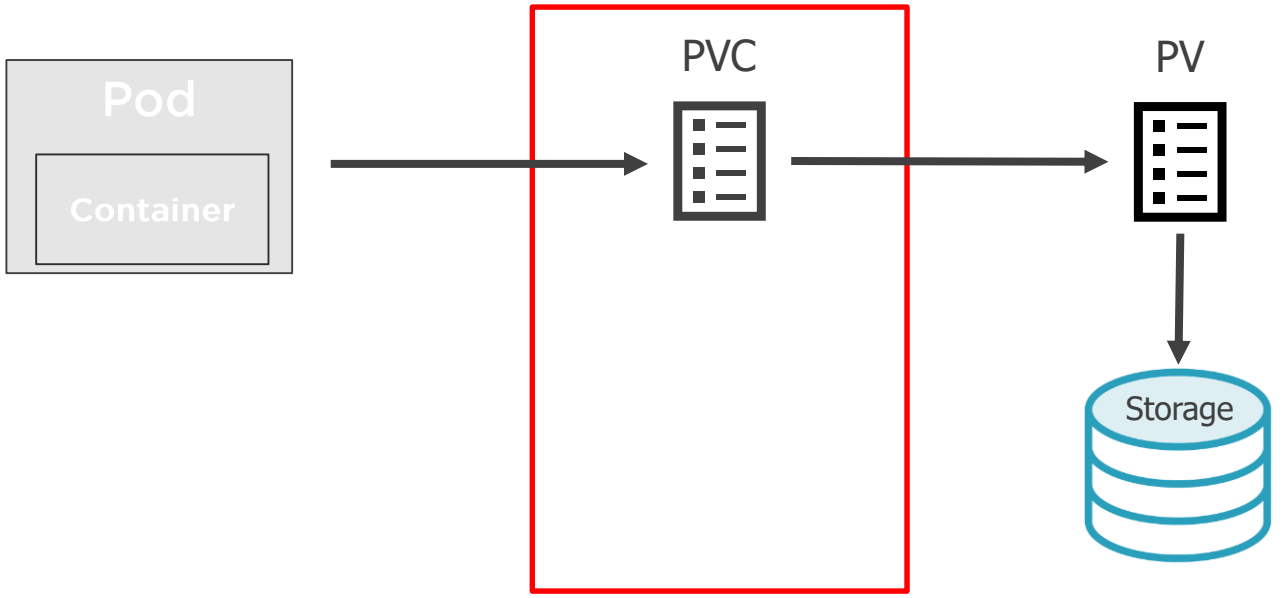
- Create PersistentVolume kind
- Define storage capacity
- One client can mount for read/write
- Many clients can mount for reading
- Retain even after claim is deleted (not erased/deleted)
- Reference storage to use (specific to Cloud provider, NFS setup, etc.)

