Kubernetes for Developers: Deploying Your Code

KUBERNETES DEPLOYMENTS OVERVIEW

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Course Overview

- Kubernetes Deployments Overview
- Performing Rolling Update Deployments
- Performing Canary Deployments
- Performing Blue-Green Deployments
- Running Jobs and CronJobs
- Performing Monitoring and Troubleshooting Tasks
- Putting It All Together

Target Audience

Developers looking to learn different techniques for deploying code to Kubernetes

Course Pre-Reqs

Comfortable using command-line tools and virtual machines

General understanding of Docker containers and how they work

Understand Kubernetes core concepts

It's recommended that you watch the **Kubernetes for Developers: Core Concepts** course first

Required Software

Docker Desktop

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

Minikube

https://github.com/kubernetes/minikube

kind

https://kind.sigs.k8s.io

kubeadm

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

Code Samples

https://github.com/DanWahlin/DockerAndKubernetesCourseCode

Look in the "samples" folder

Introduction

Module Overview

Kubernetes Deployments Overview

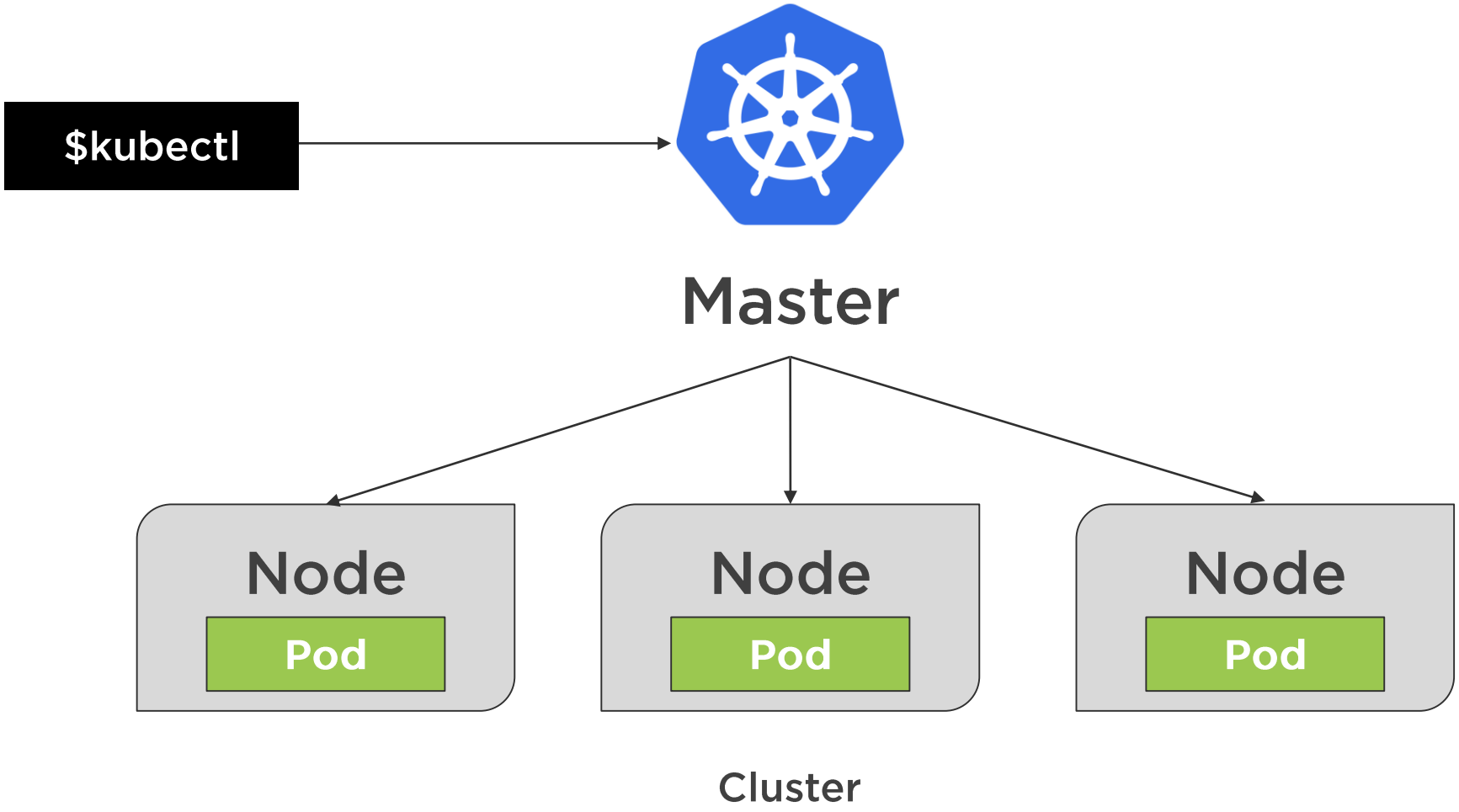
Creating an Initial Deployment

Kubernetes Deployments in Action

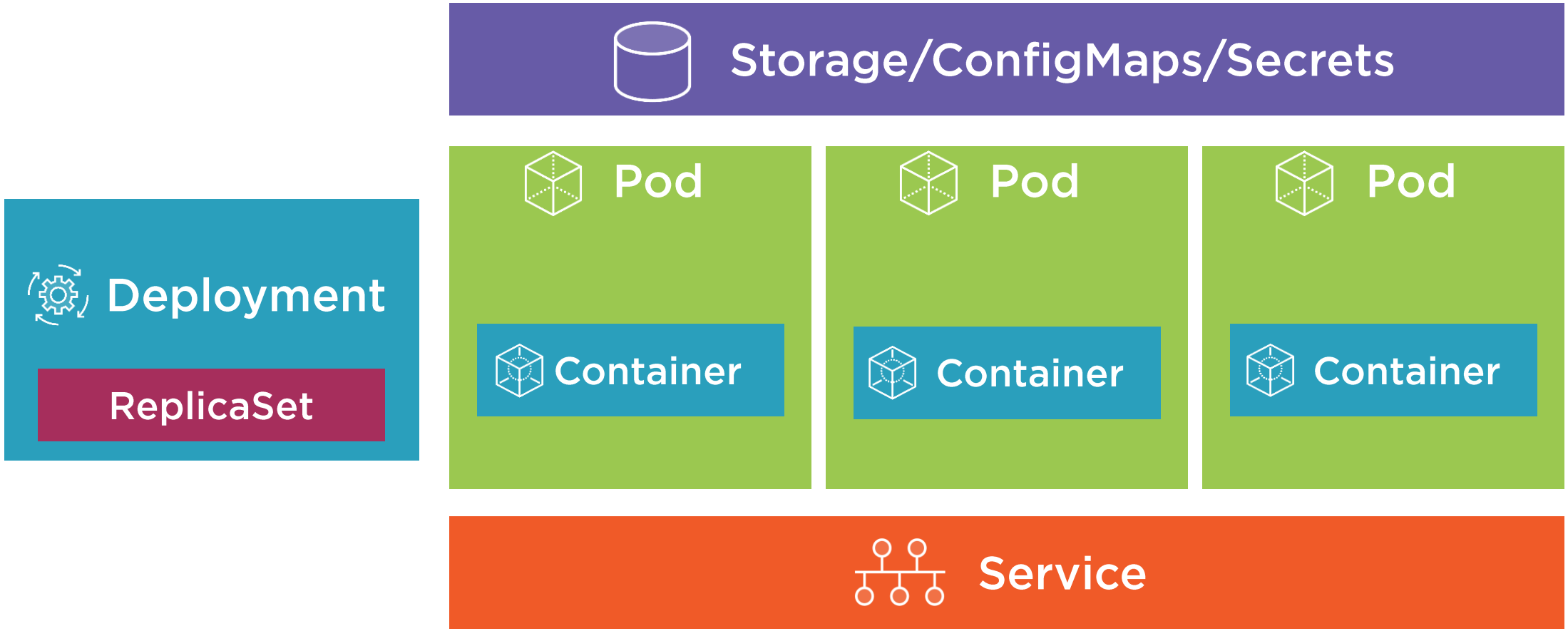
Kubernetes Deployment Options

Kubernetes Deployments Overview

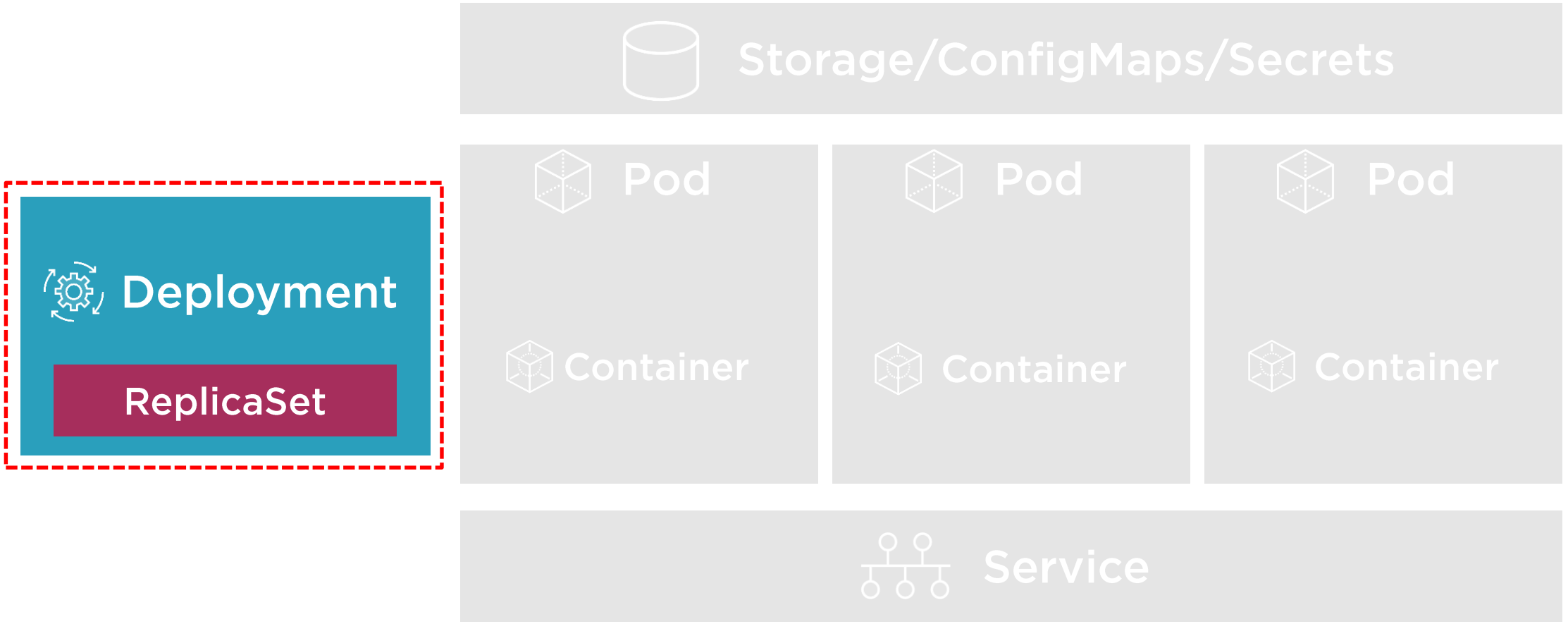
The Big Picture



Kubernetes Resources



Kubernetes Resources

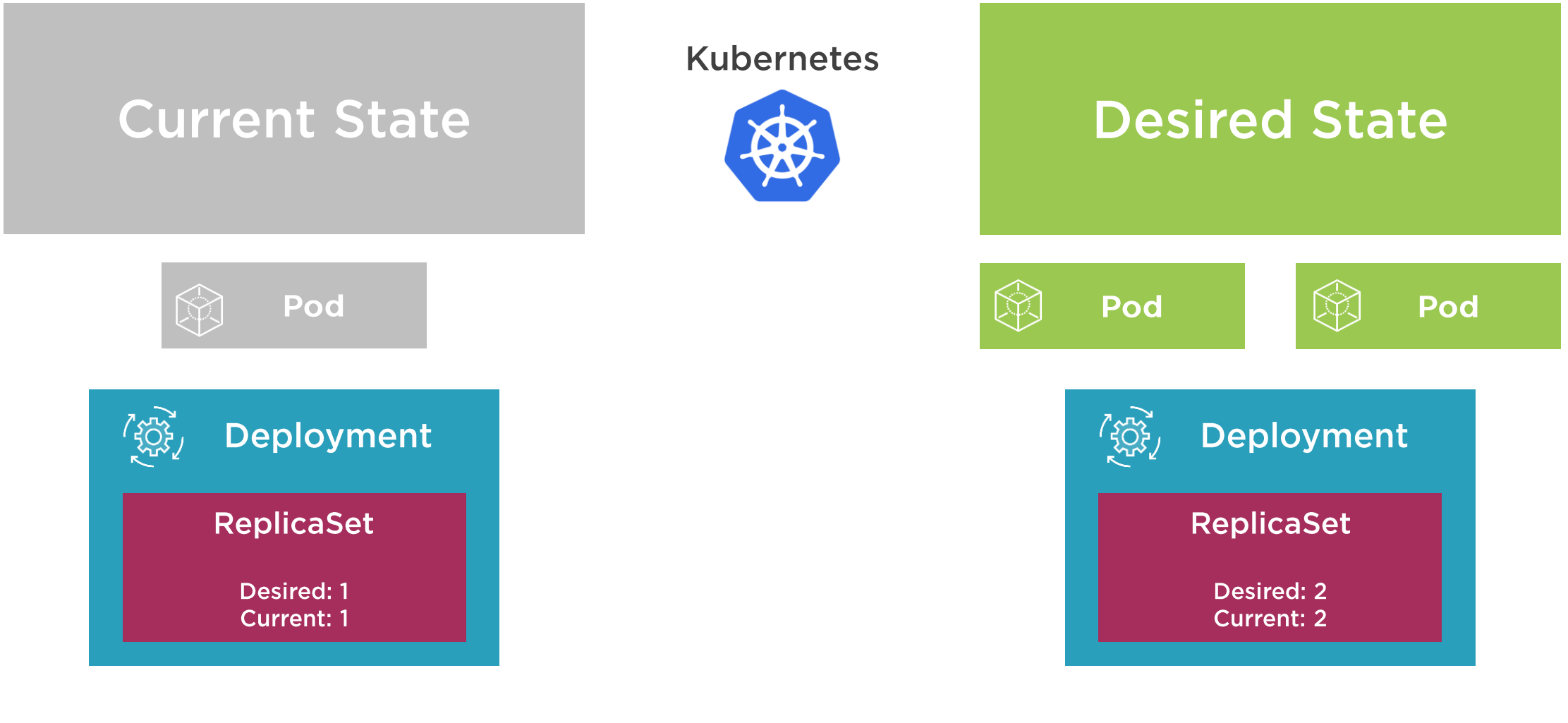


A ReplicaSet is a declarative way to manage Pods.

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


Pod
Deployment/ReplicaSet
Pod
Pod

Moving to a Desired State

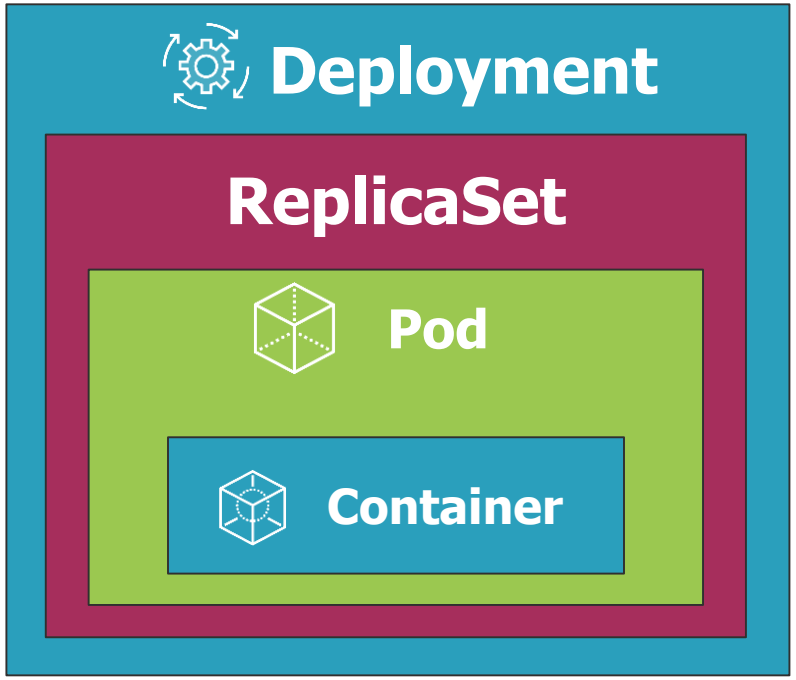


Creating an Initial Deployment

Creating a Deployment

Deployment

+ kubectl =



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)

Store current properties in resource's annotations

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

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
```

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

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

Scaling Pods Horizontally

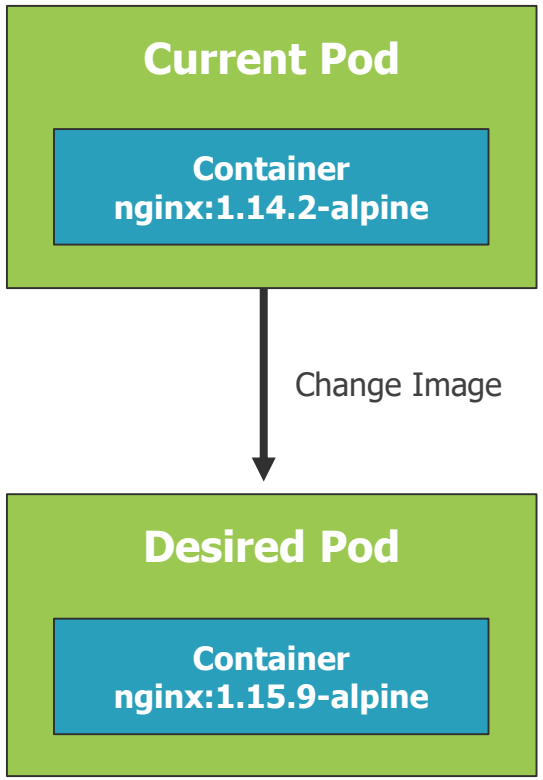
Update the YAML file (declarative) or use the kubectl scale command

