Certified Kubernetes Application Developer: Application Deployment

Use Kubernetes Primitives to Implement Common Deployment Strategies

Dan Wahlin
Wahlin Consulting

@danwahlin codewithdan.com

Certified Kubernetes Application Developer

Application Design and Build

Application Deployment

Application Observability and Maintenance

Application Environment, Configuration and Security

Services and Networking

About this Course

Use Kubernetes Primitives to Implement Common Deployment Strategies

Understand Deployments and How to Perform Rolling Updates

Use the Helm Package Manager to Deploy Existing Packages

Pre-requisites

Cloud
Basic experience

Containers
Basic experience

Kubernetes
Basic experience

Suggested courses

Containers and Kubernetes: The Big Picture

Nigel Poulton

Kubernetes for Developers: Core Concepts

Dan Wahlin

@danwahlin

@danwahlin

- Building and Running Your First Docker App
- Docker for Web Developers
- Kubernetes for Developers: Core Concepts
- Kubernetes for Developers: Deploying Your Code
- Kubernetes for Developers: Moving from Docker Compose to Kubernetes

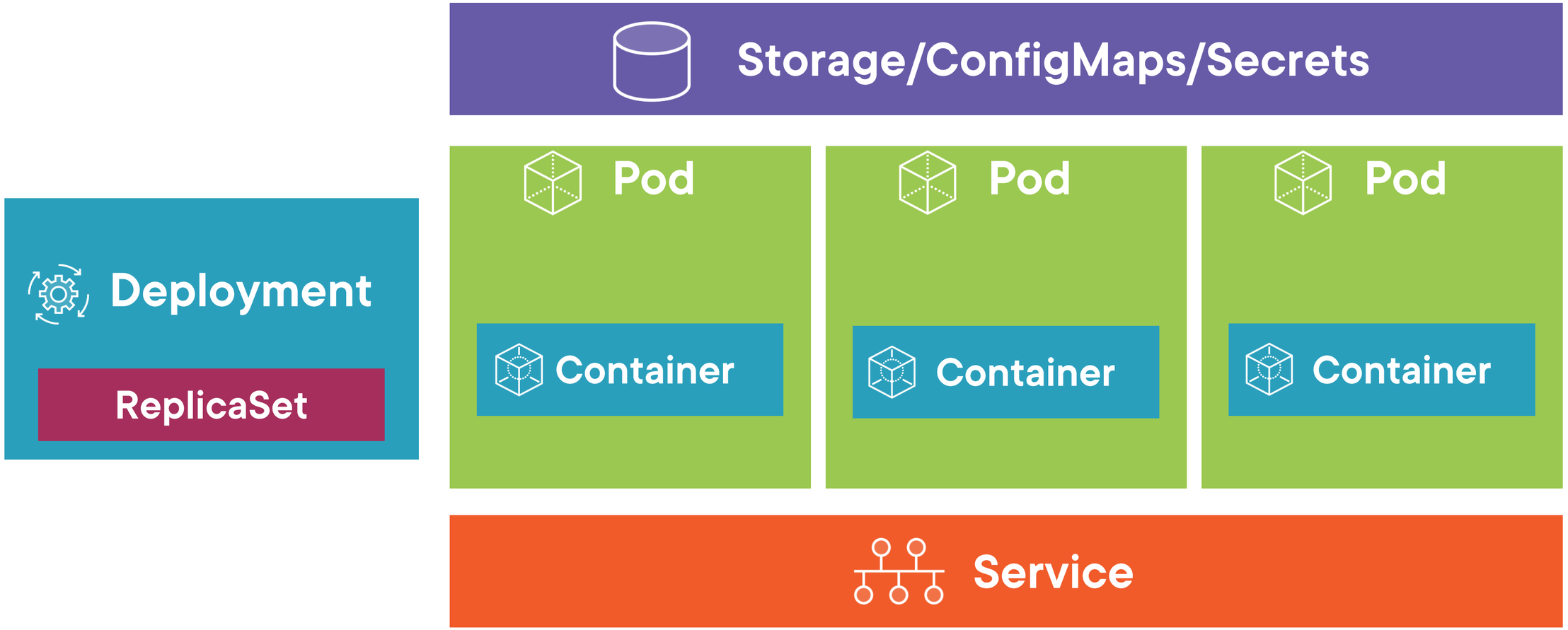
Agenda

Use Kubernetes Primitives to Implement Common Deployment Strategies

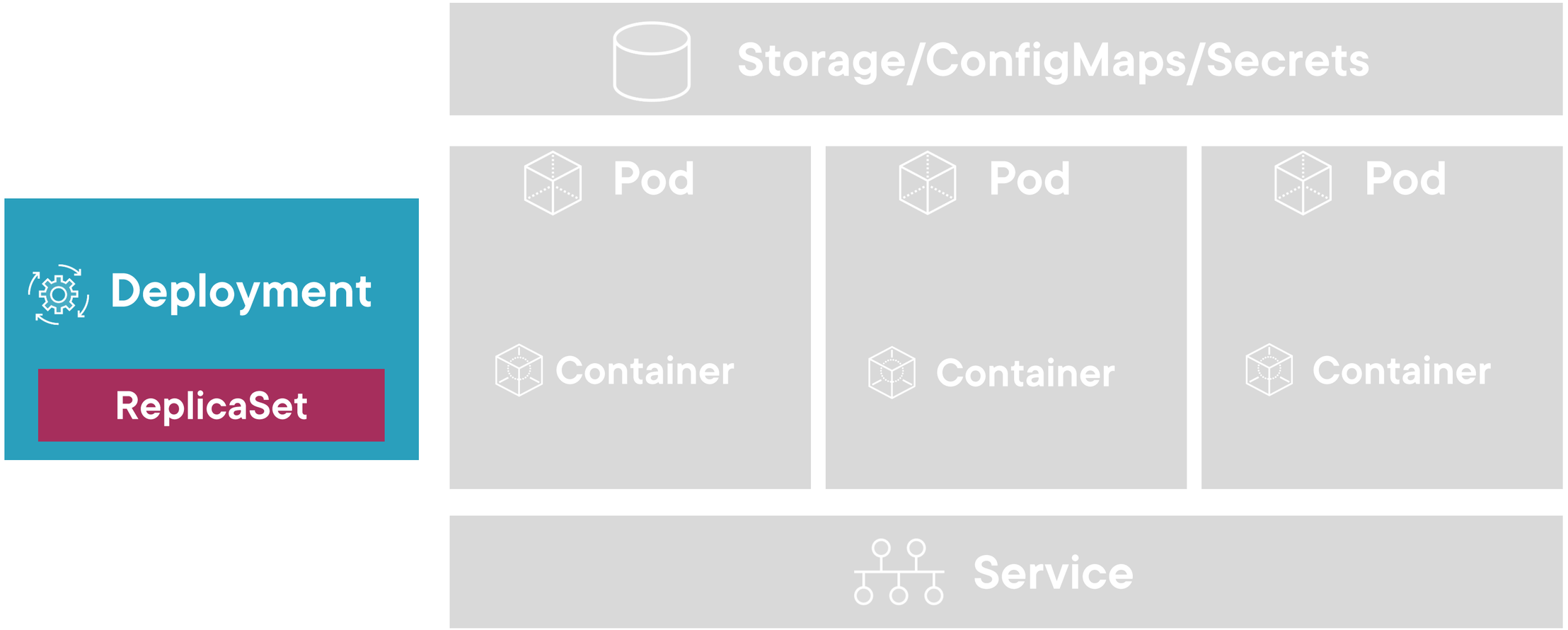
- Understanding Deployments
- Working with Blue/Green Deployments
- Working with Canary Deployments
- Exam Scenarios
- Recap and Test Yourself