https://github.com/kubernetes/examples

Creating a PersistentVolume



Creating a PersistentVolumeClaim

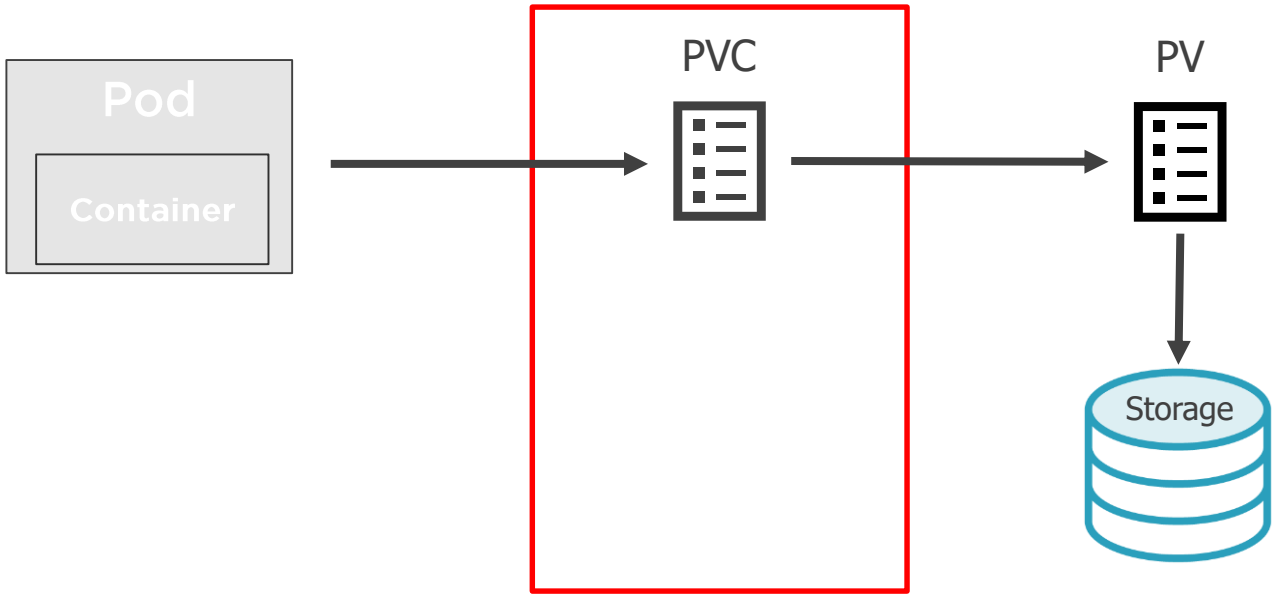


Defining a PersistentVolumeClaim

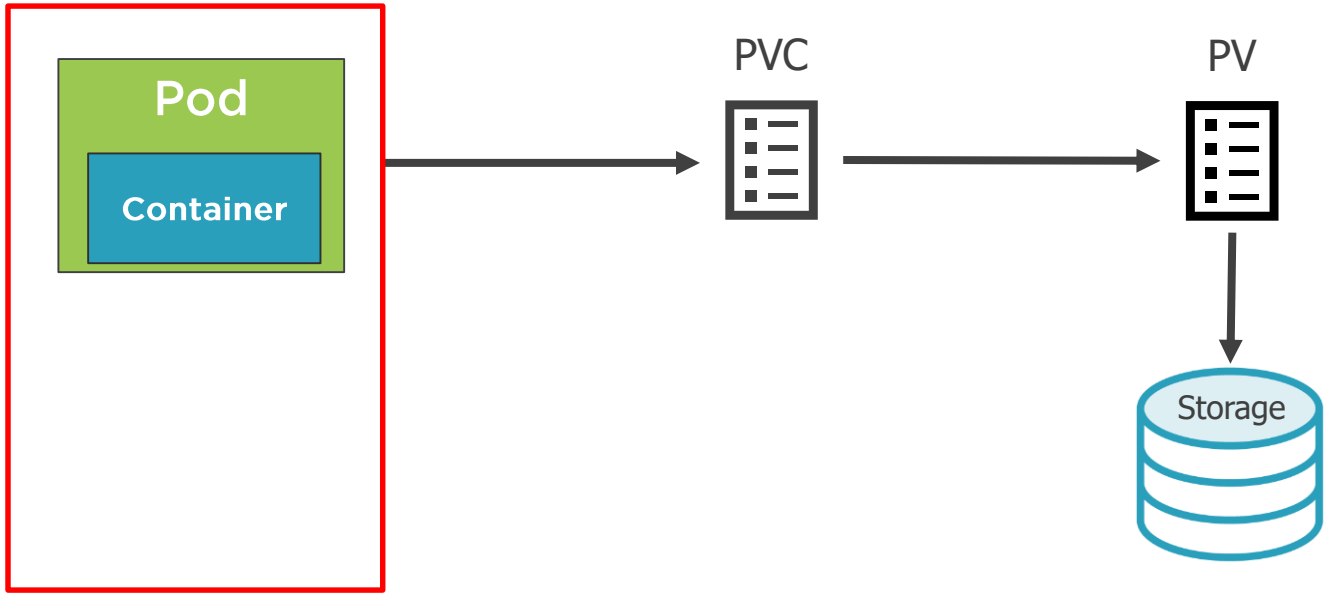
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pv-dd-account-hdd-5g
  annotations:
    volume.beta.kubernetes.io/storage-class: accounthdd
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

- Define a PersistentVolumeClaim (PVC)
- Define access mode
- Request storage amount

Creating a PersistentVolumeClaim



Defining a Volume that Uses a PVC



Using a PersistentVolumeClaim

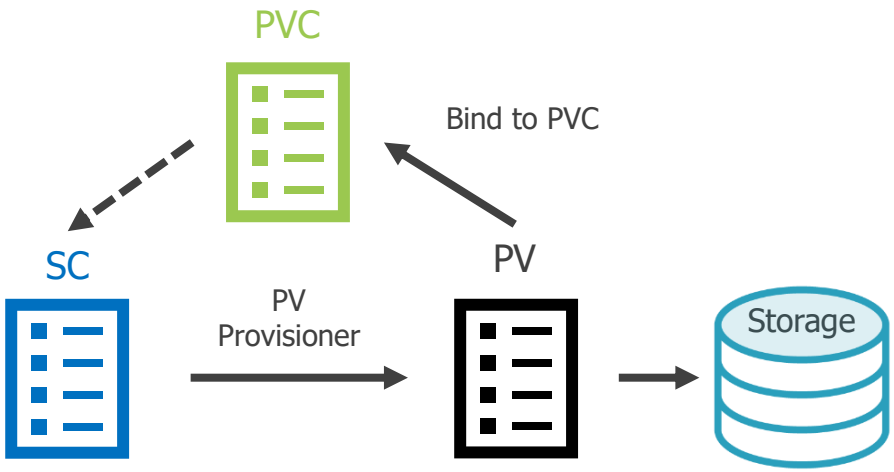
```
kind: Pod
apiVersion: v1
metadata:
  name: pod-uses-account-hdd-5g
  labels:
    name: storage
spec:
  containers:
  - image: nginx
    name: az-c-01
    command:
    - /bin/sh
    - -c
    - while true; do echo $(date) >>
      /mnt/blobdisk/outfile; sleep 1; done
    volumeMounts:
    - name: blobdisk01
      mountPath: /mnt/blobdisk
  volumes:
  - name: blobdisk01
    persistentVolumeClaim:
      claimName: pv-dd-account-hdd-5g
```

- Mount to Volume
- Create Volume that binds to PersistentVolumeClaim

StorageClasses

A StorageClass (SC) is a type of storage template that can be used to dynamically provision storage.