```
spec:
  replicas: 5
  selector:
    tier: bizrules
```

Kubernetes Deployments in Action

Kubernetes Deployment Options

How Do You Update Existing Pods?



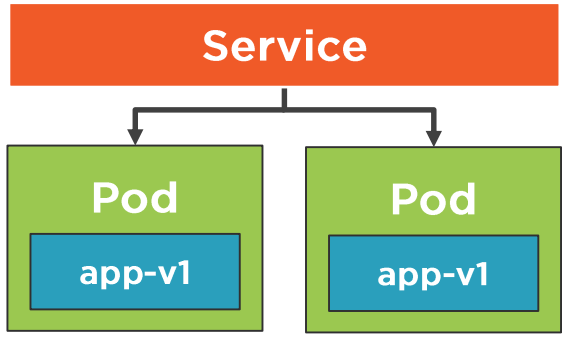
Delete all existing Pods and replace with new Pods? Leads to a short down-time.

Start new Pods and then delete old Pods? Need to be able to run two versions simultaneously.

Replace existing Pods one by one without impacting traffic to Pods?

Delete Existing Pods – Replace with New Pods

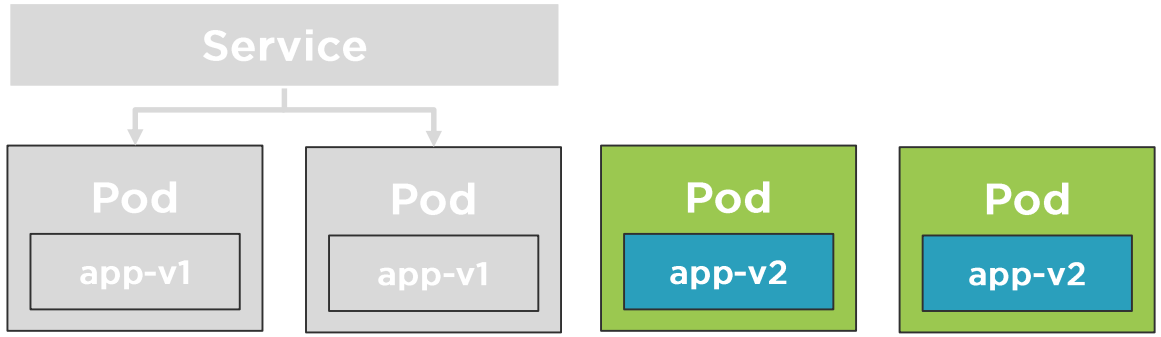
Initial Pod State



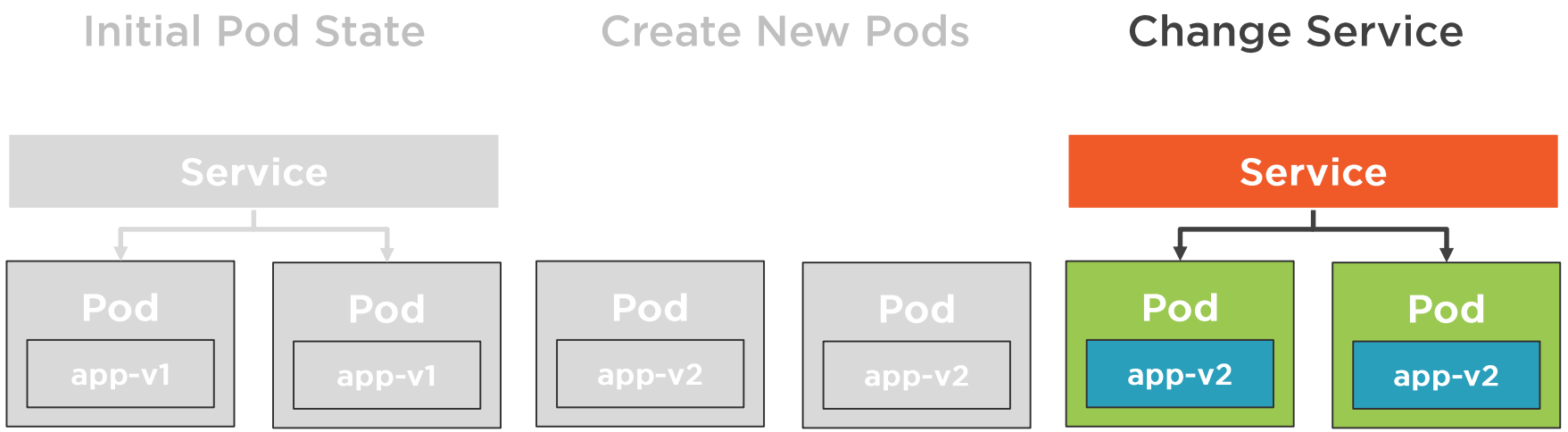
Delete Existing Pods – Replace with New Pods

Initial Pod State

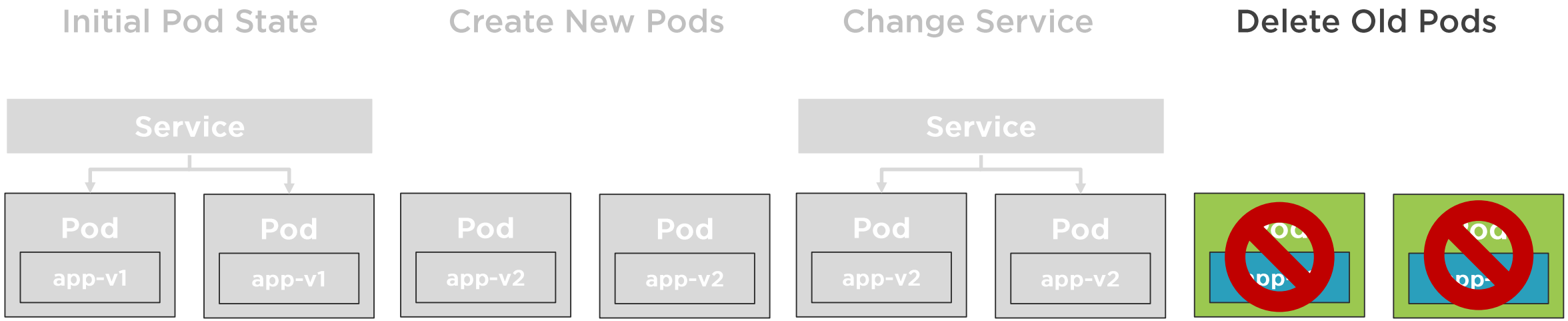
Create New Pods



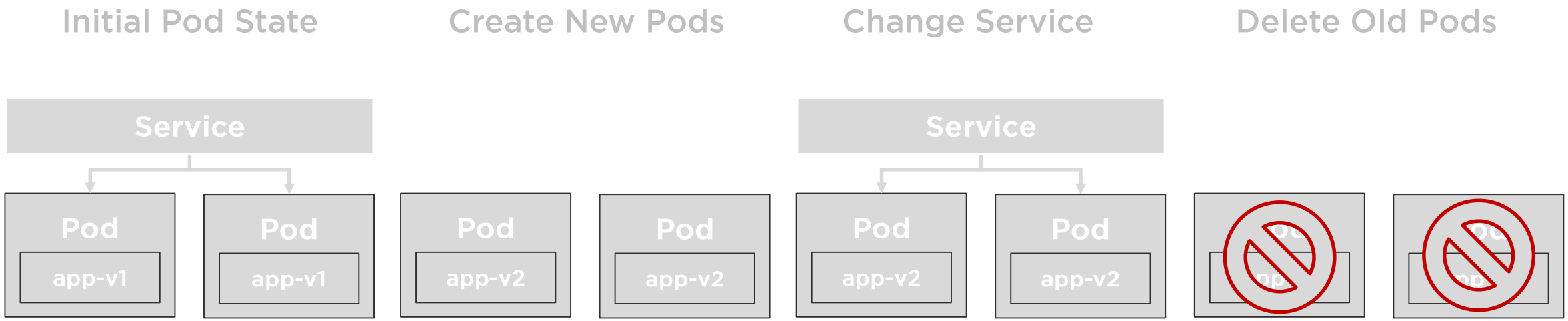
Delete Existing Pods – Replace with New Pods



Delete Existing Pods – Replace with New Pods

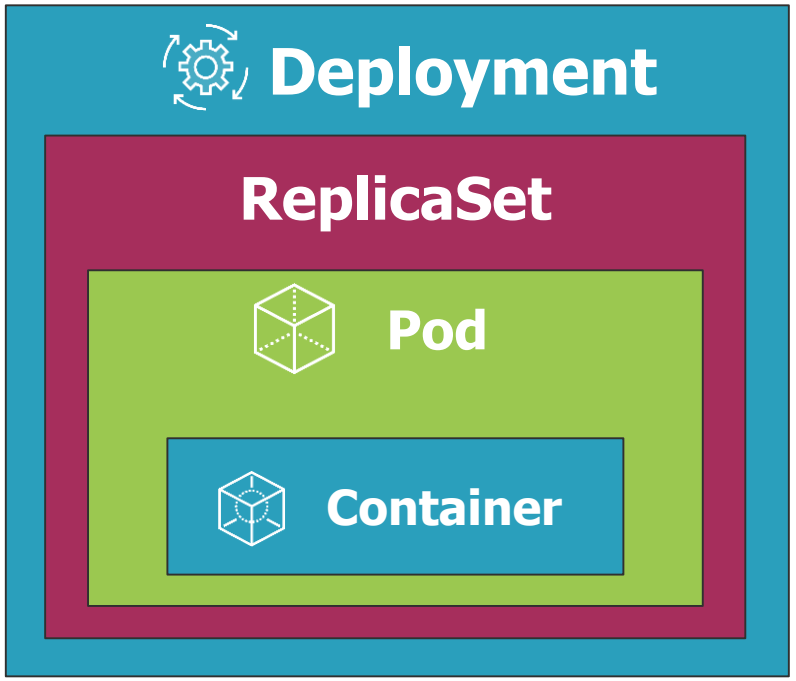


Delete Existing Pods – Replace with New Pods



Can your servers run the new and old Pods at the same time?

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

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

Summary

Deployments are a key resource provided by Kubernetes

Deployments rely on ReplicaSets to schedule and manage Pods

Kubernetes supports Zero-downtime deployments out of the box

Several Deployment options exist:

- Zero-downtime
- Rolling Updates
- Canary
- Blue-Green

Performing Rolling Update Deployments

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

Understanding Rolling Update Deployments

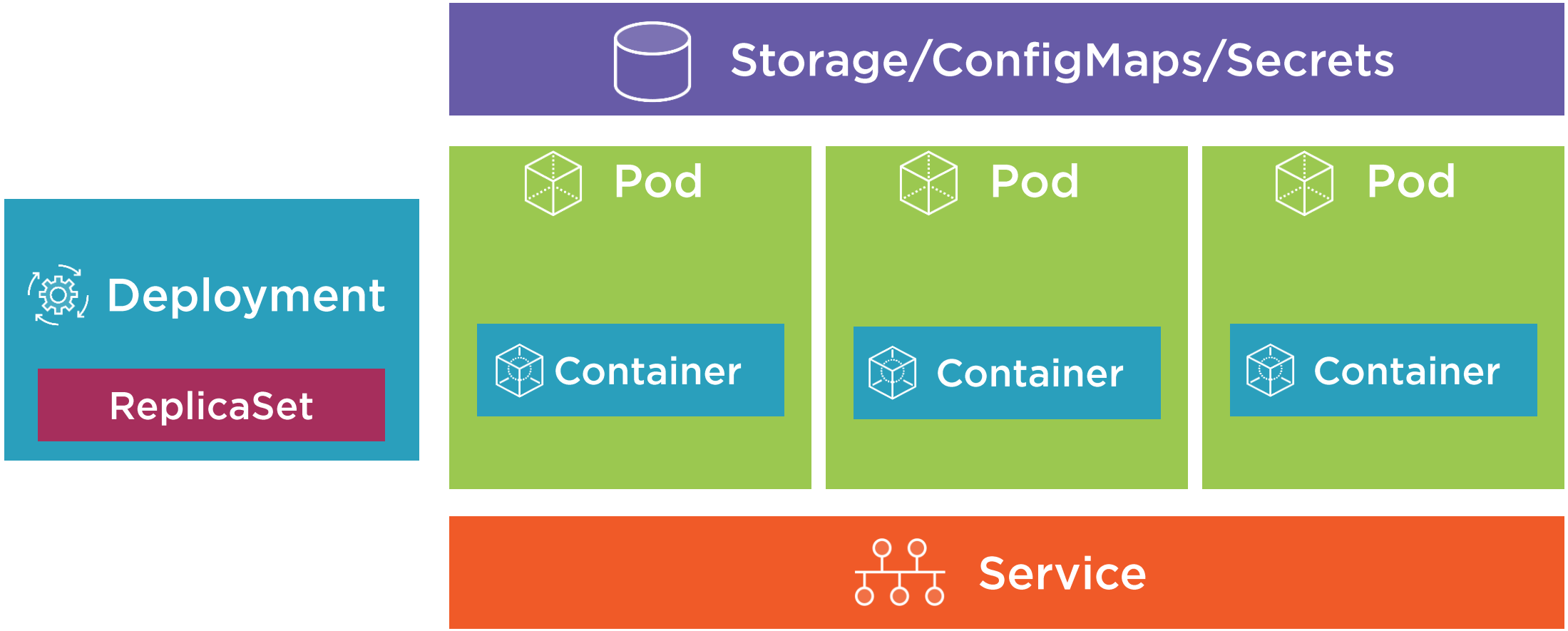
Creating a Rolling Update Deployment

Rolling Update Deployment in Action

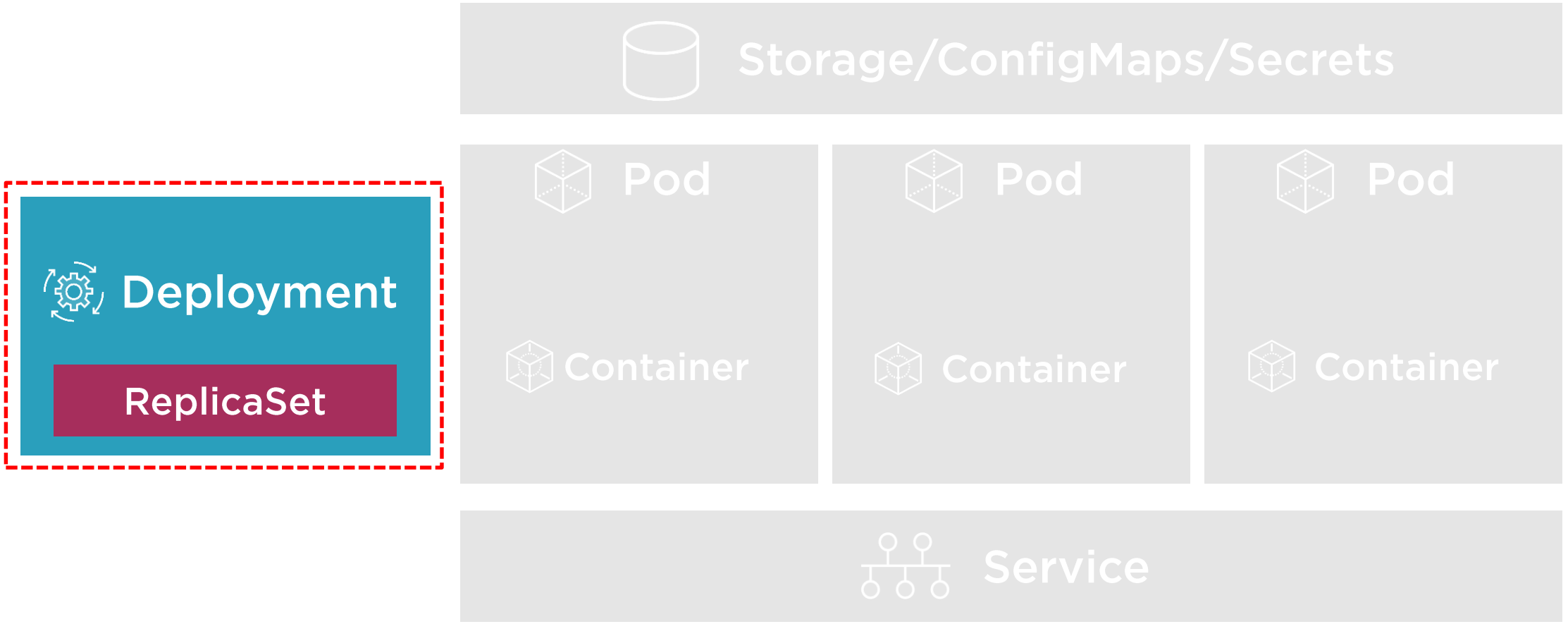
Rolling Back Deployments

Rolling Back Deployments in Action

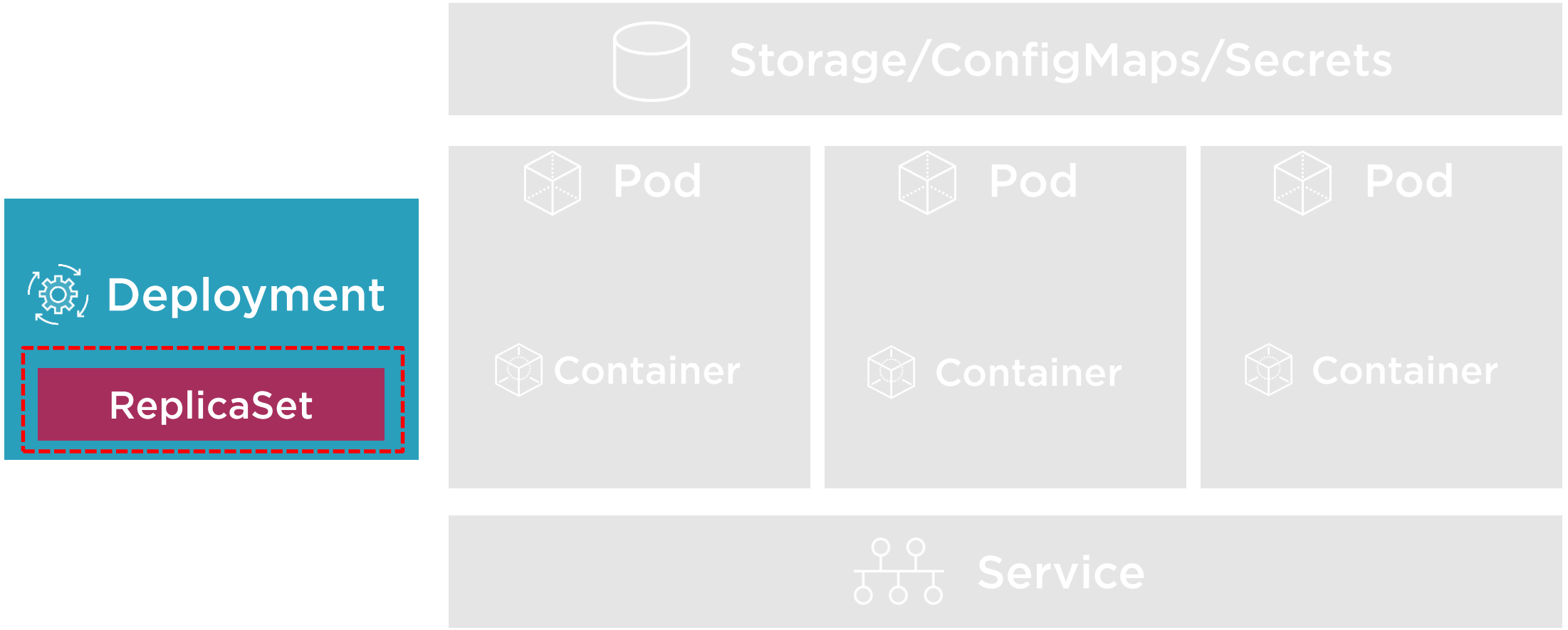
Kubernetes Resources



Kubernetes Resources



Kubernetes Resources

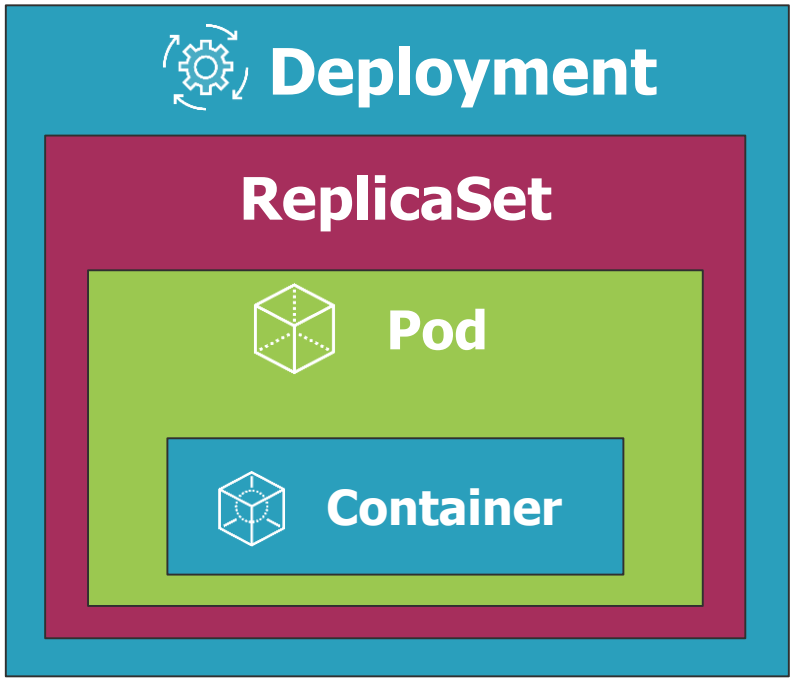


Understanding Rolling Update Deployments

"Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones."

Kubernetes Documentation

Rolling Update Deployments



ReplicaSets increase new Pods while decreasing old Pods

Service handles load balancing traffic to available Pods

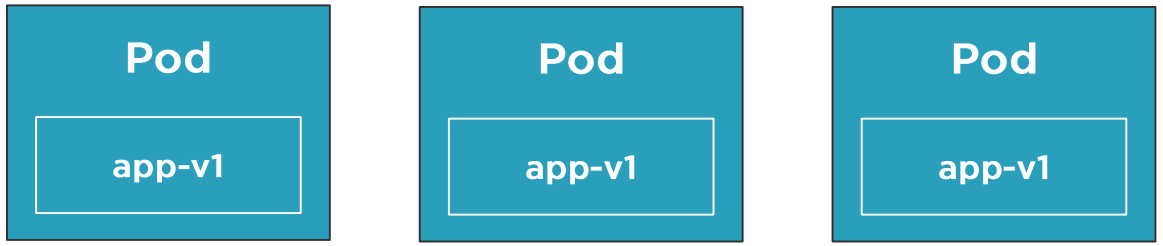
New Pods only scheduled on available Nodes

Deployments support two strategy options:

- Rolling Update (default and our focus here)
- Recreate (can result in down-time)

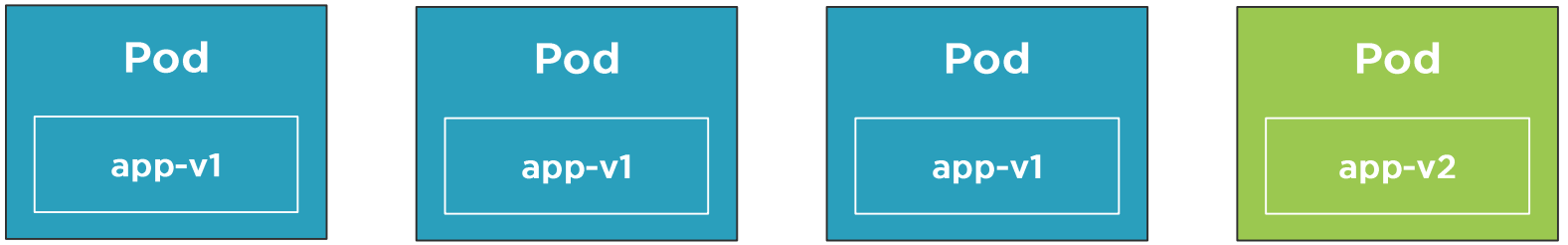
Rolling Update Deployments

Initial Pod State



Rolling Update Deployments

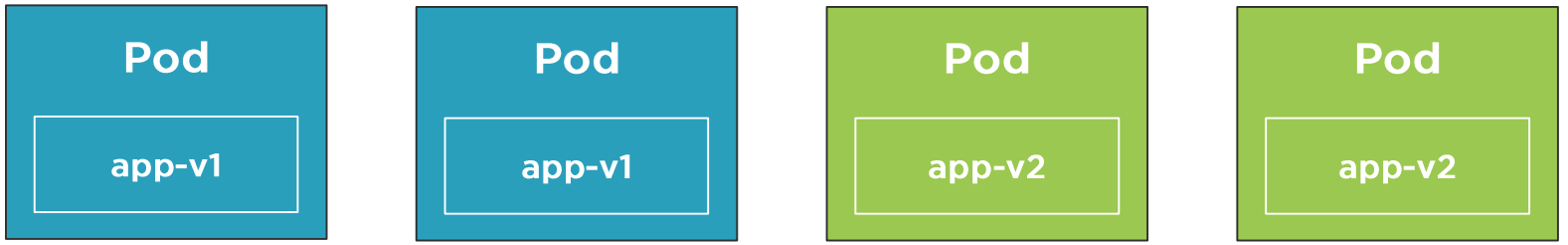
Rollout New Pod



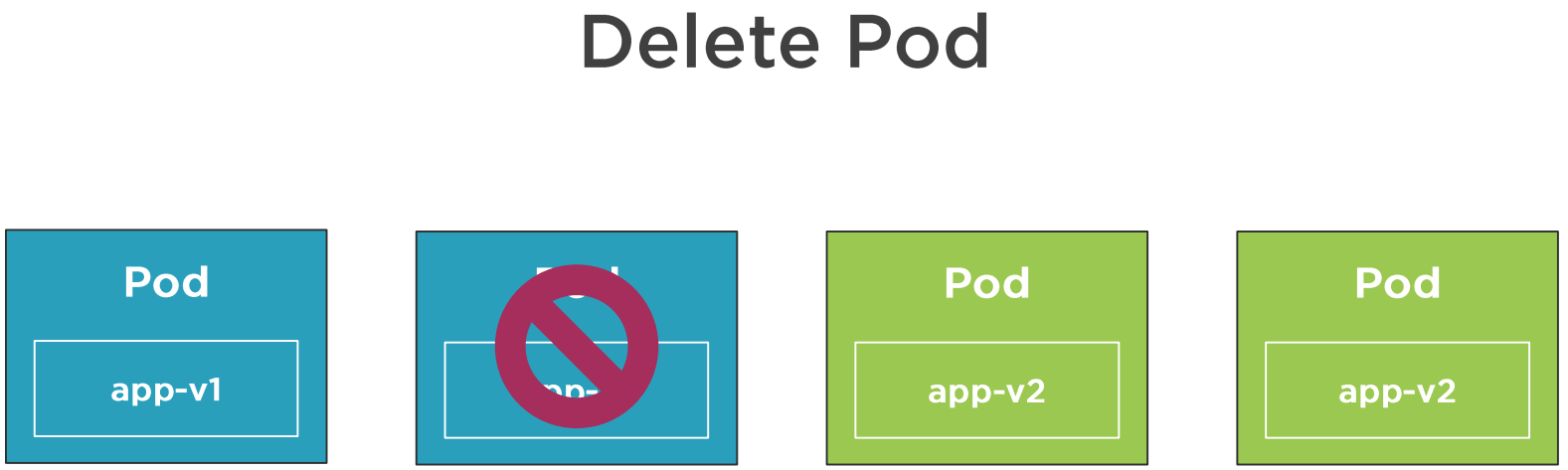
Rolling Update Deployments

Rolling Update Deployments

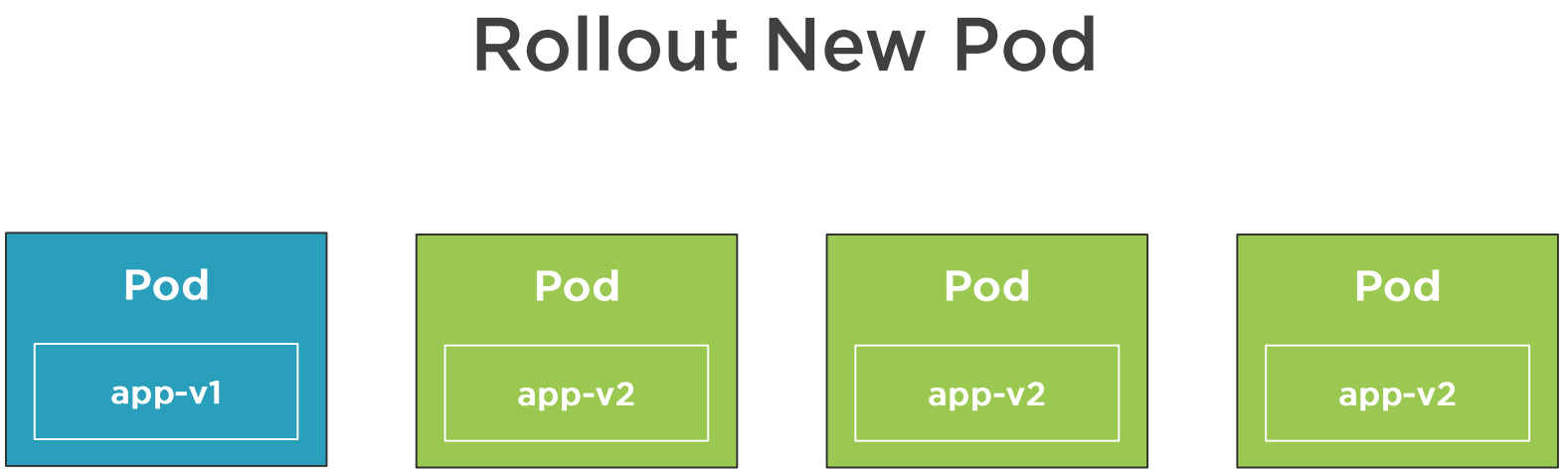
Rollout New Pod



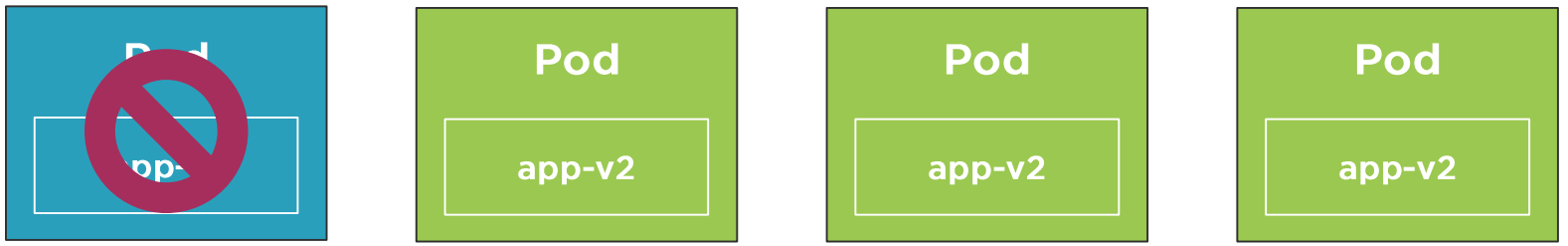
Rolling Update Deployments



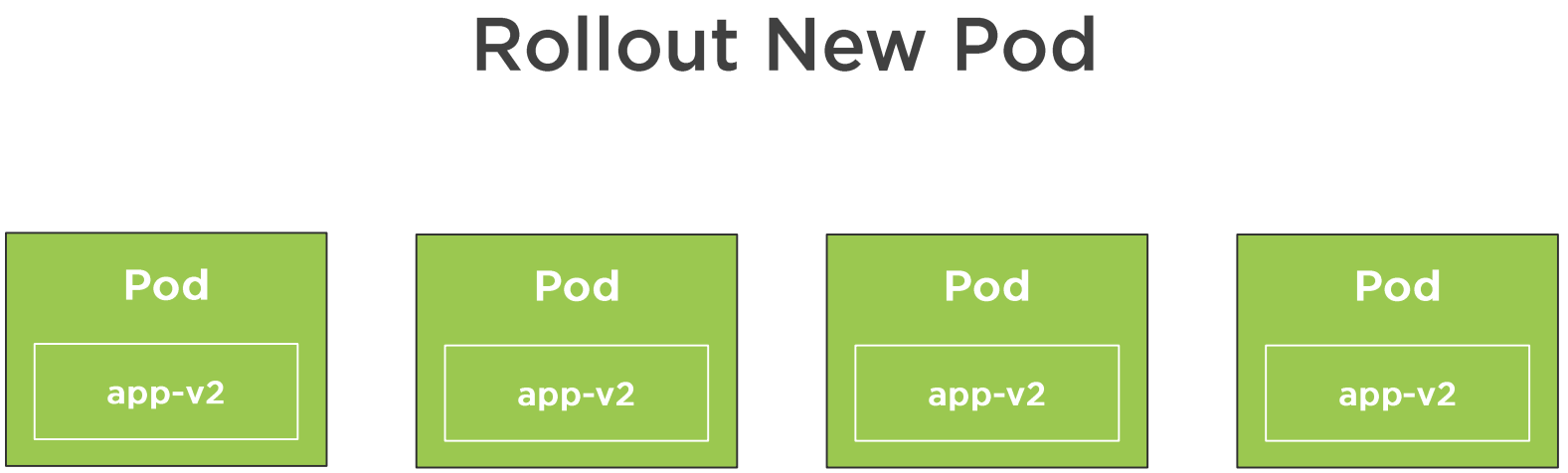
Rolling Update Deployments



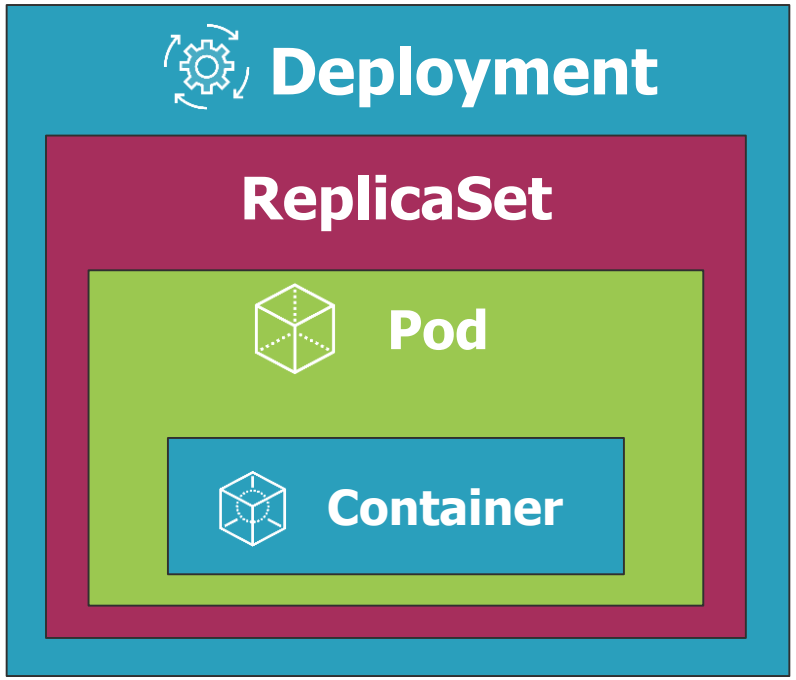
Rolling Update Deployments



Rolling Update Deployments



Deployments and Replicasets



Creating a Rolling Update Deployment

Defining a Rolling Update Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
  minReadySeconds: 1
  progressDeadlineSeconds: 60
  revisionHistoryLimit: 5
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1

  ...
```