StorageClass



Used to define different "classes" of storage

Act as a type of storage template

Supports dynamic provisioning of PersistentVolumes

Administrators don't have to create PVs in advance

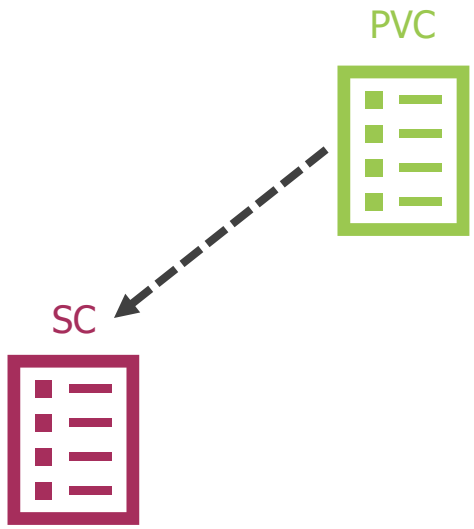
StorageClass Workflow

1 **Create Storage Class**

StorageClass Workflow

2

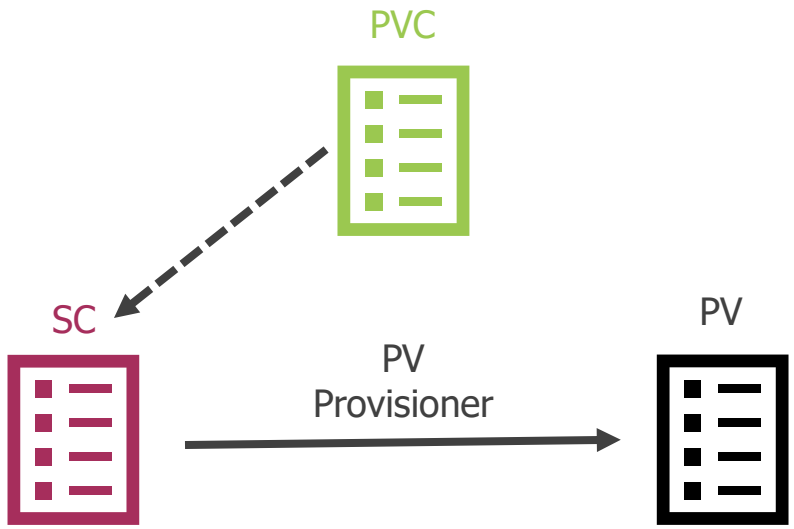
Create PersistentVolumeClaim that references StorageClass



StorageClass Workflow

3

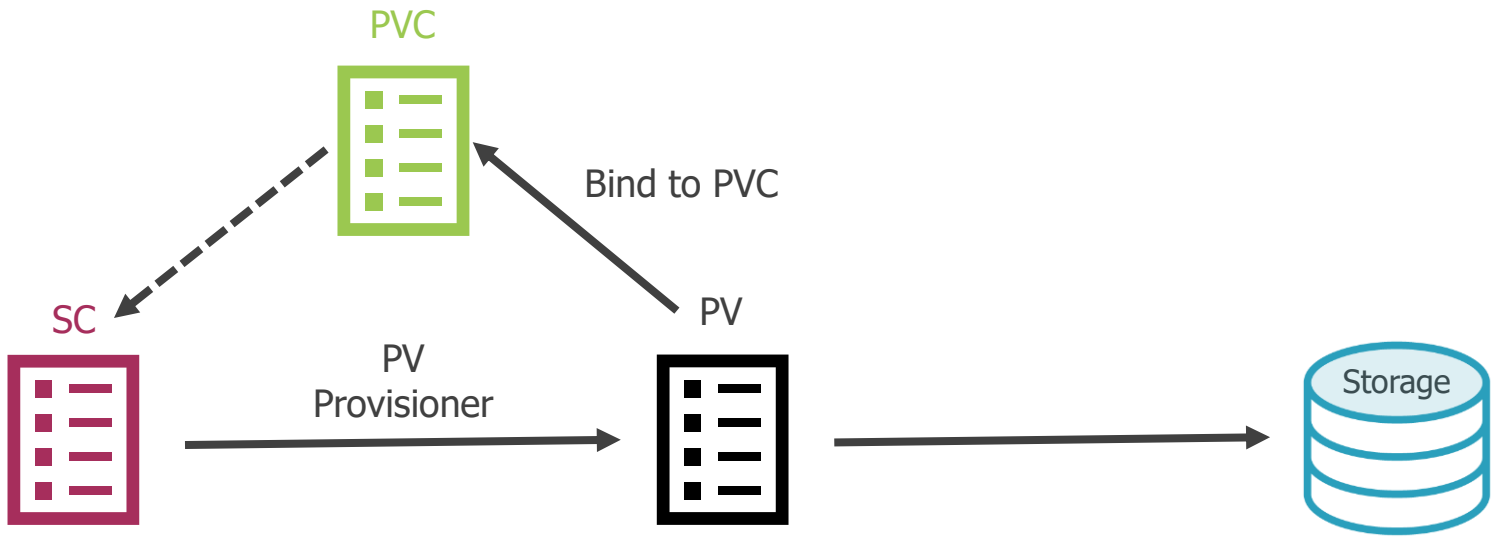
Kubernetes uses StorageClass provisioner to provision a PersistentVolume



StorageClass Workflow

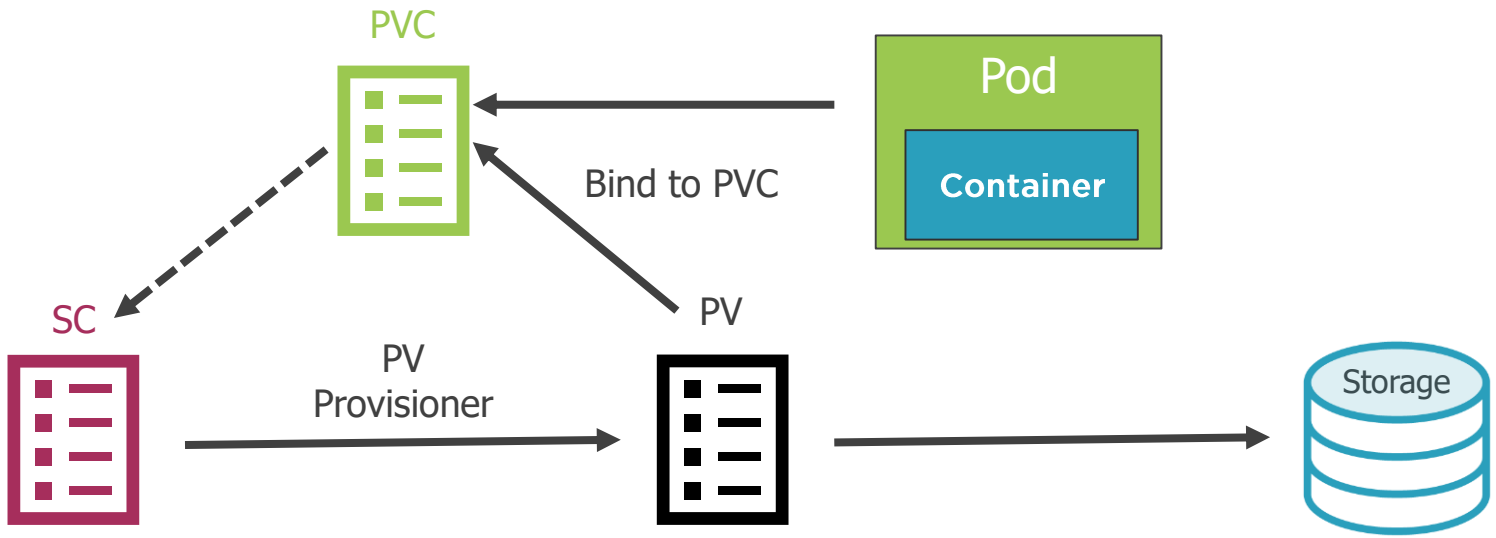
4

Storage provisioned, PersistentVolume created and bound to PersistentVolumeClaim



StorageClass Workflow

5 **Pod volume references PersistentVolumeClaim**



Defining a Local Storage StorageClass

```
apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

  name: local-storage

reclaimPolicy: Retain

provisioner: kubernetes.io/no-provisioner

volumeBindingMode: WaitForFirstConsumer
```

- API version
- A StorageClass resource
- Retain storage or Delete (default) after PVC is released
- Provisioner (volume plugin) that will be used to create PersistentVolume resource.
- Wait to create until Pod making PVC is created. Default is Immediate (create once PVC is created)

Defining a Local Storage PersistentVolume

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 10Gi
  volumeMode: Block
  accessModes:
  - ReadWriteOnce
  storageClassName: local-storage
  local:
    path: /data/storage
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - <node-name>
```

- One client can mount for read/write
- Reference StorageClass
- Path where data is stored on Node
- Select the Node where the local storage PV is created

Defining a PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: local-storage
  resources:
    requests:
      storage: 1Gi
```

- Define a PersistentVolumeClaim (PVC)
- Access Mode and storage classification PV needs to support
- Storage request information

Using a PersistentVolumeClaim

```
apiVersion: apps/v1
kind: [Pod | StatefulSet | Deployment]
...
  spec:
    volumes:
    - name: my-volume
      persistentVolumeClaim:
        claimName: my-pvc
```

- Define a Volume
- Use a PVC to claim the required storage

PersistentVolumes in Action

Summary

Kubernetes supports several different types of storage:

- Ephemeral storage (emptyDir)
- Persistent storage (many options)
- PersistentVolumes, PersistentVolumeClaims, and StorageClasses
- ConfigMaps (key/value pairs)
- Secrets