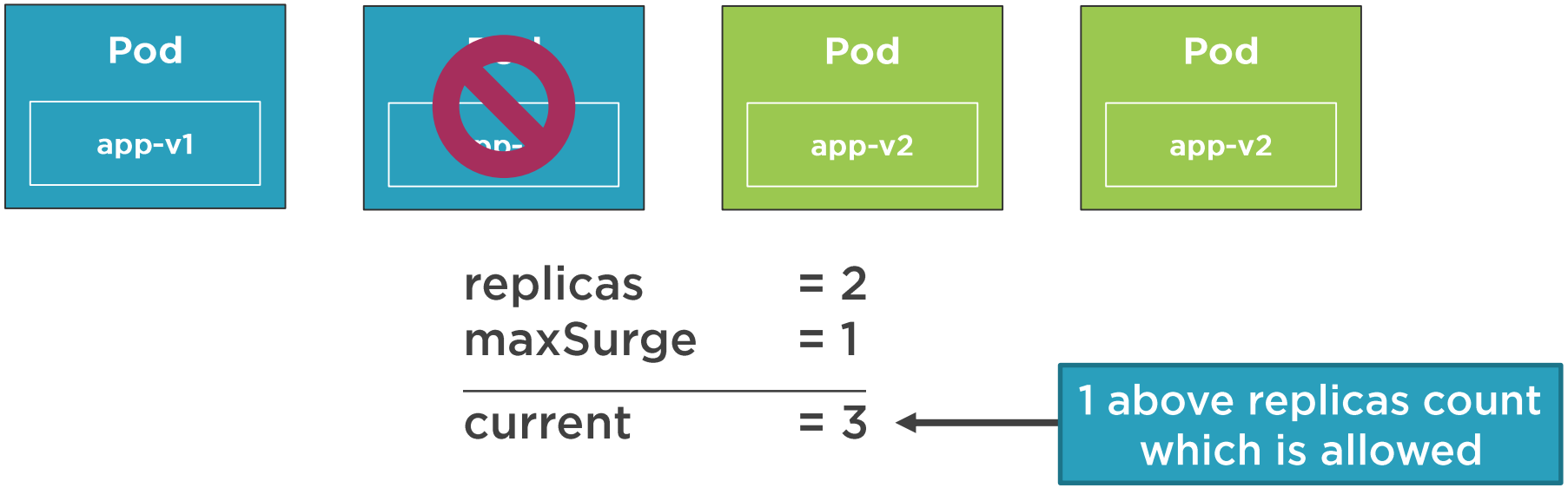
- Number of Pod replicas
- Seconds new Pod should be ready to be considered healthy (0)
- Seconds to wait before reporting stalled Deployment
- Number of ReplicaSets that can be rolled back (10)

- RollingUpdate (default) or Recreate strategy

- Max Pods that can exceed the replicas count (25%)
- Max Pods that are not operational (25%)

Understanding maxSurge

How many Pods can be added above the replicas count during the rolling update?