Creating ConfigMaps and Secrets

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

ConfigMaps Core Concepts

Creating a ConfigMap

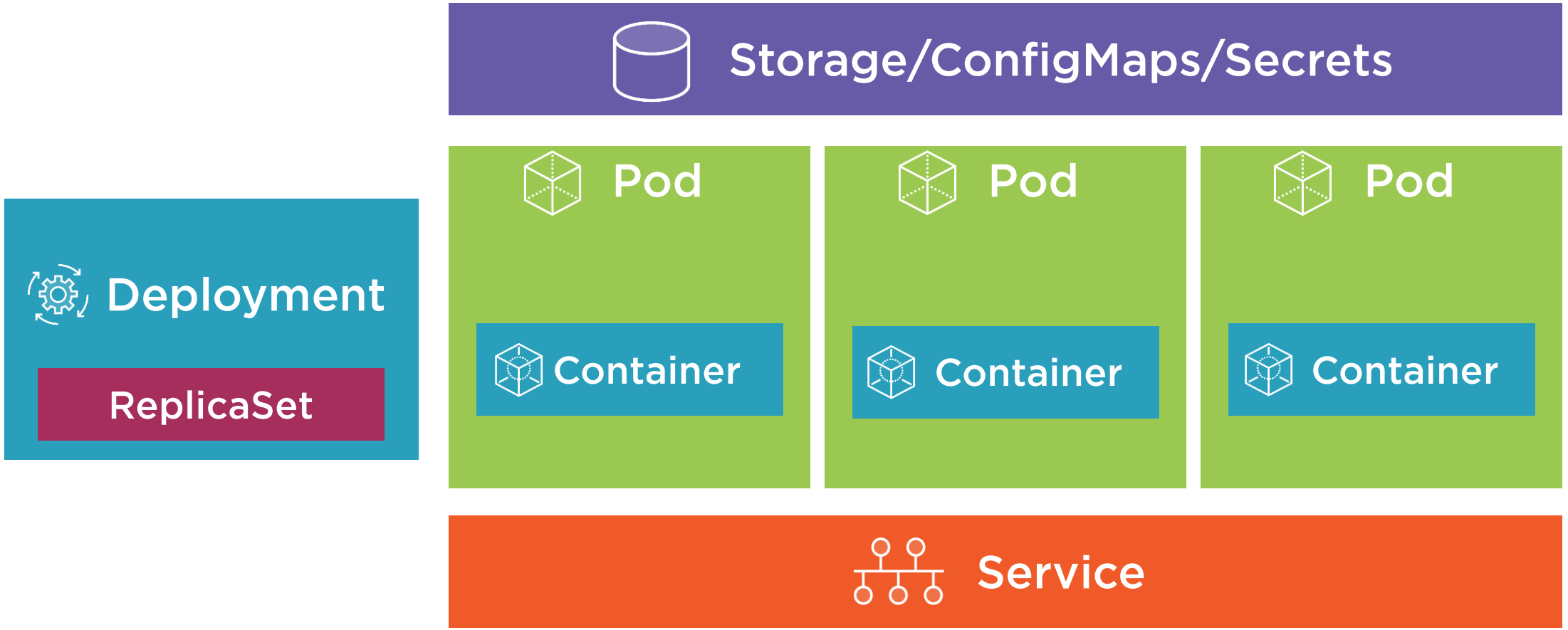
Using a ConfigMap

Secrets Core Concepts

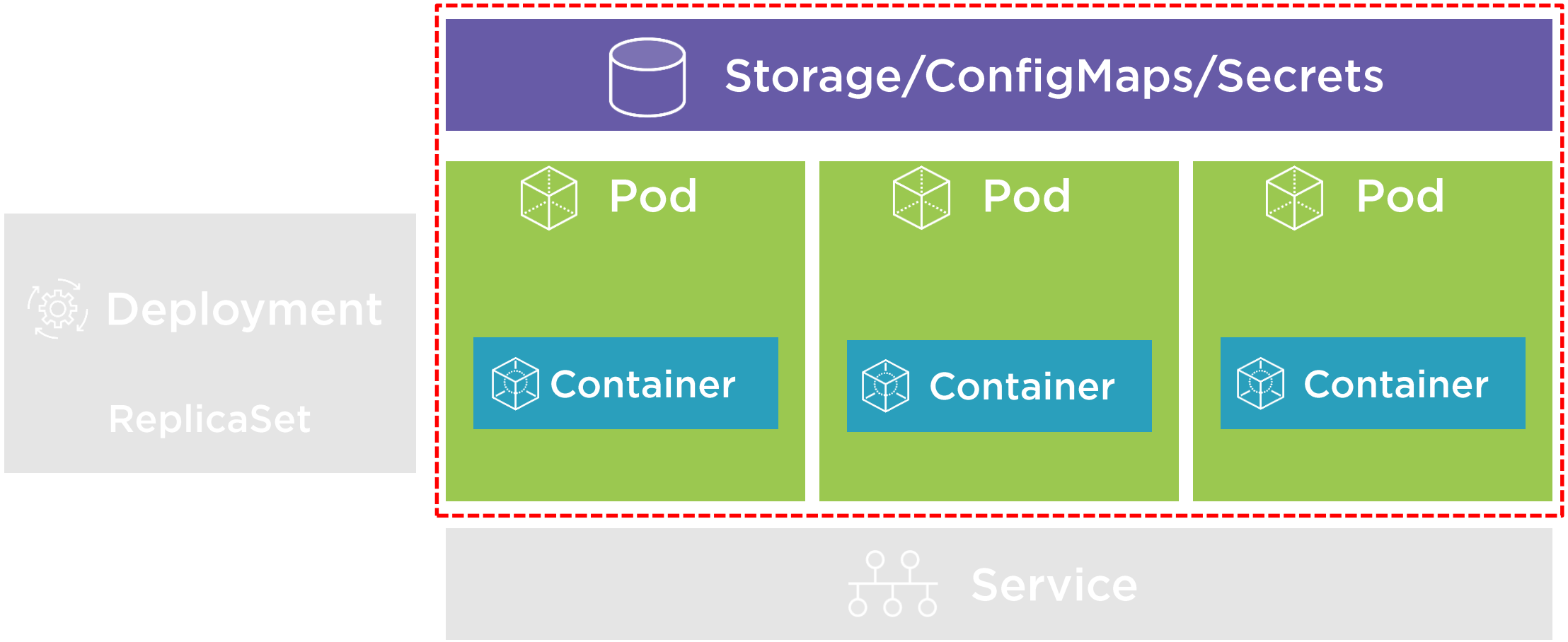
Creating a Secret

Using a Secret

You Are Here


Storage/ConfigMaps/Secrets
Pod
Pod
Pod
Deployment
ReplicaSet
Container
Container
Container
Service

You Are Here


Storage/ConfigMaps/Secrets
Pod
Pod
Pod
Container
Container
Container
Deployment
ReplicaSet
Service

ConfigMaps Core Concepts

ConfigMaps provide a way to store configuration information and provide it to containers.

ConfigMaps

Provides a way to inject configuration data into a container

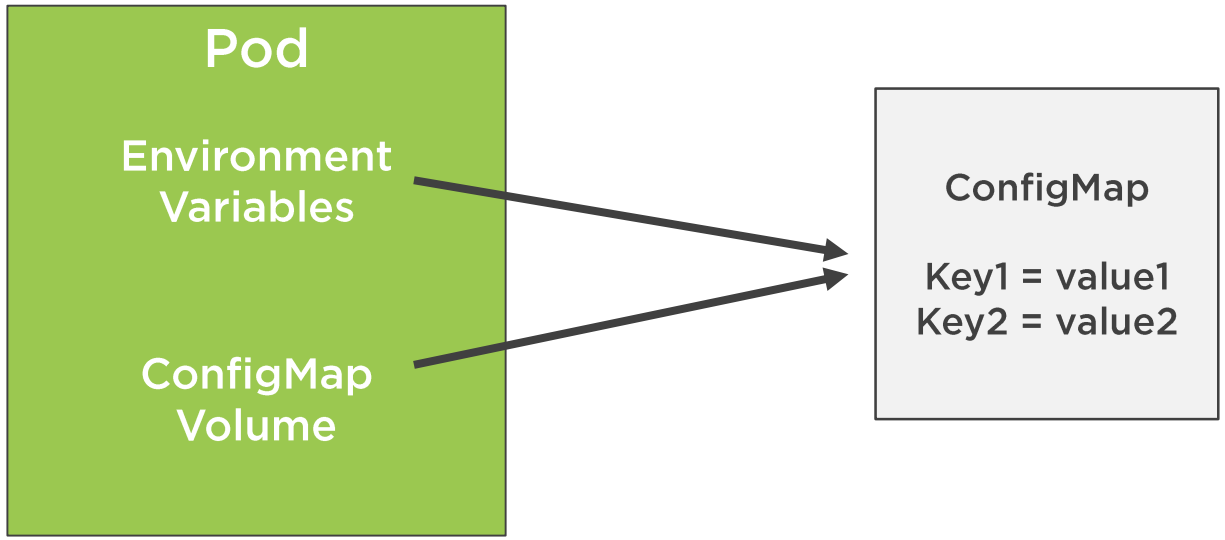
Can store entire files or provide key/value pairs:

- Store in a File. Key is the filename, value is the file contents (can be JSON, XML, keys/values, etc.).
- Provide on the command-line
- ConfigMap manifest

Accessing ConfigMap Data in a Pod

ConfigMaps can be accessed from a Pod using:

- **Environment variables (key/value)**
- **ConfigMap Volume (access as files)**



Creating a ConfigMap

Defining Values in a ConfigMap Manifest

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
  labels:
    app: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
# Create from a ConfigMap manifest
kubectl create -f file.configmap.yml
```

- A ConfigMap resource
- Name of ConfigMap
- ConfigMap data

Defining Key/Value Pairs in a File

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
```

```
# Create a ConfigMap using data from a file
kubectl create configmap [cm-name]
  --from-file=[path-to-file]
```

```
apiVersion: v1
kind: ConfigMap
data:
  game.config: |-
    enemies=aliens
    lives=3
    enemies.cheat=true
    enemies.cheat.level=noGoodRotten
```

- Key/value pairs defined in a file named game.config
- Nested properties can be defined and assigned a value

- Note that the file name is used as the key for the values
- Your application can now work with the content just as it would a normal configuration file (JSON, XML, keys/values, could be used)

Defining Key/Value Pairs in an Env File

```
enemies=aliens
lives=3
enemies.cheat=true
enemies.cheat.level=noGoodRotten
```

```
# Create a env ConfigMap using data from a file
kubectl create configmap [cm-name]
  --from-env-file=[path-to-file]
```

```
apiVersion: v1
kind: ConfigMap
data:
  enemies=aliens
  lives=3
  enemies.cheat=true
  enemies.cheat.level=noGoodRotten
```

- Key/value pairs can be defined in an "environment" variables file (game-config.env)
- Nested properties can be defined and assigned a value
- Note that the file name is NOT included as a key

```
# Create a ConfigMap using data from a config file
kubectl create configmap [cm-name] --from-file=[path-to-file]

# Create ConfigMap from an env file
kubectl create configmap [cm-name] --from-env-file=[path-to-file]


# Create a ConfigMap from individual data values
kubectl create configmap [cm-name]
  --from-literal=apiUrl=https://my-api
  --from-literal=otherKey=otherValue

# Create from a ConfigMap manifest
kubectl create -f file.configmap.yml
```

Creating a ConfigMap

A ConfigMap can be created using **kubectl create**

Key command-line switches include:

- --from-file
- --from-env-file
- --from-literal

Using a ConfigMap

```
# Get a ConfigMap
kubectl get cm [cm-name] -o yaml
```

Getting a ConfigMap

kubectl get cm can be used to get a ConfigMap and view its contents

Accessing a ConfigMap: Environment Vars

Pods can access ConfigMap values through environment vars

ENEMIES environment variable created (value=aliens)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    containers: ...
    env:
    - name: ENEMIES
      valueFrom:
        configMapKeyRef:
          name: app-settings
          key: enemies
```

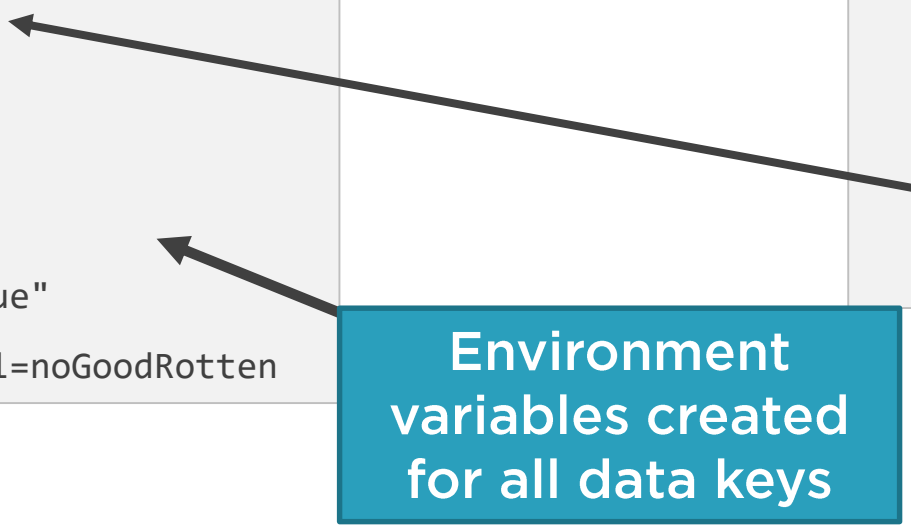
Environment variable name

Accessing a ConfigMap: Environment Vars

envFrom can be used to load all ConfigMap keys/values into environment variables

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    containers: ...
      envFrom:
      - configMapRef:
        name: app-settings
```