Understanding maxUnavailable

How many of the existing Pods can be made unavailable during a rolling update?

Creating the Deployment

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

```
# Create initial deployment
kubectl create -f file.deployment.yml --save-config --record
```

Record the command in the Deployment revision history

Save configuration in resource's annotations

Creating or Modifying a Deployment

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


```
# Create initial deployment
kubectl apply -f file.deployment.yml --record
```

Checking the Deployment Status

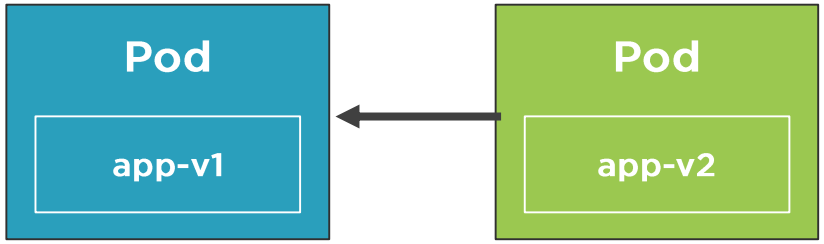
The kubectl rollout status command can be used to get information about a specific Deployment

```
# Get information about a Deployment
kubectl rollout status deployment [deployment-name]
```

Rolling Update Deployment in Action

Rolling Back Deployments

Rolling Back Deployments



Rolling update revisions can be tracked using --record

If a Deployment has issues, a new Deployment can be applied, or you can revert to a previous revision

Several kubectl commands can be used for rollbacks:

- kubectl rollout status
- kubectl rollout history
- kubectl rollout undo

Checking Deployment History

The kubectl rollout history command can be used to view history of a Deployment

```
# Get information about a Deployment
kubectl rollout history deployment [deployment-name]

# Get information about a Deployment
kubectl rollout history deployment [deployment-name] --revision=2
```

Rolling Back a Deployment

Use the kubectl rollout undo command to rollback to a specific Deployment revision

```
# Check status
kubectl rollout status -f file.deployment.yml

# Rollback a Deployment
kubectl rollout undo -f file.deployment.yml

# Rollback to a specific revision
kubectl rollout undo deployment [deployment-name] --to-revision=2
```

Rolling Back Deployments in Action

Summary

Rolling updates are the default Deployment strategy used by Kubernetes

Ensures zero-downtime during a Deployment

Maximum and minimum Pods available during a Deployment can be defined

Deployments can be recorded and stored in history using **--record**

Deployments can be rolled back to a specific revision

Performing Canary Deployments

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

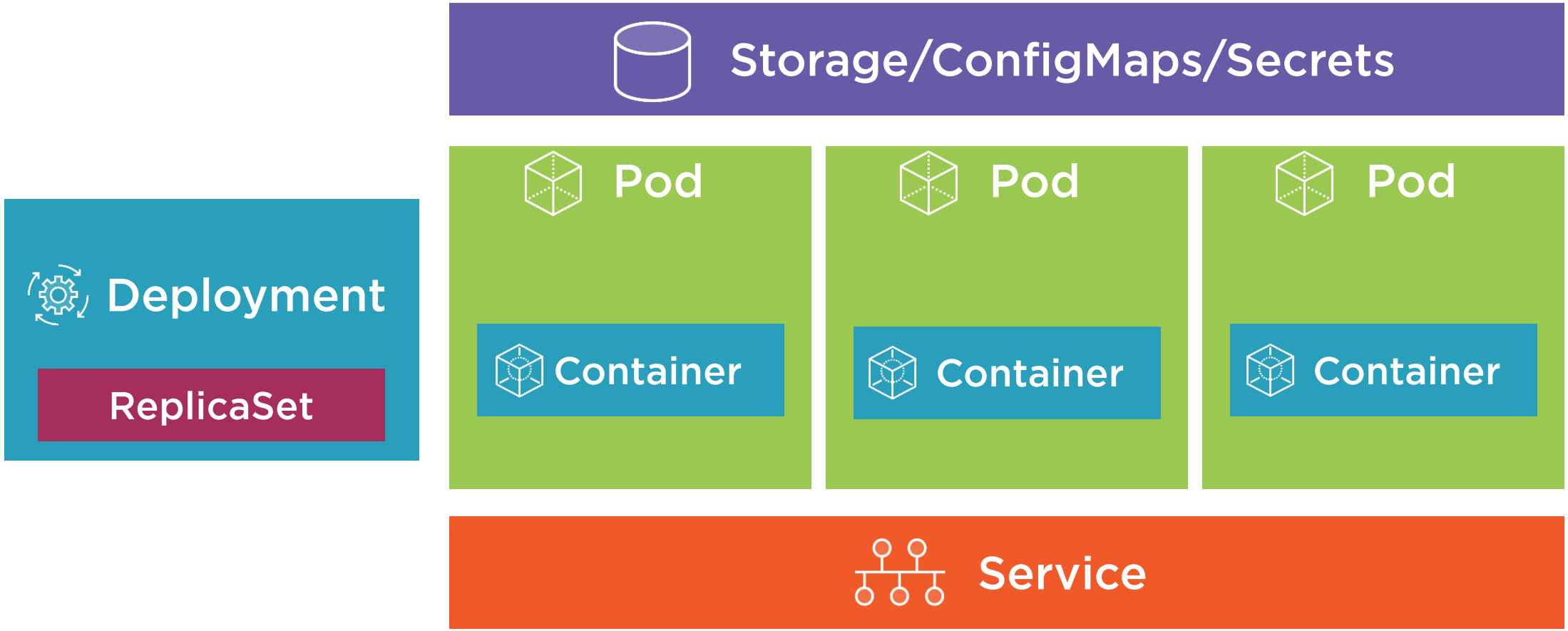
Module Overview

Understanding Canary Deployments

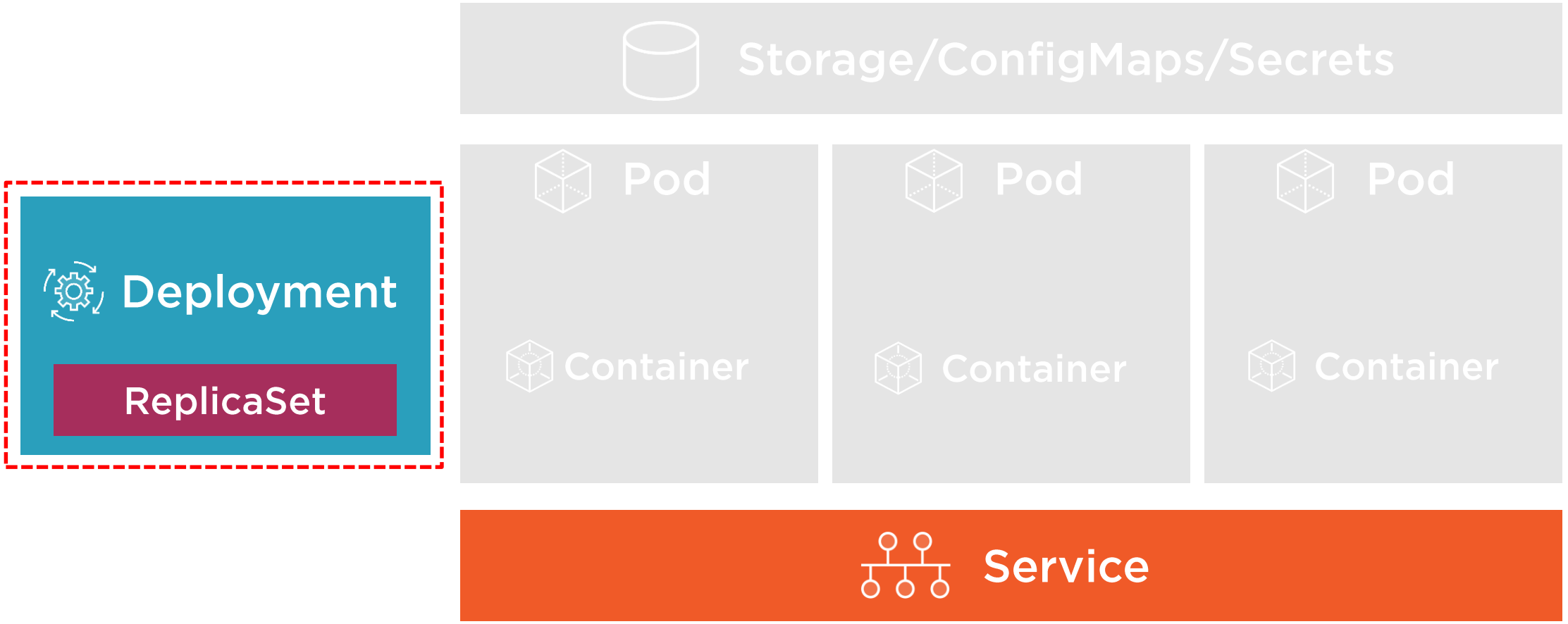
Creating a Canary Deployment

Canary Deployments in Action

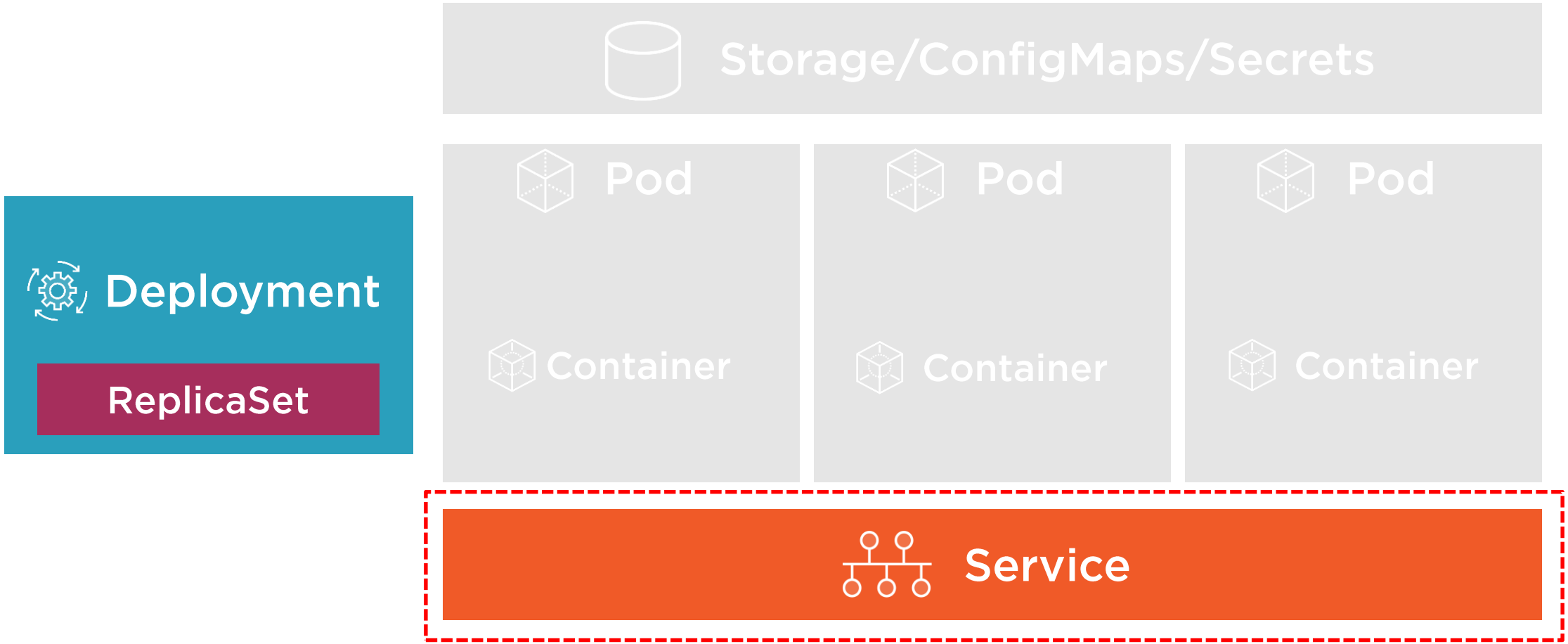
Kubernetes Resources



Kubernetes Resources



Kubernetes Resources



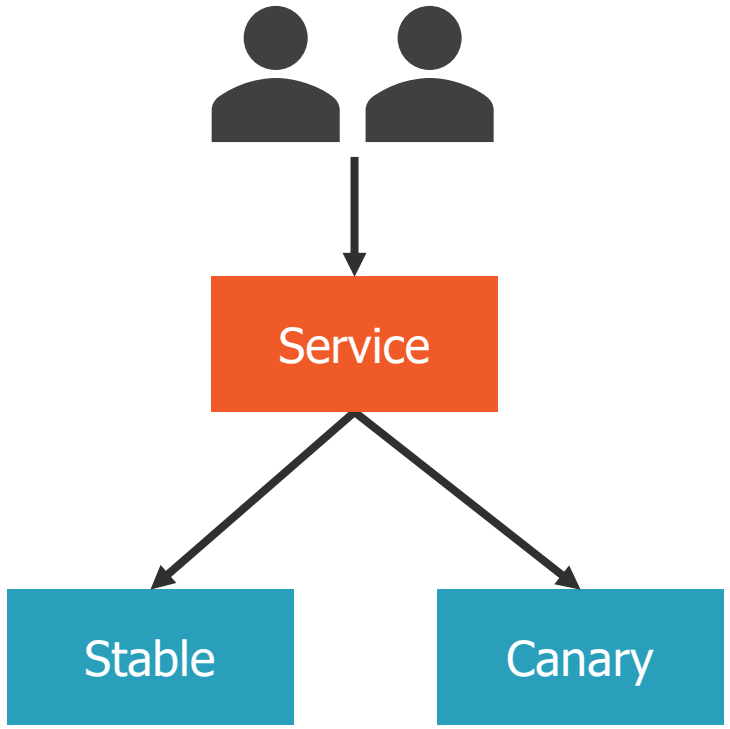
Understanding Canary Deployments

Wouldn't it be nice to rollout a new Deployment but only route a small percentage of the overall traffic to it to ensure it's working properly?

"Canary deployment strategy involves deploying new versions of applications next to stable production versions to see how the canary version compares against the baseline before promoting or rejecting the deployment."

~ https://docs.microsoft.com

Canary Deployments



Strategy for checking the viability of a deployment

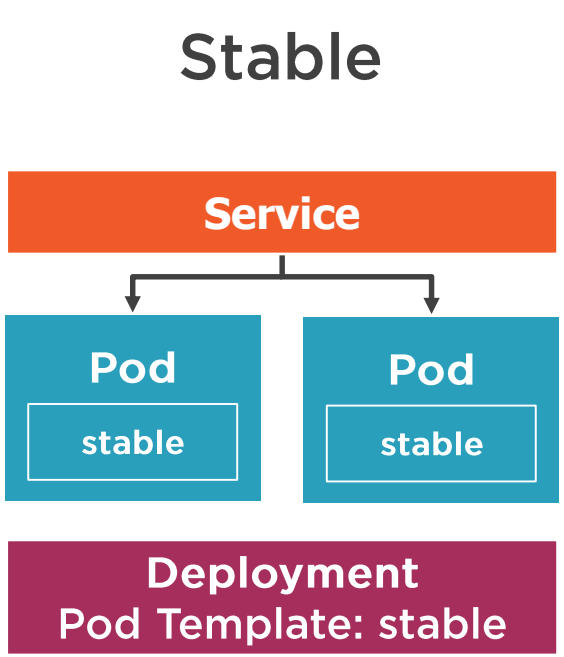
Run two identical production environments at the same time

Canary Deployment runs alongside the existing stable Deployment

Canary Deployment is setup to receive minimal traffic

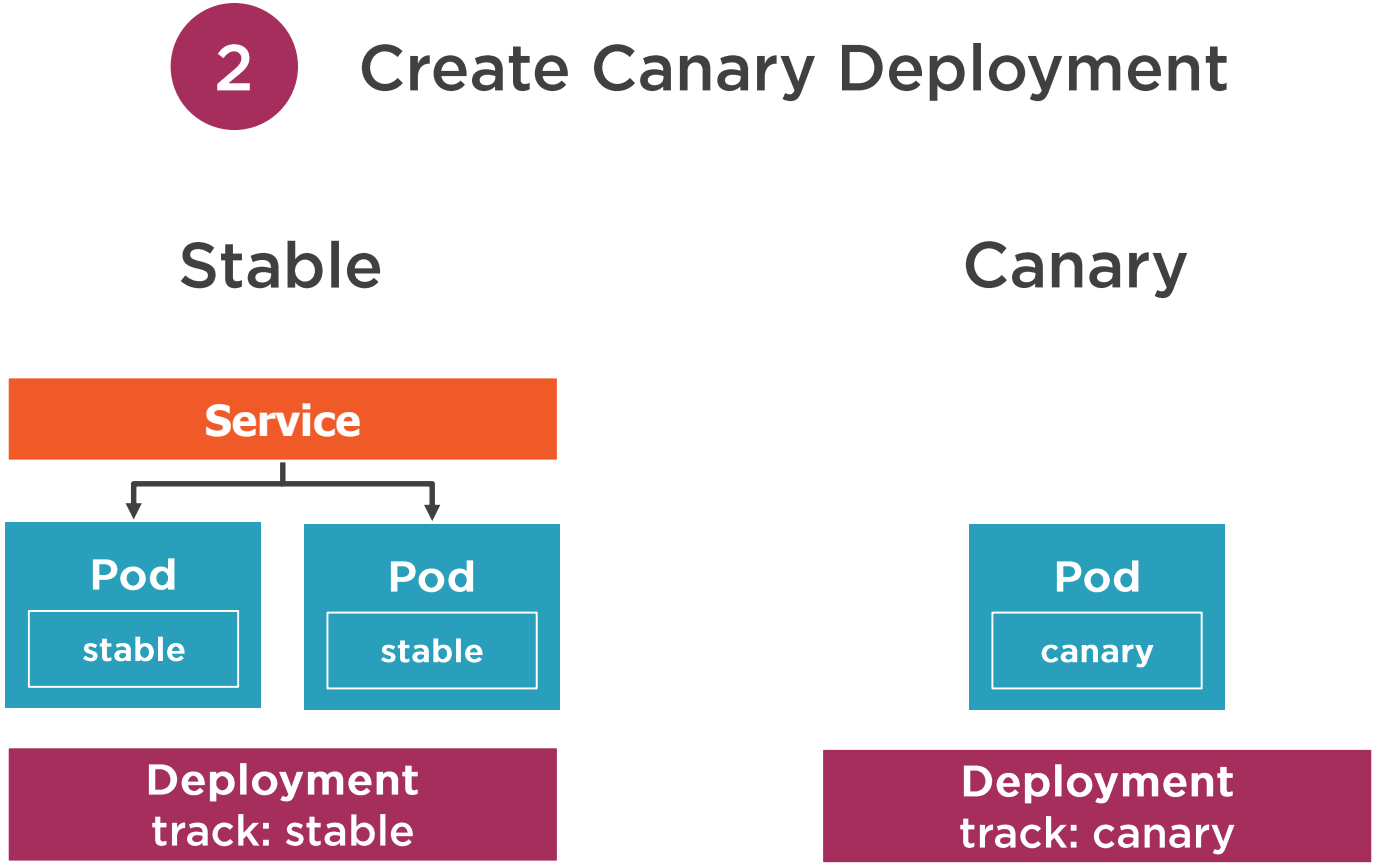
Canary Deployments

1 **Create Stable Deployment and Service**



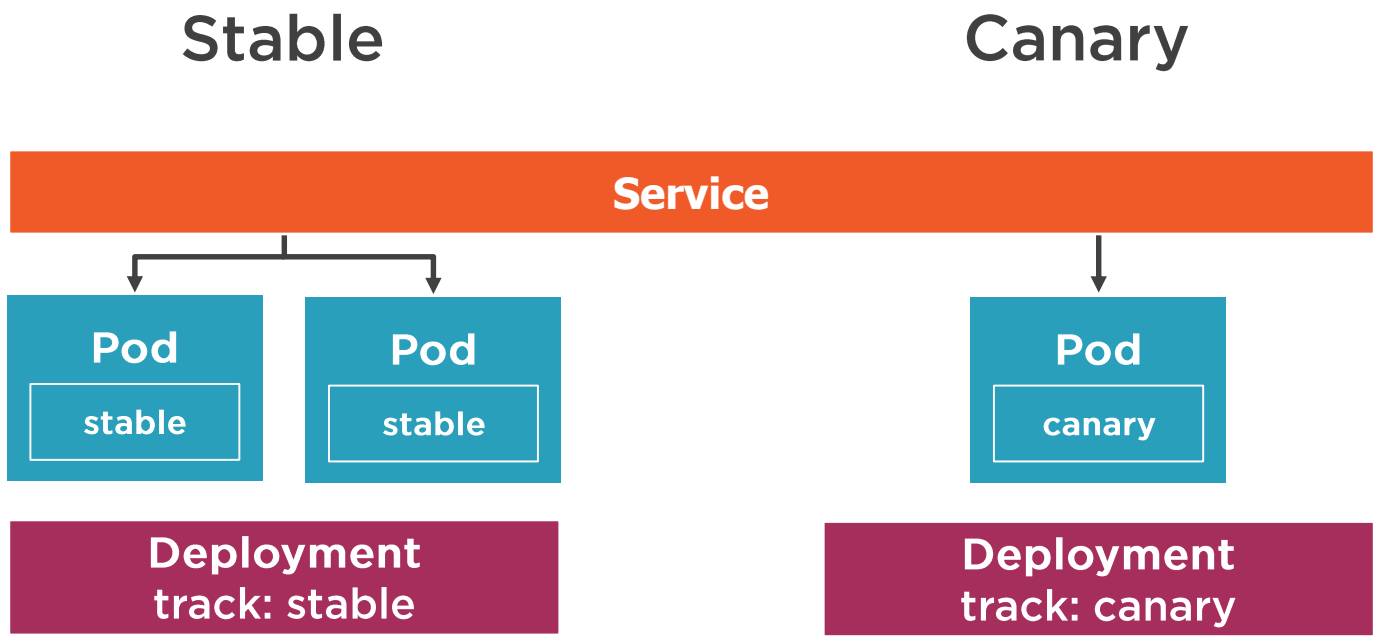
Canary Deployments

2 Create Canary Deployment



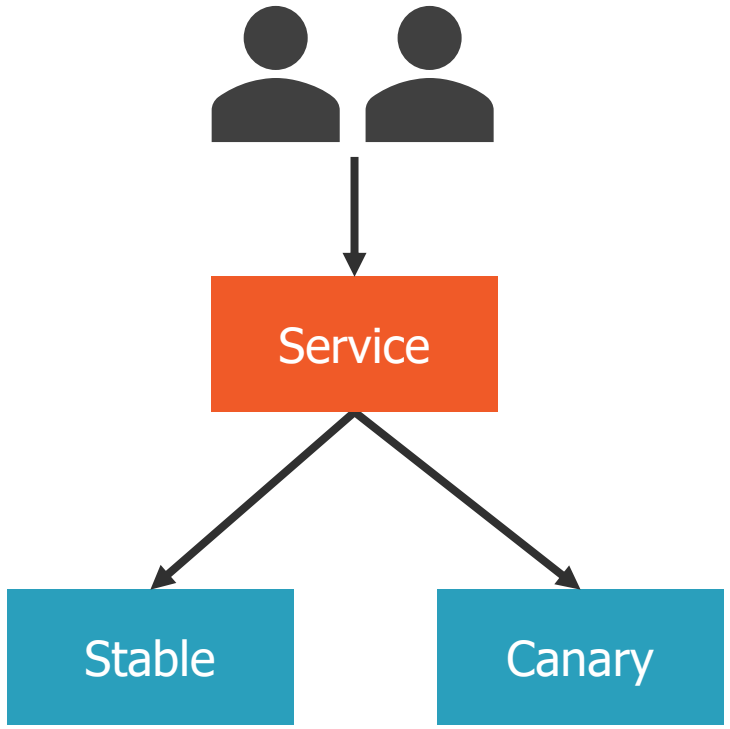
Canary Deployments

3 Service adds Canary Pod(s) and traffic is routed



Creating a Canary Deployment

Canary Deployment Resources

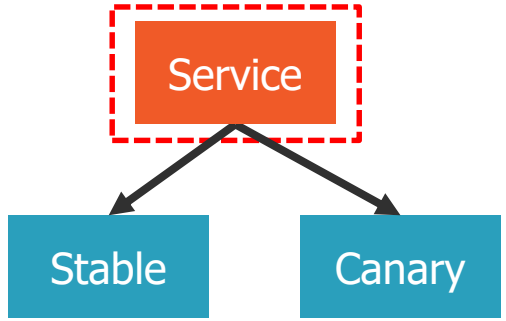


A Canary Deployment involves 3 main Kubernetes resources:

- Service
- Stable Deployment
- Canary Deployment

Defining a Service

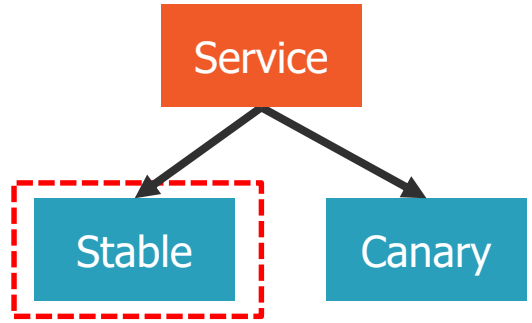
```
kind: Service
apiVersion: v1
metadata:
  name: stable-service
  labels:
    app: aspnetcore
spec:
  type: LoadBalancer
  selector:
    app: aspnetcore
  ports:
    - port: 80
      targetPort: 80
```



◂ Pod Label to select for Service

Defining a Stable Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: stable-deployment
spec:
  replicas: 4
  selector:
    matchLabels:
      app: aspnetcore
      track: stable
  template:
    metadata:
      labels:
        app: aspnetcore
        track: stable
    spec:
      containers:
      - name: stable-app
        image: stable-app
```

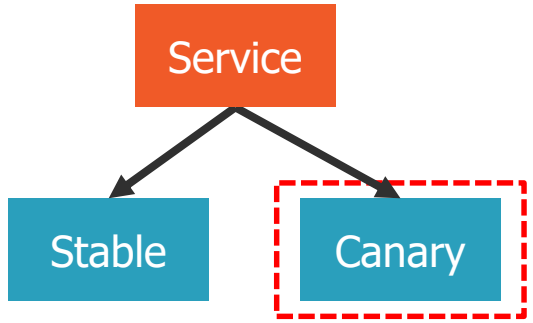


- Create stable replicas

- Pod labels (recall that **app:aspnetcore** is targeted by the Service)

Defining a Canary Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: canary-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: aspnetcore
      track: canary
  template:
    metadata:
      labels:
        app: aspnetcore
        track: canary
    spec:
      containers:
      - name: canary-app
        image: canary-app
```



- Create canary replicas (25% of stable in this example)

- Pod labels (recall that **app:aspnetcore** is targeted by the Service)

Creating the Stable and Canary Resources

Use kubectl create or kubectl apply commands to create the Service, Stable Deployment, and Canary Deployment

```
# Create Service, Stable Deployment, and Canary Deployment
kubectl create –f [folder-name] --save-config --record
```

Canary Deployments in Action

Summary

Canary Deployments allows a new version to be deployed next to a stable version

Configured to only handle a small percentage of the traffic initially

Once the Canary Deployment is verified it can be scaled up and the existing stable Deployment can be scaled down