Accessing a ConfigMap: Volume

ConfigMap values can be loaded through a Volume

Each key is converted to a file - value is added into the file

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-settings
data:
  enemies: aliens
  lives: "3"
  enemies.cheat: "true"
  enemies.cheat.level=noGoodRotten
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    volumes:
      - name: app-config-vol
        configMap:
          name: app-settings
    containers:
      volumeMounts:
        - name: app-config-vol
          mountPath: /etc/config
```

ConfigMaps in Action

Secrets Core Concepts

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.

Secrets

Kubernetes can store sensitive information (passwords, keys, certificates, etc.)

Avoids storing secrets in container images, in files, or in deployment manifests

Mount secrets into pods as files or as environment variables

Kubernetes only makes secrets available to Nodes that have a Pod requesting the secret

Secrets are stored in tmpfs on a Node (not on disk)

Enable encryption at rest for cluster data (https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data)

Limit access to etcd (where Secrets are stored) to only admin users

Use SSL/TLS for etcd peer-to-peer communication

Manifest (YAML/JSON) files only base64 encode the Secret

Pods can access Secrets so secure which users can create Pods. Role-based access control (RBAC) can be used.

https://kubernetes.io/docs/concepts/configuration/secret/#best-practices

Secrets Best Practices

Creating a Secret

Creating a Secret

Secrets can be created using kubectl create secret

```
# Create a secret and store securely in Kubernetes
kubectl create secret generic my-secret
  --from-literal=pwd=my-password

# Create a secret from a file
kubectl create secret generic my-secret
  --from-file=ssh-privatekey=~/.ssh/id_rsa
  --from-file=ssh-publickey=~/.ssh/id_rsa.pub

# Create a secret from a key pair
kubectl create secret tls tls-secret --cert=path/to/tls.cert
  --key=path/to/tls.key
```

Question:

Can I declaratively define secrets using YAML?

Answer:

Yes – but any secret data is only base64 encoded in the manifest file!

Defining a Secret in YAML

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  app-password: cGFzc3dvcmQ=
  admin-password: dmVyeV9zZWNyZXQ=
```

- Define a Secret
- Secret name
- Keys/values for Secret

Using a Secret

```
# Get secrets
kubectl get secrets
```

```
danwahlin — -bash — 73×6
iMac-3:~ danwahlin$ k get secrets
+ kubectl get secrets
NAME                  TYPE                                  DATA   AGE
db-passwords          Opaque                                2      34m
default-token-rxmjb   kubernetes.io/service-account-token   3      66d
```

```
# Get YAML for specific secret
kubectl get secrets db-passwords -o yaml
```

```
danwahlin — -bash — 73×16
iMac-3:~ danwahlin$ k get secrets db-passwords -o yaml
+ kubectl get secrets db-passwords -o yaml
apiVersion: v1
data:
  mongodb-password: cGFzc3dvcmQ=
  mongodb-root-password: cGFzc3dvcmQ=
kind: Secret
metadata:
  creationTimestamp: "2019-03-22T00:40:05Z"
  name: db-passwords
  namespace: default
  resourceVersion: "3481795"
  selfLink: /api/v1/namespaces/default/secrets/db-passwords
  uid: 0982413e-4c3b-11e9-b7f0-025000000001
type: Opaque
```

Listing Secret Keys

A list of secrets can be retrieved using kubectl get secrets

Accessing a Secret: Environment Vars

Pods can access Secret values through environment vars

DATABASE_PASSWORD environment var created

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  db-password: cGFzc3dvcmQ=
  admin-password: dmVyeV9zZWNyZXQ=
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    containers: ...
    env:
    - name: DATABASE_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-passwords
          key: db-password
```

Accessing a Secret: Volumes

Pods can access secret values through a volume

Each key is converted to a file - value is added into the file

```
apiVersion: v1
kind: Secret
metadata:
  name: db-passwords
type: Opaque
data:
  db-password: cGFzc3dvcmQ=
  admin-password: dmVyeV9zZWNyZXQ=
```