Performing Blue-Green Deployments

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

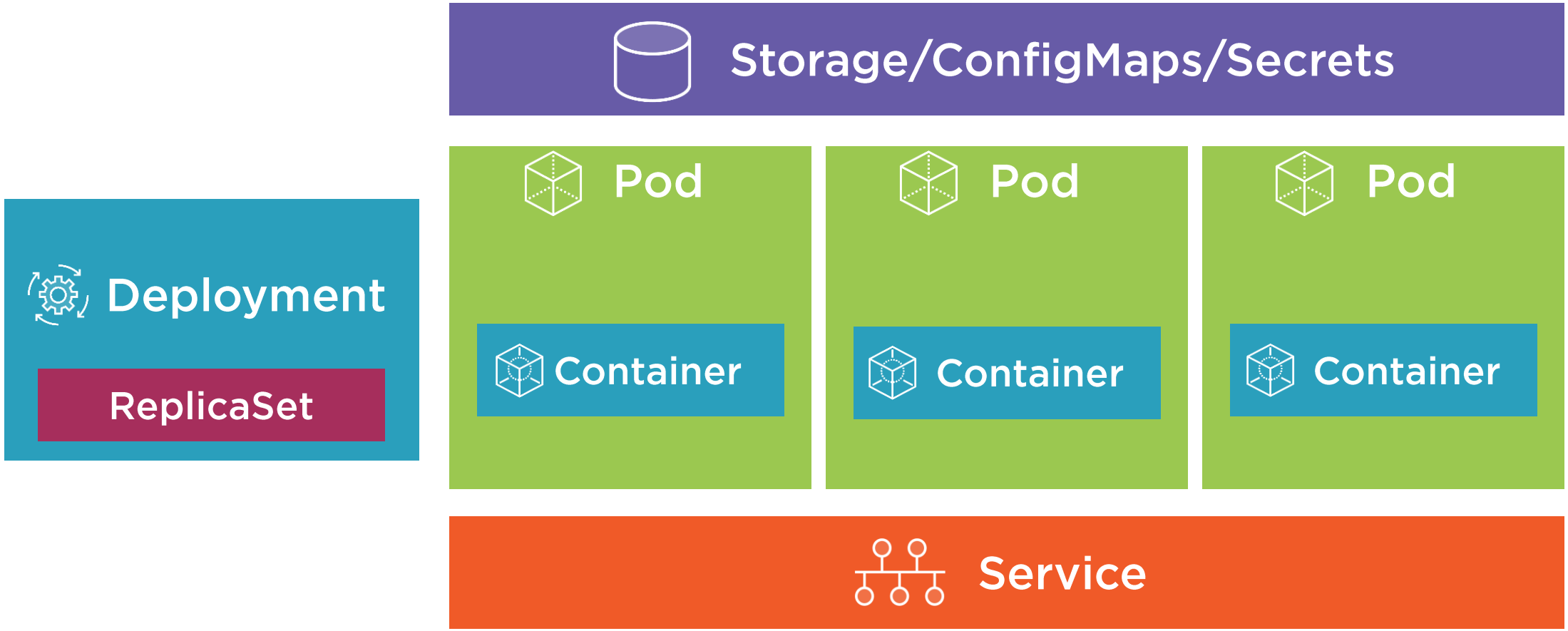
Module Overview

Understanding Blue-Green Deployments

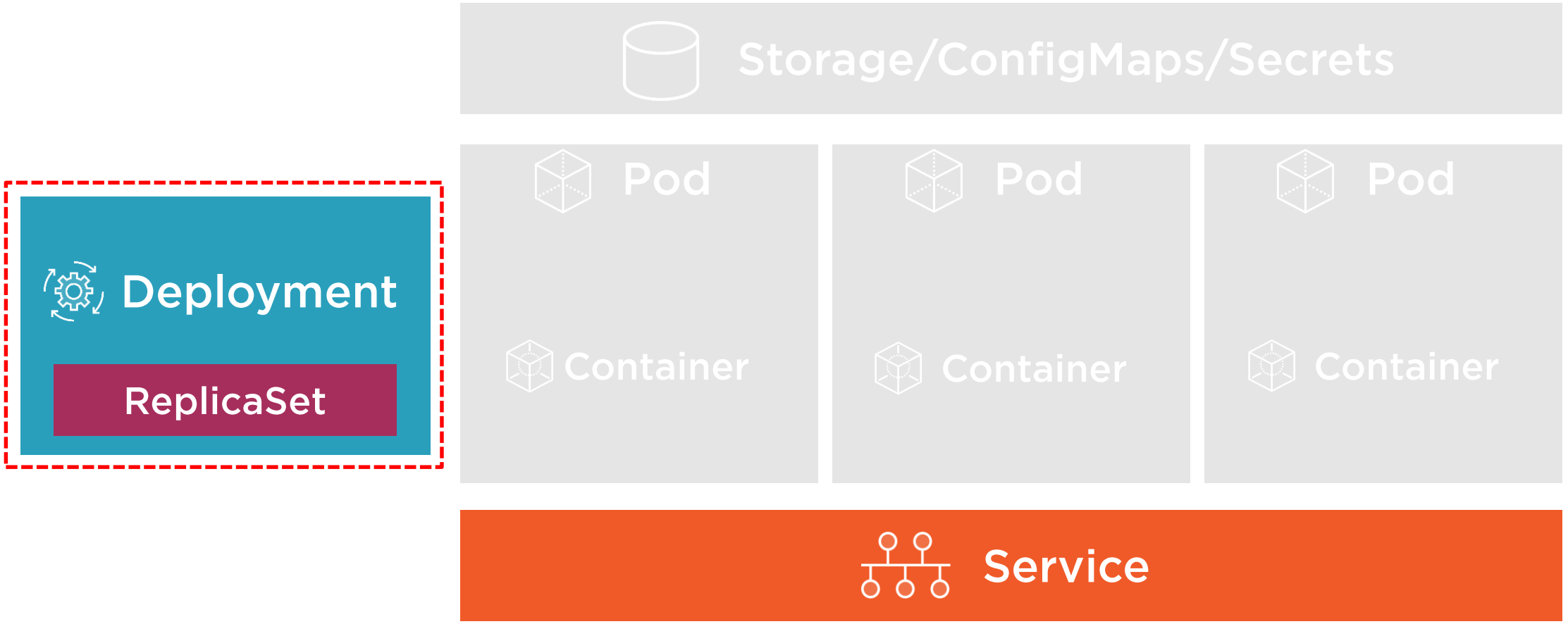
Creating a Blue-Green Deployment

Blue-Green Deployments in Action

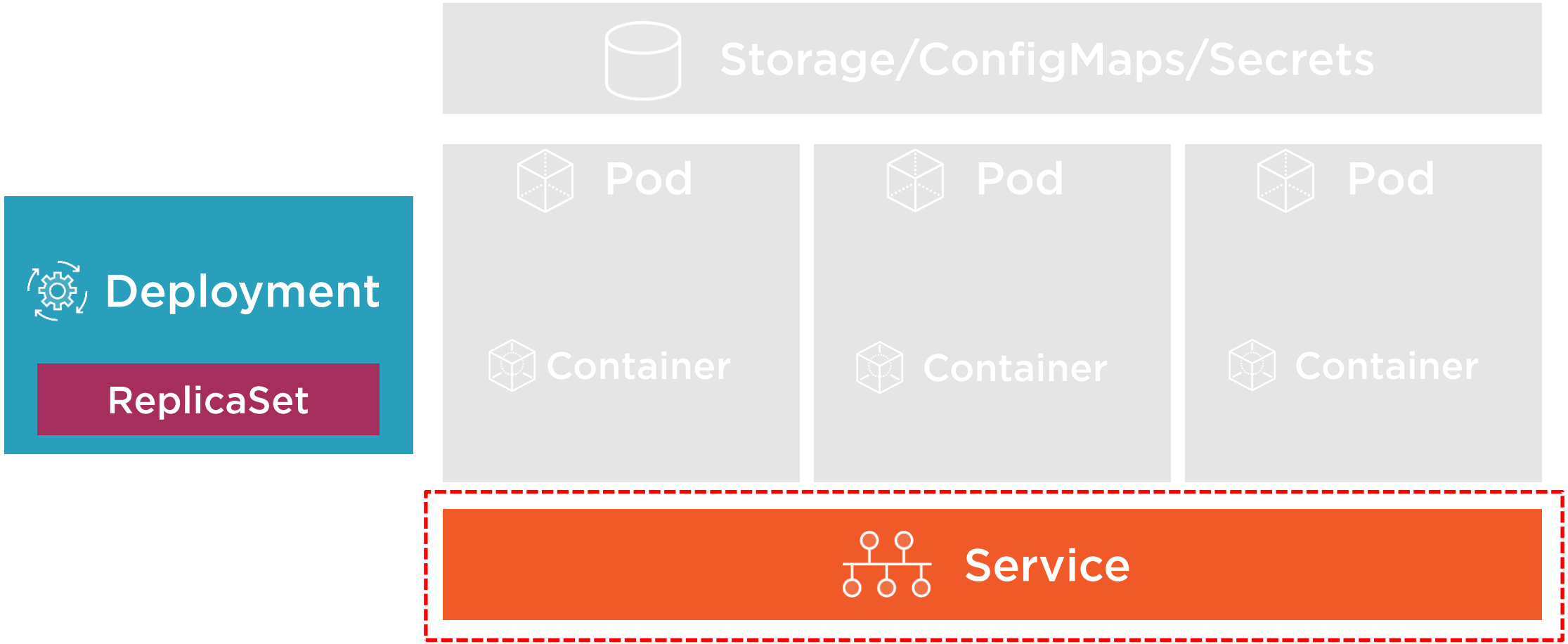
Kubernetes Resources



Kubernetes Resources



Kubernetes Resources



Understanding Blue-Green Deployments

Have you ever deployed an application to production and experienced problems? (don't laugh too much ☺)

"A blue/green deployment is a change management strategy for releasing software code."

~ TechTarget.com

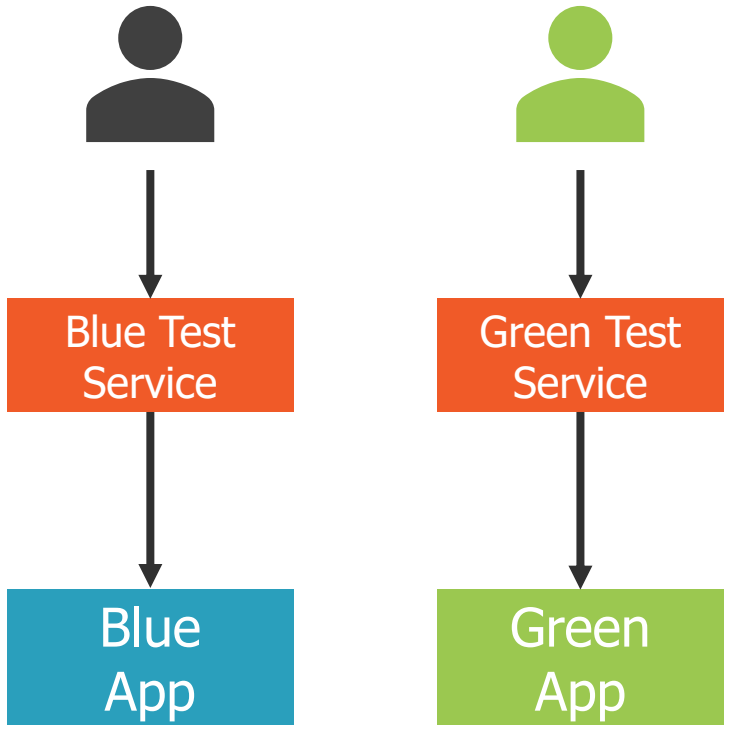
"A blue/green deployment is a change management strategy for releasing software code. Blue/green deployments, which may also be referred to as A/B deployments require two identical hardware environments that are configured exactly the same way. While one environment is active and serving end users, the other environment remains idle."

~ TechTarget.com

"A blue/green deployment is a change management strategy for releasing software code. Blue/green deployments, which may also be referred to as A/B deployments require two identical hardware environments that are configured exactly the same way. While one environment is active and serving end users, the other environment remains idle."

~ TechTarget.com

Blue-Green Deployments



Strategy for checking the viability of a deployment before it's publicly available

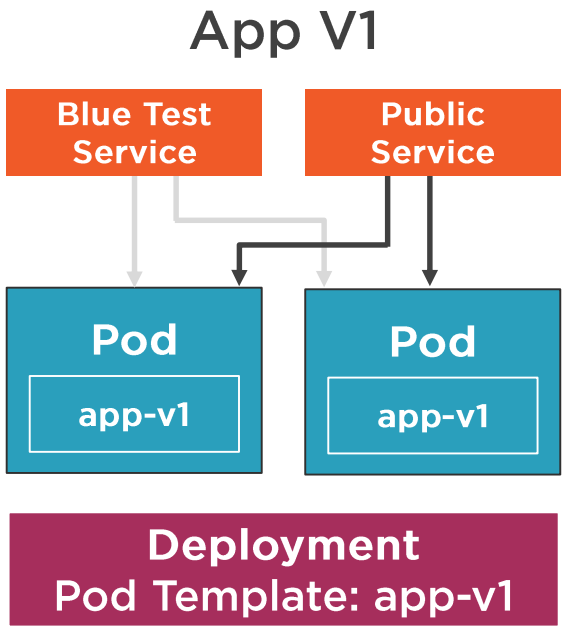
Run two identical production environments at the same time

New application (green) is deployed alongside the old application (blue)

Traffic routed from blue to green when checks pass

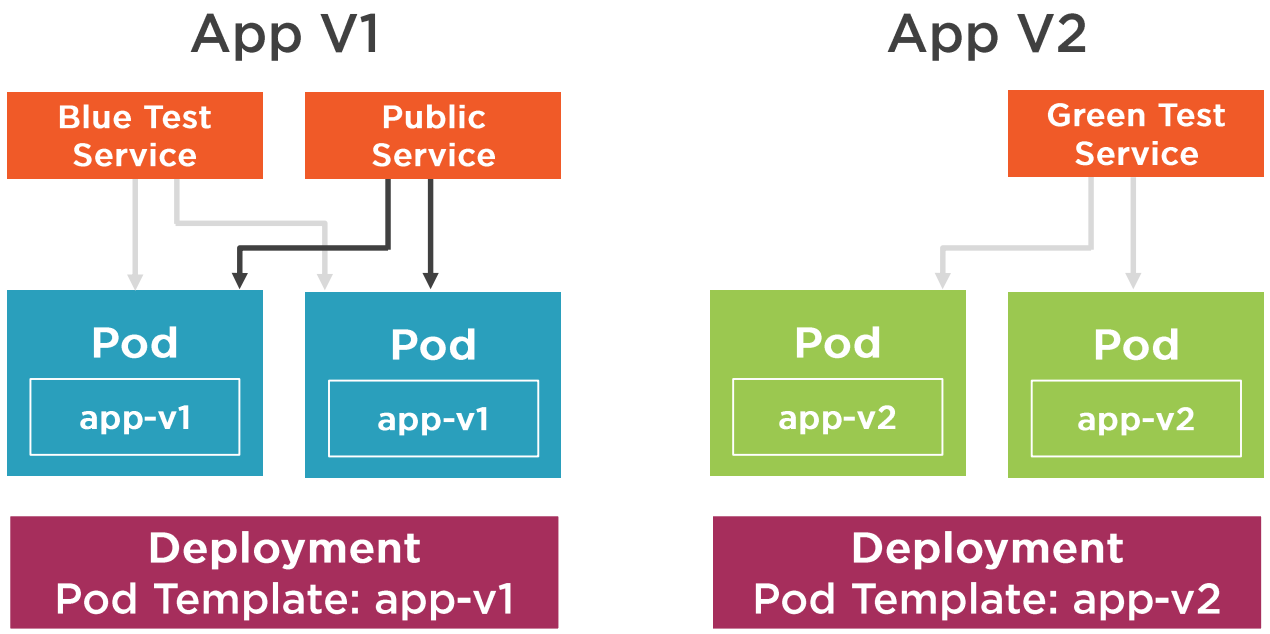
Blue-Green Deployments

1 **Create BLUE Deployment and Services**



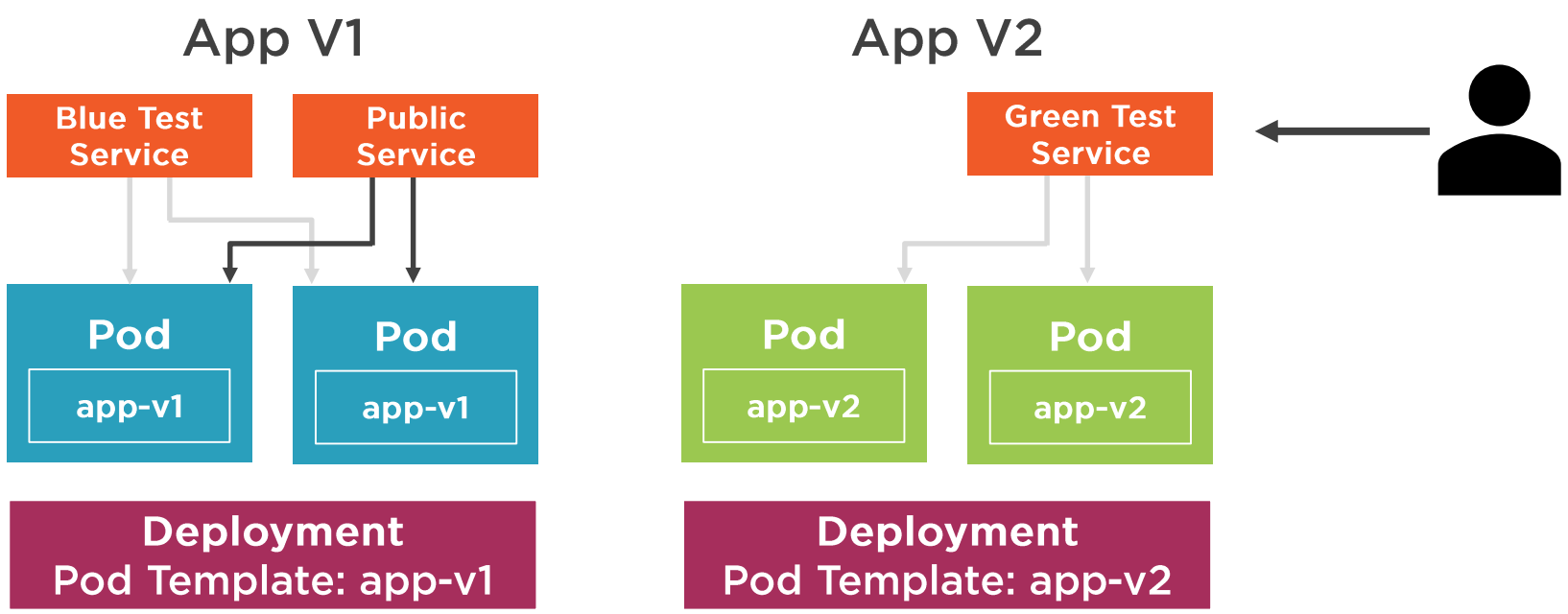
Blue-Green Deployments

2 Create GREEN Deployment and Service



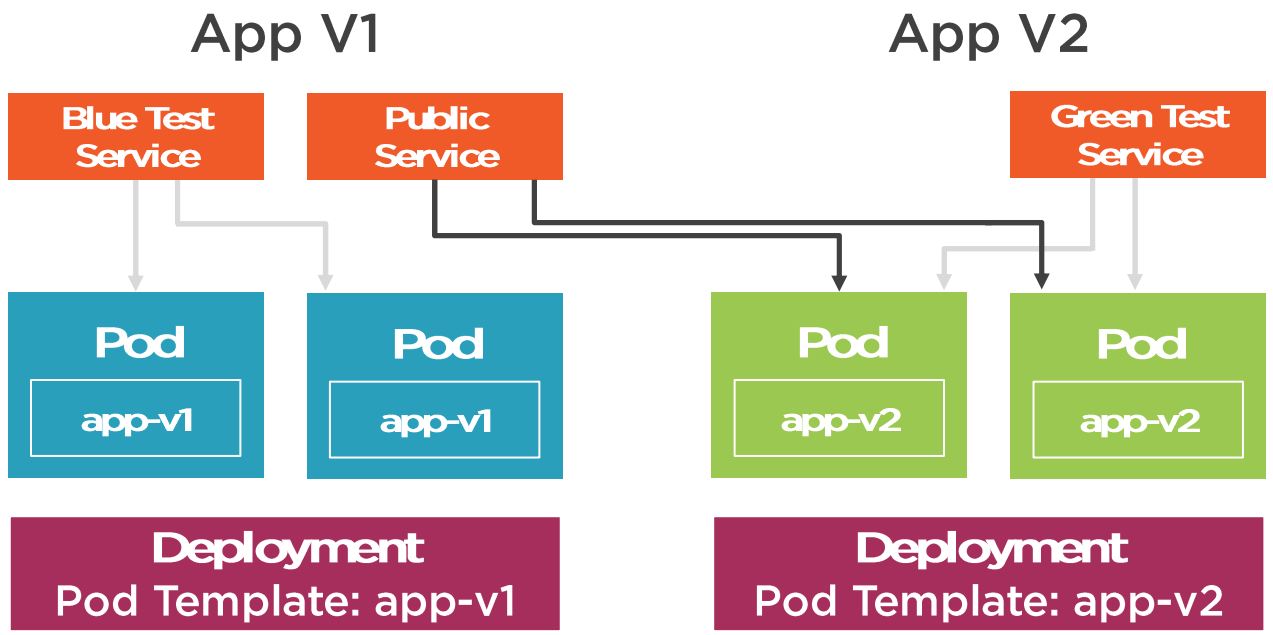
Blue-Green Deployments

3 Test GREEN Pods



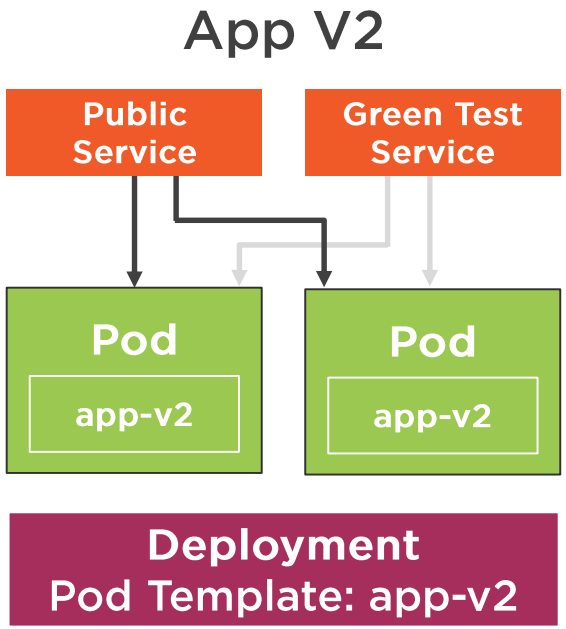
Blue-Green Deployments

4 Change public Service from BLUE to GREEN

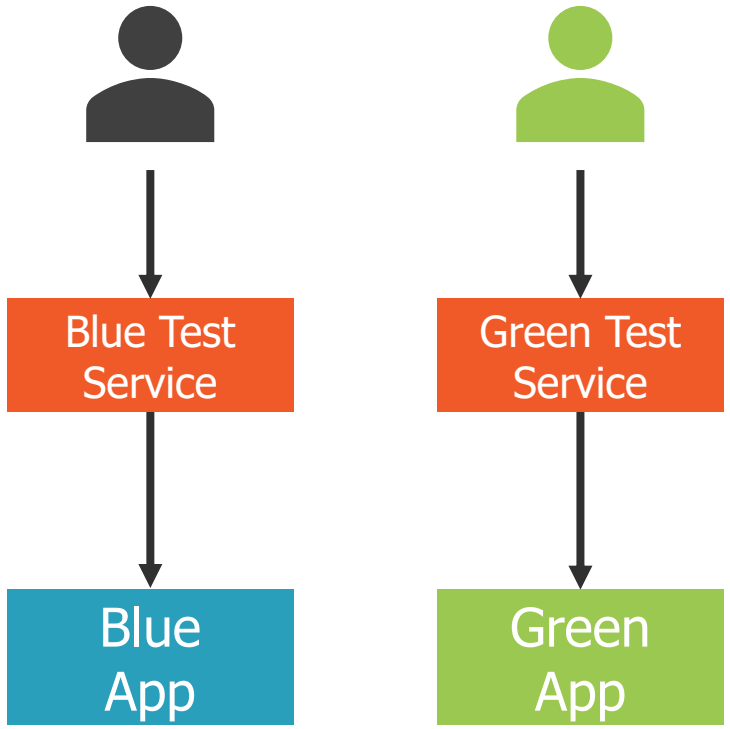


Blue-Green Deployments

5 Remove BLUE Deployment and Service



Blue-Green Deployment Considerations

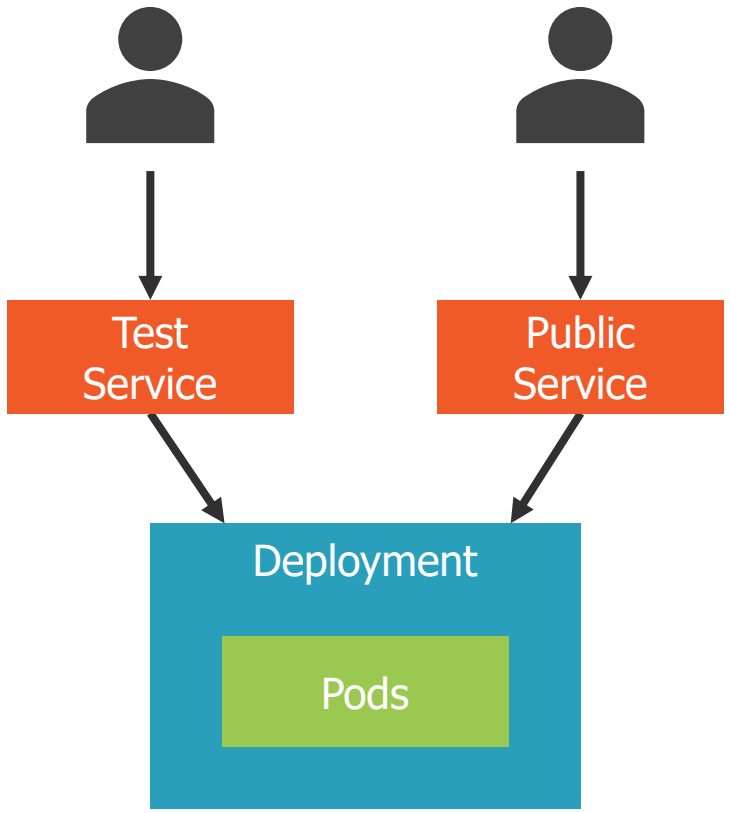


Key considerations:

- How many Pods are being deployed to each environment?
- How much memory is required to run the Pods?
- What are the CPU requirements?
- Other considerations (volumes, sessions, node affinity, etc.)

Creating a Blue-Green Deployment

Blue-Green Deployment Resources

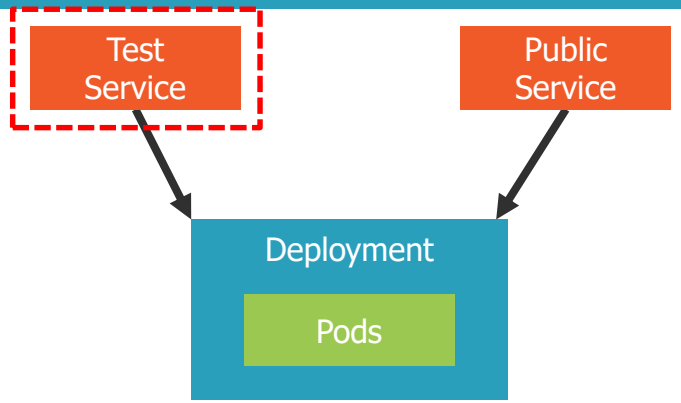


A Blue-Green Deployment involves 3 main Kubernetes resources:

- Test Service
- Public Service
- Deployment

Defining a Test Service

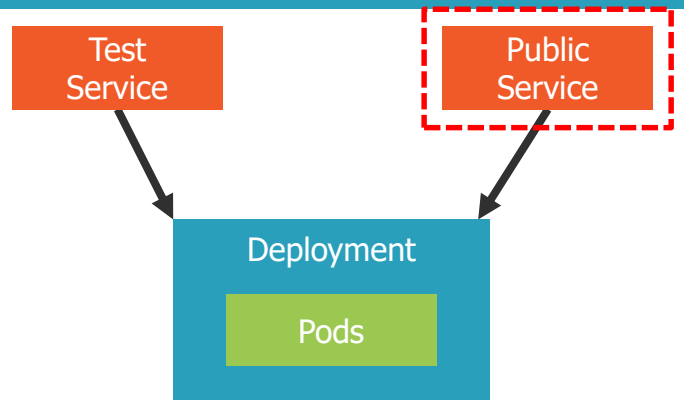
```
kind: Service
apiVersion: v1
metadata:
  name: nginx-blue-test
  labels:
    app: nginx
    role: blue-test
    env: test
spec:
  type: LoadBalancer
  selector:
    app: nginx
    role: blue
  ports:
    - port: 9000
      targetPort: 80
```



- Role that the service plays (blue-test for test environment)
- Service will apply to Pods with these labels (note the role: blue)
- Expose port 9000 (test port)

Defining a Public Service

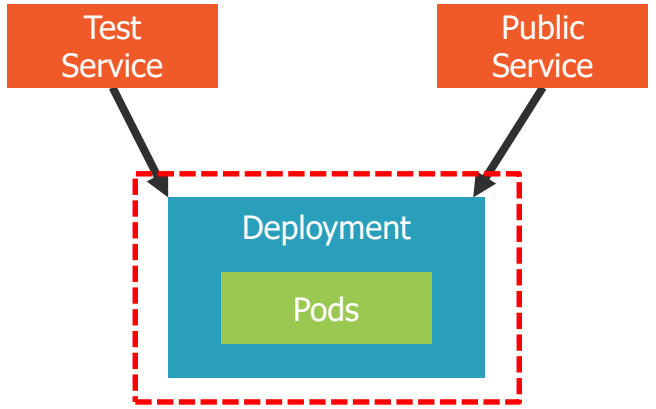
```
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
  labels:
    app: nginx
    role: blue
    env: prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    role: blue
  ports:
    - port: 80
      targetPort: 80
```



- Role that the service plays (blue for production environment)
- Service will apply to Pods with these labels (note the role: blue)
- Expose port 80 (public port)

Defining a Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
      role: blue
  template:
    metadata:
      labels:
        app: nginx
        role: blue
    spec:
      containers:
      - name: nginx-blue
        image: nginx:1.x.x-alpine
        ports:
        - containerPort: 80
```



- Apply to Pods with these labels (note the role: blue)
- Pod labels (note role: blue)
- Container image

Changing From Blue to Green

Once a GREEN deployment has been successfully rolled out and tested, change the public service's selector to "green"


```
# Apply changes made to service's YAML (declarative)
kubectl apply -f file.service.yml

# Change Service's selector to green (imperative)
kubectl set selector svc [service-name] 'role=green'
```

Blue-Green Deployments in Action – The Blue Deployment

Blue-Green Deployments in Action – The Green Deployment

Summary

Blue-Green Deployments allow two environments to be deployed at the same time

Provides a way to test a new version of an application before switching over to it

Works by changing the "blue" Service's selector to point to the "green" Deployment

Running Jobs and CronJobs

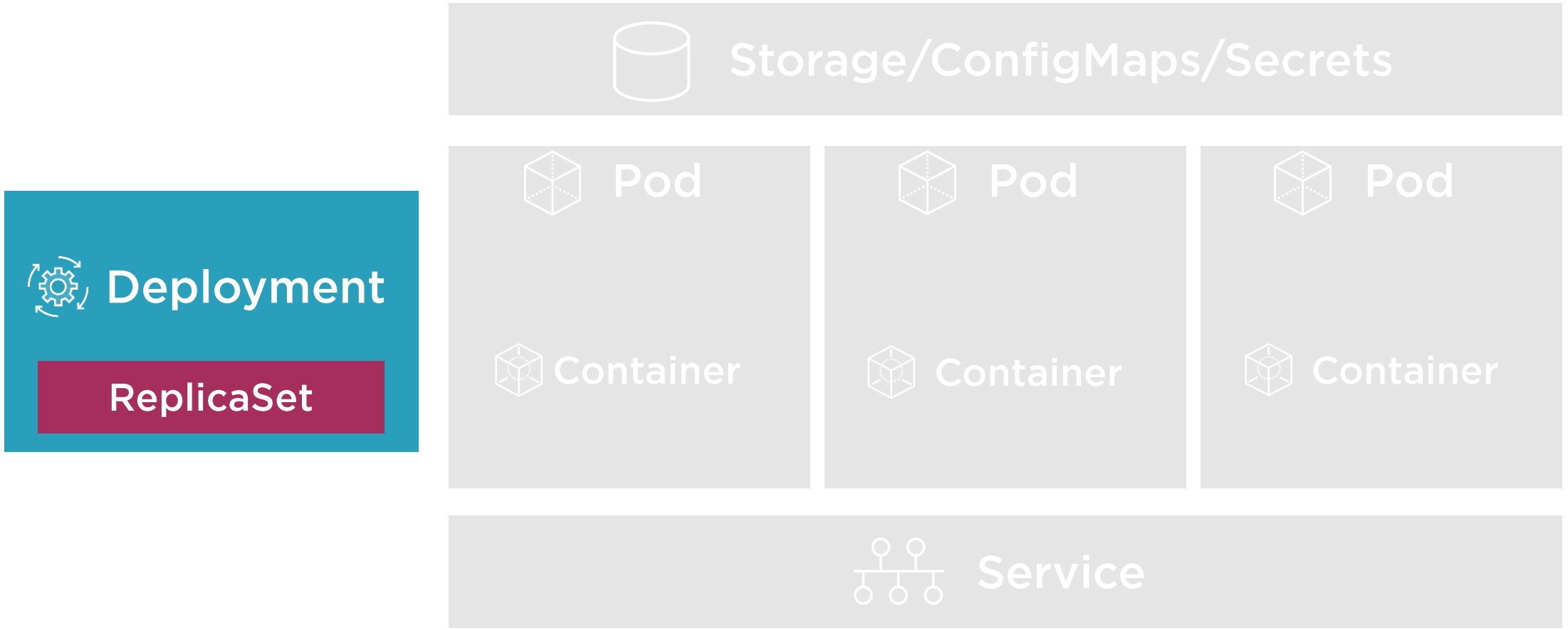
Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

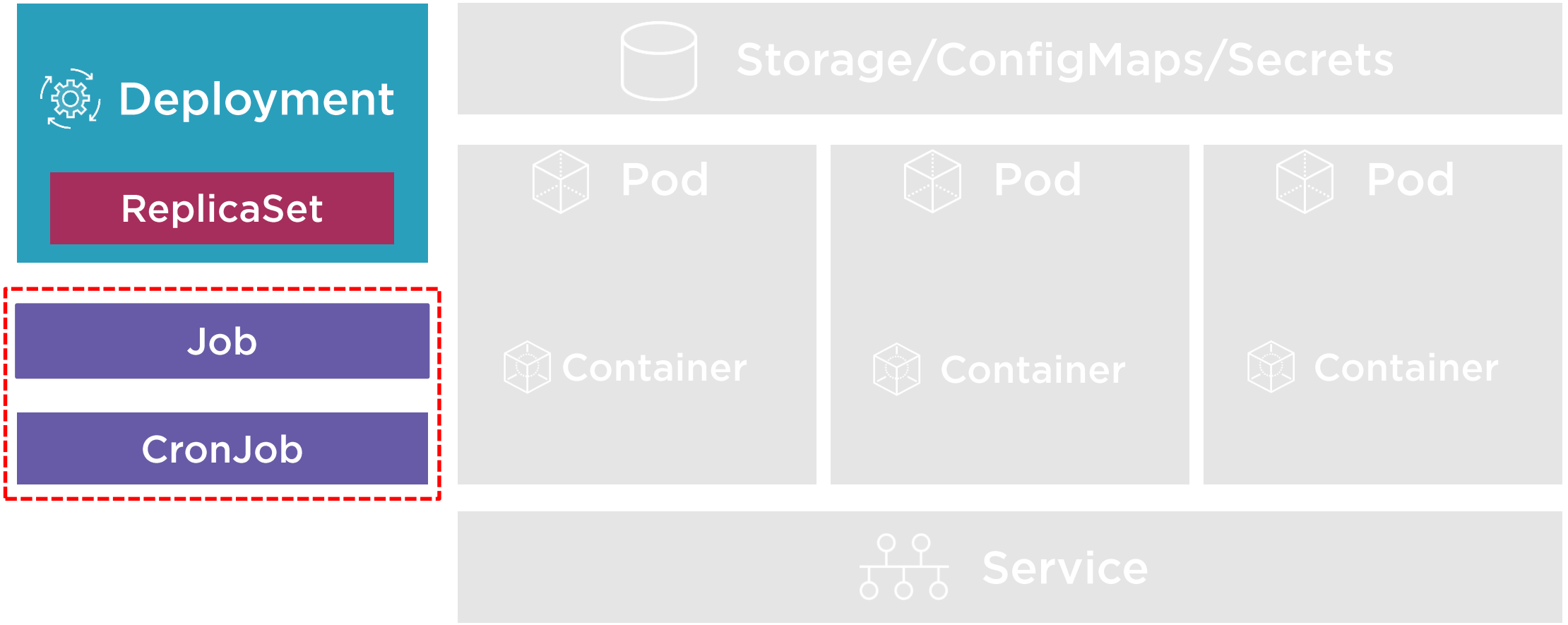
Module Overview

- Understanding Jobs
- Understanding CronJobs
- Creating a Job and CronJob
- Jobs in Action
- CronJobs in Action

Kubernetes Resources



Kubernetes Resources



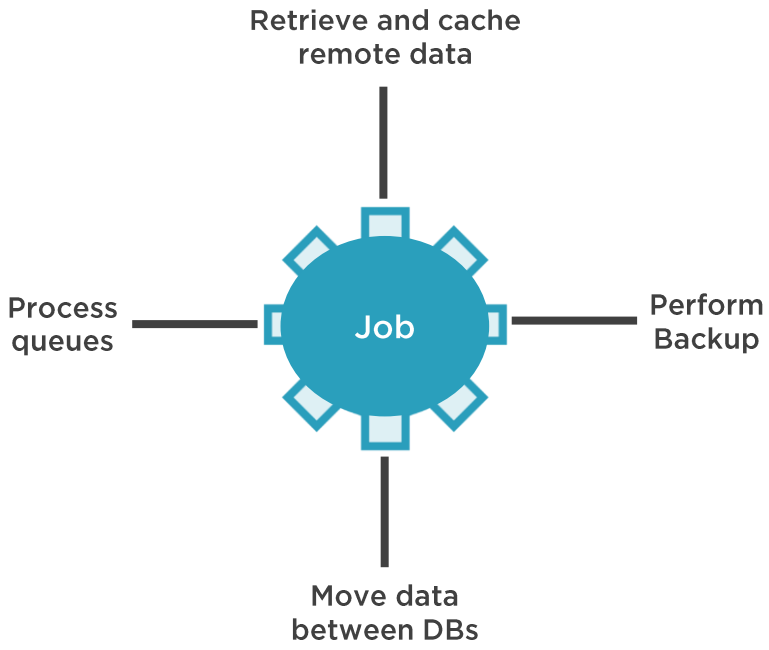
Understanding Jobs

Have you ever needed to a run a job that performs a task and then terminates?

"A Job creates one or more Pods and ensures that a specified number of them successfully terminate."

~ **Kubernetes Documentation**

Understanding Jobs



A Job creates a Pod(s) that performs a task or batch process

Unlike standard Pods, a Job does not run indefinitely

A Job can be configured to run multiple Pods in parallel

Successful completions are tracked

Once a Job is deleted its Pods are removed

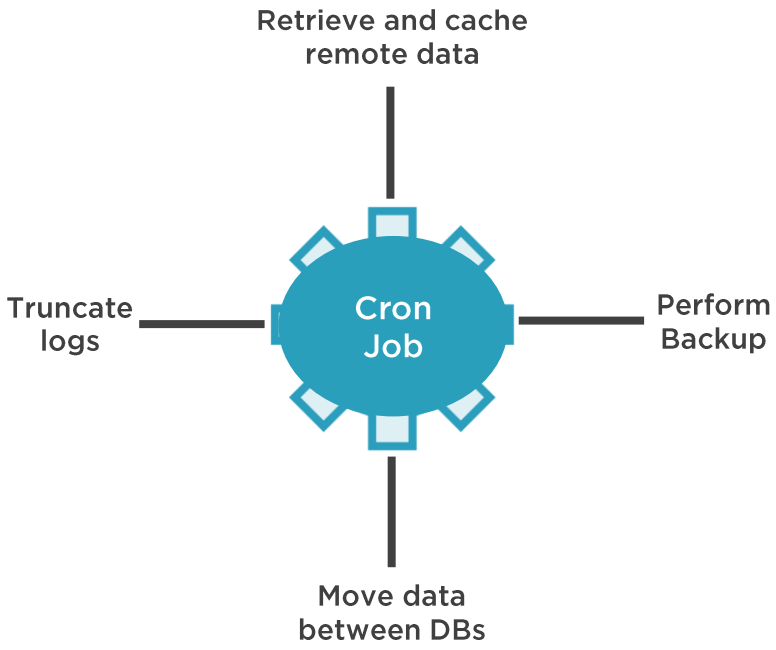
Understanding CronJobs

Have you ever needed to a run a job on a scheduled basis?

"A Cron Job creates Jobs on a time-based schedule."

~ **Kubernetes Documentation**

Understanding CronJobs

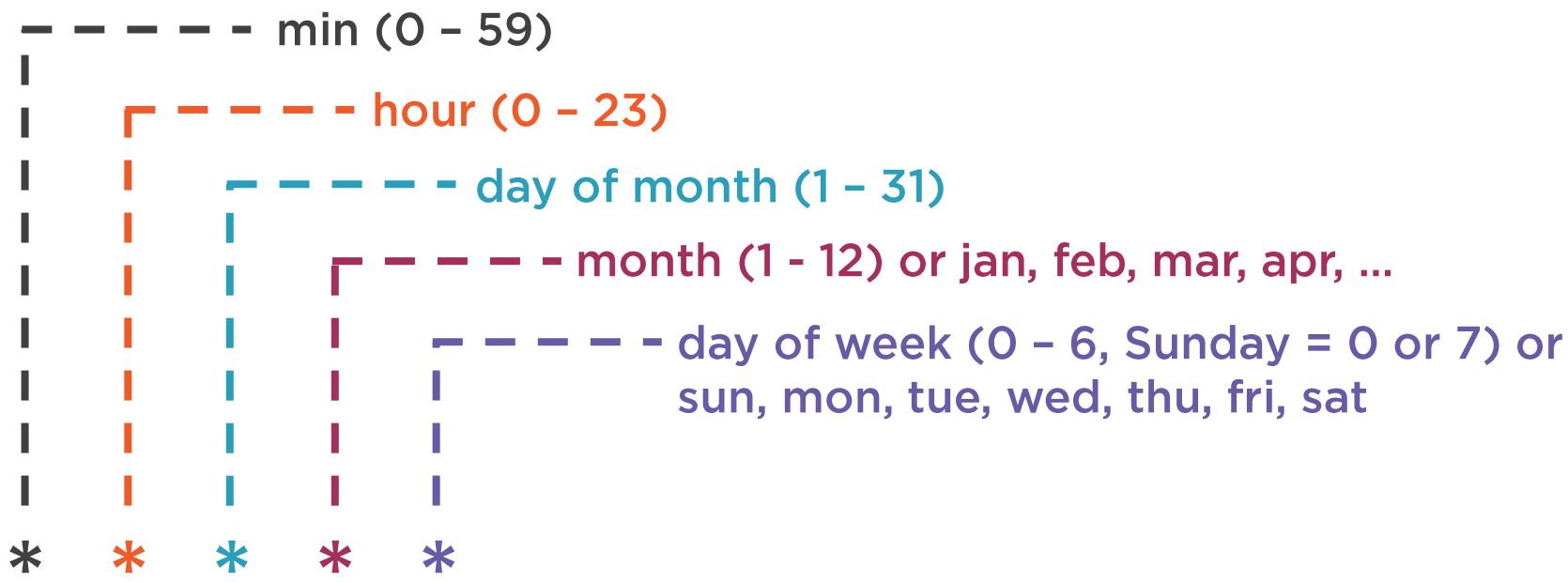


A CronJob is a Job that runs on a scheduled basis

Scheduled using the Cron format

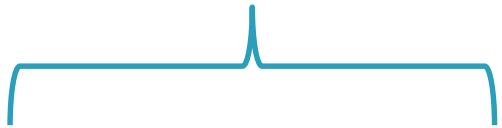
CronJob names must be less than 52 characters

Understanding the Cron Format



Cron Format Examples

Run at 22:30 every Monday