```
apiVersion: apps/v1
...
spec:
  template:
    ...
  spec:
    volumes:
      - name: secrets
        secret:
          secretName: db-passwords
        containers:
        volumeMounts:
          - name: secrets
            mountPath: /etc/db-passwords
            readOnly: true
```

Secrets in Action

Summary

ConfigMaps provide a way to store configuration data

Secrets provide a way to store sensitive data or files

Access key/value pairs using environment variables or volumes

Use caution when working with Secrets and ensure proper security is in place

Putting It All Together

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

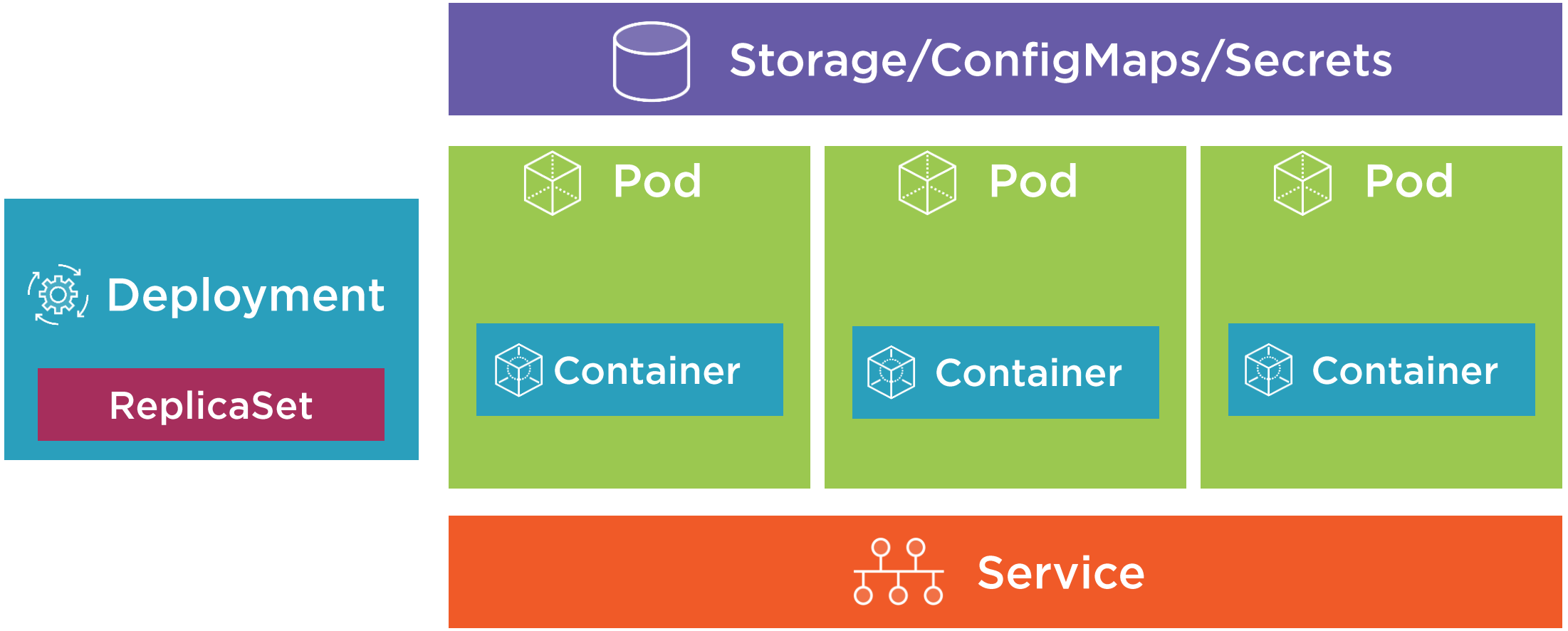
Application Overview

YAML Manifests

Running the Application

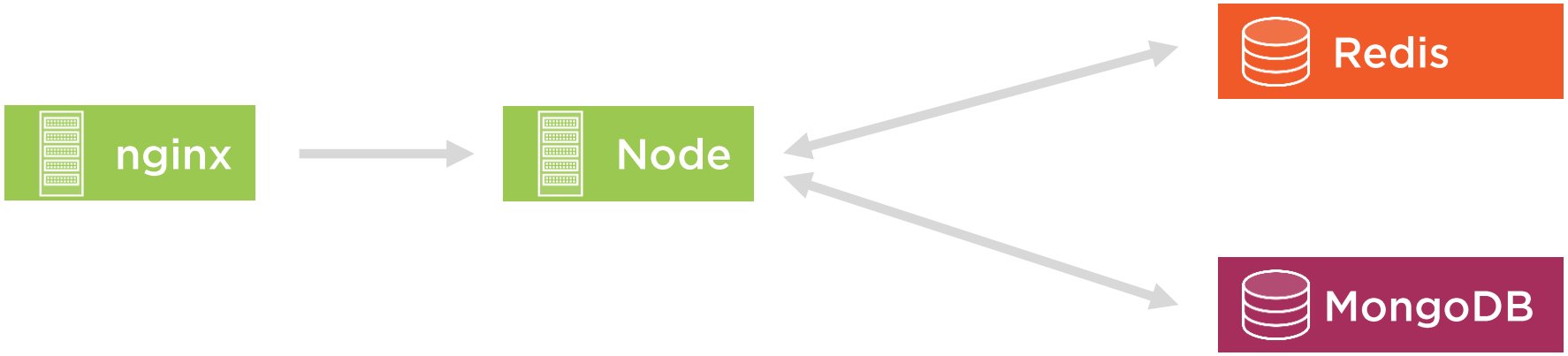
Troubleshooting Techniques

You Are Here


Storage/ConfigMaps/Secrets
Pod
Pod
Pod
Deployment
ReplicaSet
Container
Container
Container
Service

Application Overview

Application Overview



YAML Manifests

Running the Application

Troubleshooting Techniques

```
# View the logs for a Pod's container
kubectl logs [pod-name]

# View the logs for a specific container within a Pod
kubectl logs [pod-name] -c [container-name]

# View the logs for a previously running Pod
kubectl logs -p [pod-name]

# Stream a Pod's logs
kubectl logs -f [pod-name]
```

Accessing Logs

Pod container logs can be viewed using the kubectl logs command

```
# Describe a Pod
kubectl describe pod [pod-name]

# Change a Pod's output format
kubectl get pod [pod-name] -o yaml

# Change a Deployment's output format
kubectl get deployment [deployment-name] -o yaml
```

Getting Details About a Pod

Get details about a Pod using kubectl describe or kubectl get with -o

```
# Shell into a Pod container
kubectl exec [pod-name] -it sh
```

Shell into a Pod Container

Shell into a Pod container using kubectl exec

Summary

YAML manifest files are used to define different Kubernetes resources

kubectl create or apply can be used with -f to deploy multiple manifest files

Learning different Pod troubleshooting commands and techniques is important to resolve issues