```
30  22  *  *  1
```

Run at 00:01 on the first day of each month

```
1  0  1  *  *
```

Additional Cron Formats

```
0 * * * *

0 0 * * *

0 0 * * 0

0 0 1 * *

0 0 1 1 *

*/1 * * * *
```

- **@hourly** – run once every hour
- **@daily** – run once every day at midnight
- **@weekly** – run once every week
- **@monthly** – run once every month
- **@yearly** – run once every year
- Run once every minute

Creating a Job and CronJob

Defining a Job

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pie-counter
spec:
  template:
    metadata:
      name: pie-counter
    spec:
      restartPolicy: Never
      containers:
      - name: pie-counter
        image: alpine
        command:
         - "sh"
         - "-c"
         - "echo 'scale=1000; 4*a(1)' …;"
```

- Batch API
- Job kind
- Never try to restart (Never or OnFailure)
- Job command to run

Defining a Job that Requires Multiple Completions

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pie-counter
spec:
  completions: 4
  template:
    ...
```

- Run 4 Pods sequentially

Defining a Job that Can Run in Parallel

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pie-counter
spec:
  completions: 4
  parallelism: 2
  template:
    ...
```

- 4 Pods must complete successfully
- 2 Pods can run in parallel at a time

Creating a Job

A job can be created using the standard kubectl create or kubectl apply commands

```
# Create a new Job
kubectl create –f file.job.yml --save-config

# Creating or modifying a Job
kubectl apply –f file.job.yml
```

Defining a CronJob

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: pie-counter
spec:
  concurrencyPolicy: Allow
  # Run the job every minute
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          restartPolicy: OnFailure
          containers:
          - name: pie-counter
            image: alpine
            command:
            - "sh"
            - "-c"
            - "echo 'scale=1000; 4*a(1)'
```

- CronJob batch API
- CronJob kind

- Allow multiple Pods to run event if their scheduling overlaps
- Cron format to use for scheduling

- Restart if there's a failure

- Command to run

Creating a CronJob

A CronJob can be created using the standard kubectl create or kubectl apply commands

```
# Create a new CronJob
kubectl create –f file.cronjob.yml --save-config

# Creating or modifying a CronJob
kubectl apply –f file.cronjob.yml
```

Jobs in Action

CronJobs in Action

Summary

- Jobs are used to run a task or batch process
- Successful completions are tracked
- CronJobs allow a task/batch process to be run on a scheduled basis
- Relies on the Cron format

Performing Monitoring and Troubleshooting Tasks

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Module Overview

Monitoring and Troubleshooting Overview

Web UI Dashboard in Action

Prometheus, Metrics Server, and kube-state-metrics in Action

Grafana in Action

Troubleshooting Techniques in Action

Monitoring and Troubleshooting Overview

How important is it to monitor Deployments, Pods, and other resources in your Kubernetes cluster?

Q What should you monitor?

A Short answer....everything!

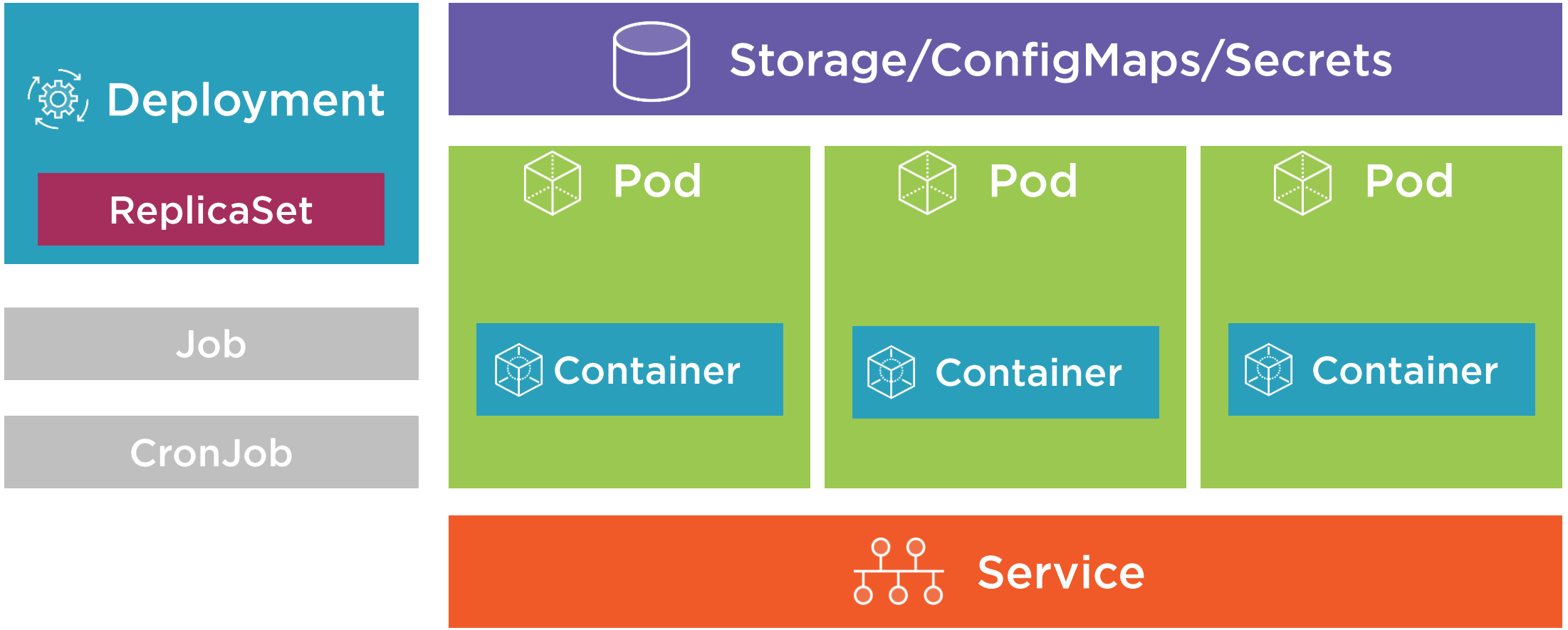
"Kubernetes makes managing a containerized infrastructure much easier by creating levels of abstractions such as pods and services. We no longer have to worry about where applications are running or if they have enough resources to work properly."

https://kubernetes.io/blog

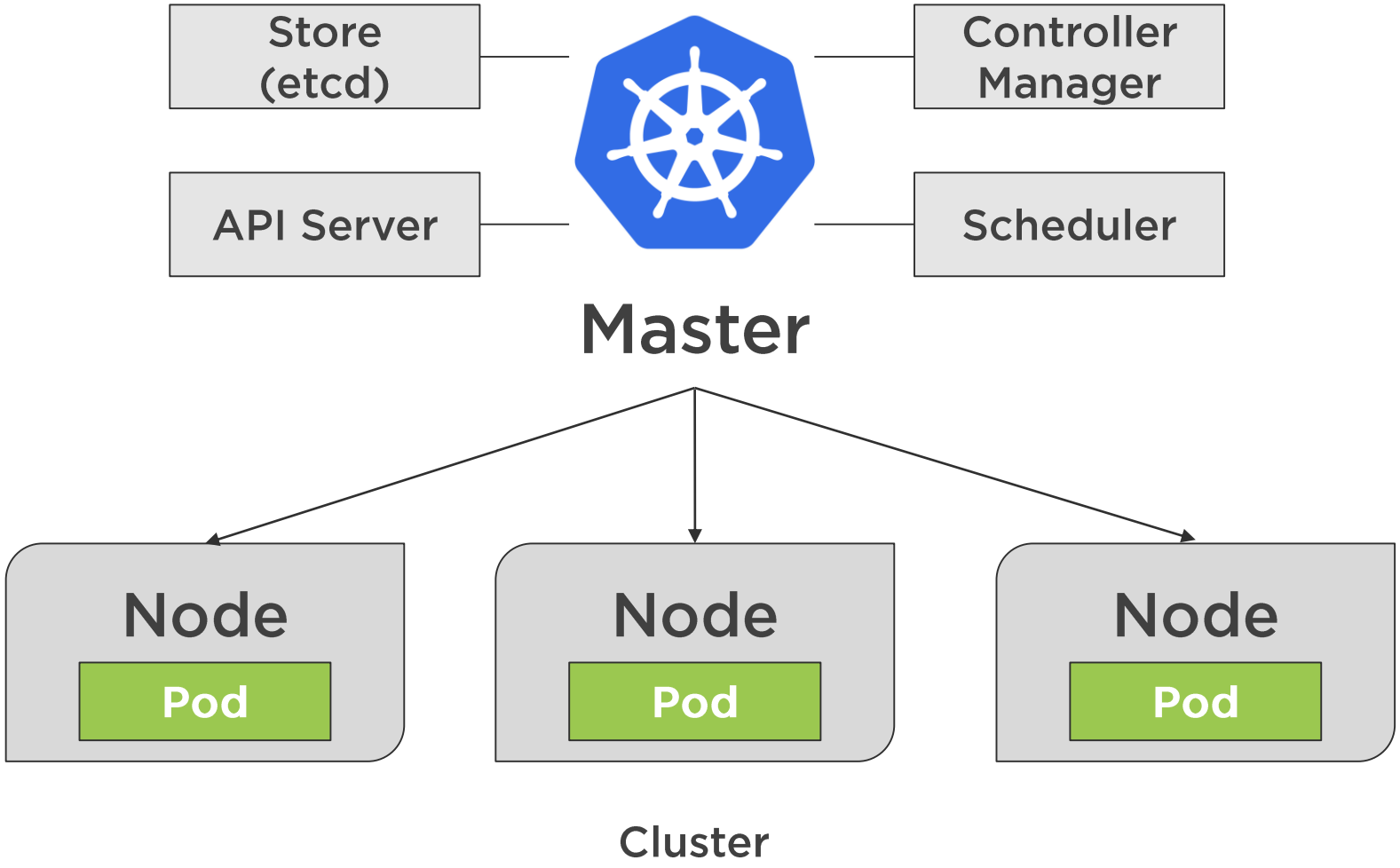
"Kubernetes makes managing a containerized infrastructure much easier by creating levels of abstractions such as pods and services. We no longer have to worry about where applications are running or if they have enough resources to work properly. But that doesn't change the fact that, in order to ensure good performance, we need to monitor our applications, the containers running them, and Kubernetes itself."

https://kubernetes.io/blog

What Should You Monitor?



What Should You Monitor?



Monitoring and Troubleshooting Options

Key monitoring and alerting options:

- Web UI Dashboard
- Metrics Server
- kube-state-metrics
- Prometheus
- Grafana
- Many more...

Web UI Dashboard in Action

"**Web UI (Dashboard)** is a web-based Kubernetes user interface. You can use Dashboard to deploy containerized applications to a Kubernetes cluster, troubleshoot your containerized application, and manage the cluster resources."

https://kubernetes.io

https://github.com/kubernetes/dashboard

Metrics Server, kube-state-metrics, and Prometheus in Action

"Metrics Server is a cluster-wide aggregator of resource usage data. It is deployed by default in clusters created by kube-up.sh script as a Deployment object."

https://kubernetes.io

https://github.com/kubernetes-sigs/metrics-server

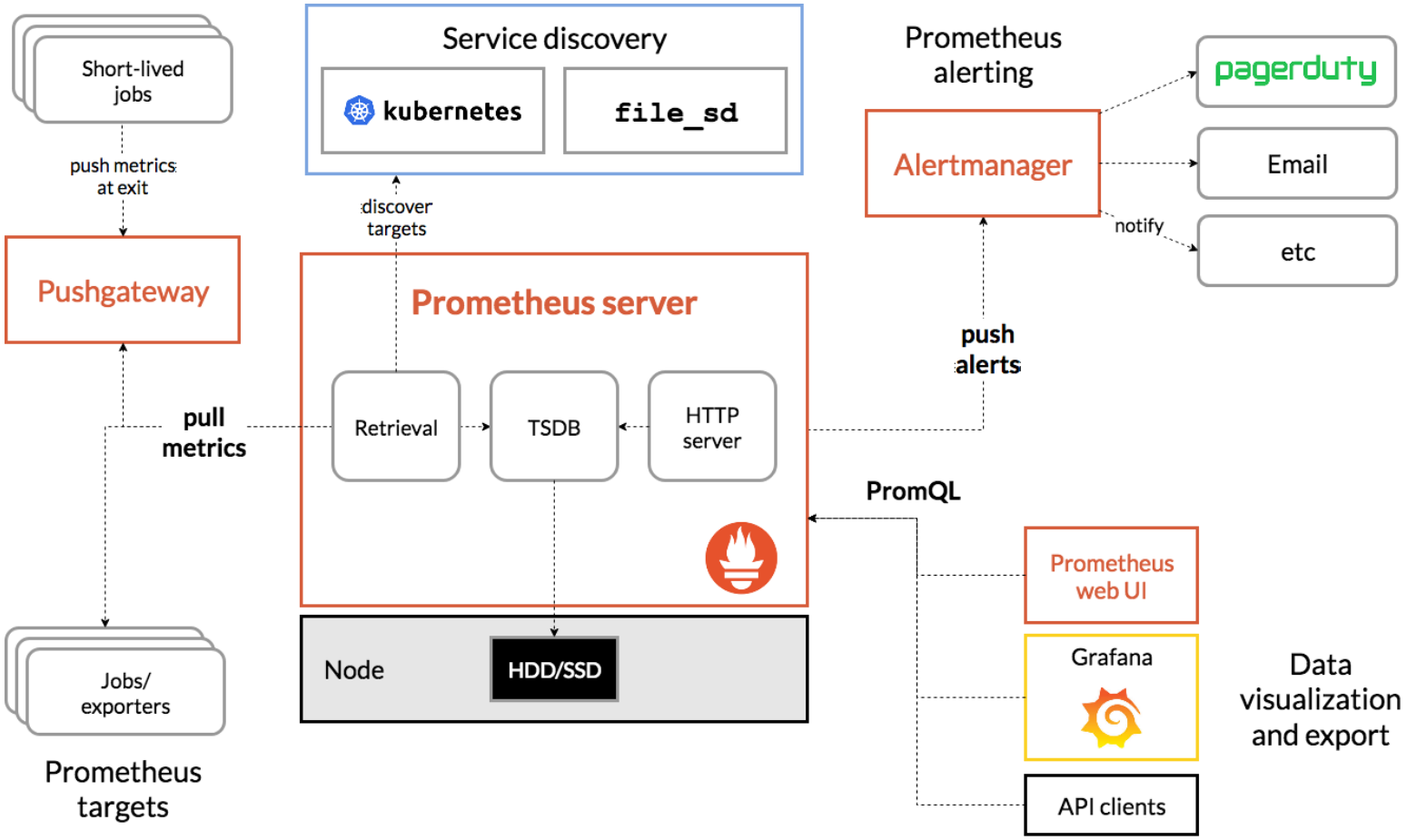
"kube-state-metrics is a simple service that listens to the Kubernetes API server and generates metrics about the state of the objects. It is not focused on the health of the individual Kubernetes components, but rather on the health of the various objects inside, such as deployments, nodes and pods."

https://github.com/kubernetes/kube-state-metrics

"Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud....Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes."

https://prometheus.io

Prometheus Architecture



https://prometheus.io/docs/introduction/overview

Grafana in Action

"Grafana allows you to query, visualize, alert on and understand your metrics no matter where they are stored. Create, explore, and share dashboards with your team and foster a data driven culture."

https://grafana.com

Troubleshooting Techniques with kubectl

Key Troubleshooting Commands

Key troubleshooting commands:

- kubectl get pod [pod-name] -o yaml
- kubectl describe pod [pod-name]
- kubectl exec [pod-name] –it sh

Viewing Pod logs

- kubectl logs [pod-name]
- kubectl logs [pod-name] –c [container-name]
- kubectl logs –p [pod-name]
- kubectl logs –f [pod-name]

Troubleshooting Techniques in Action

Summary

Monitoring should be a critical part of any Kubernetes rollout plan

Several tools can be used to provide monitoring and alerts:

- Web UI Dashboard
- Prometheus
- Grafana
- Many more...

Master kubectl commands that can help troubleshoot issues with Kubernetes resources

Putting It All Together

Dan Wahlin
WAHLIN CONSULTING

@danwahlin www.codewithdan.com

Thank You!

Dan Wahlin
Wahlin Consulting

@DanWahlin
https://codewithdan.com