Understanding Deployments

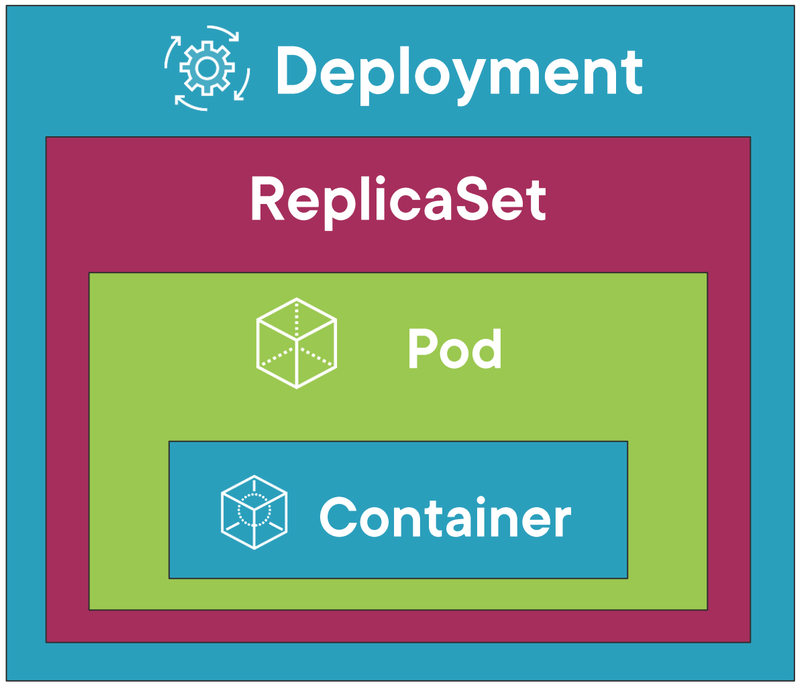
Understanding Deployments



Understanding Deployments



Deployment Options



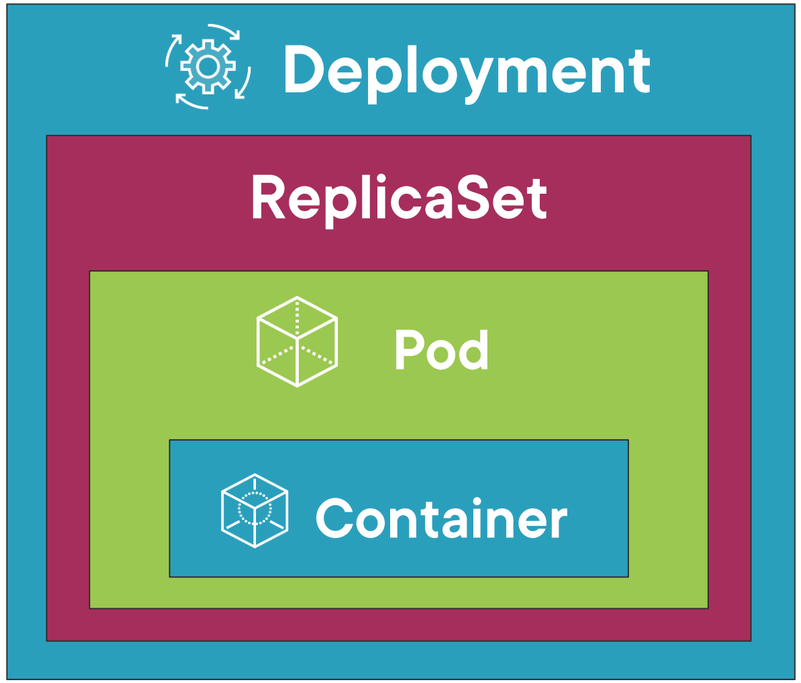
Kubernetes supports several Deployment options:

- Rolling Updates
- Blue/Green Deployments
- Canary Deployments
- Rollbacks

Creating a Deployment

Deployment

+ kubectl =



Defining a Deployment

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

- 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

```
# Create a Deployment YAML file
kubectl create deployment nginx --image=nginx:alpine
  --dry-run=client -o yaml > deploy.yaml

# Create a Deployment in Kubernetes
kubectl create –f deploy.yaml
```

Creating a Deployment

Use the *kubectl create* command to create a deployment and save it to a YAML file.

```
# Imperatively scale the Deployment Pods to 4
kubectl scale deployment nginx --replicas=4


# Declarative scale the Deployment Pods to 4
nano deploy.yaml

kubectl apply –f deploy.yml
```

```
spec:
  replicas: 4
  selector:
    tier: frontend
```

Scaling Pods

Use the *kubectl scale* command or update the YAML file

```
# Imperatively change the Deployment image
kubectl set image deployment/nginx nginx


# Declaratively change the Deployment image
nano deploy.yaml
kubectl apply -f deploy.yml
```

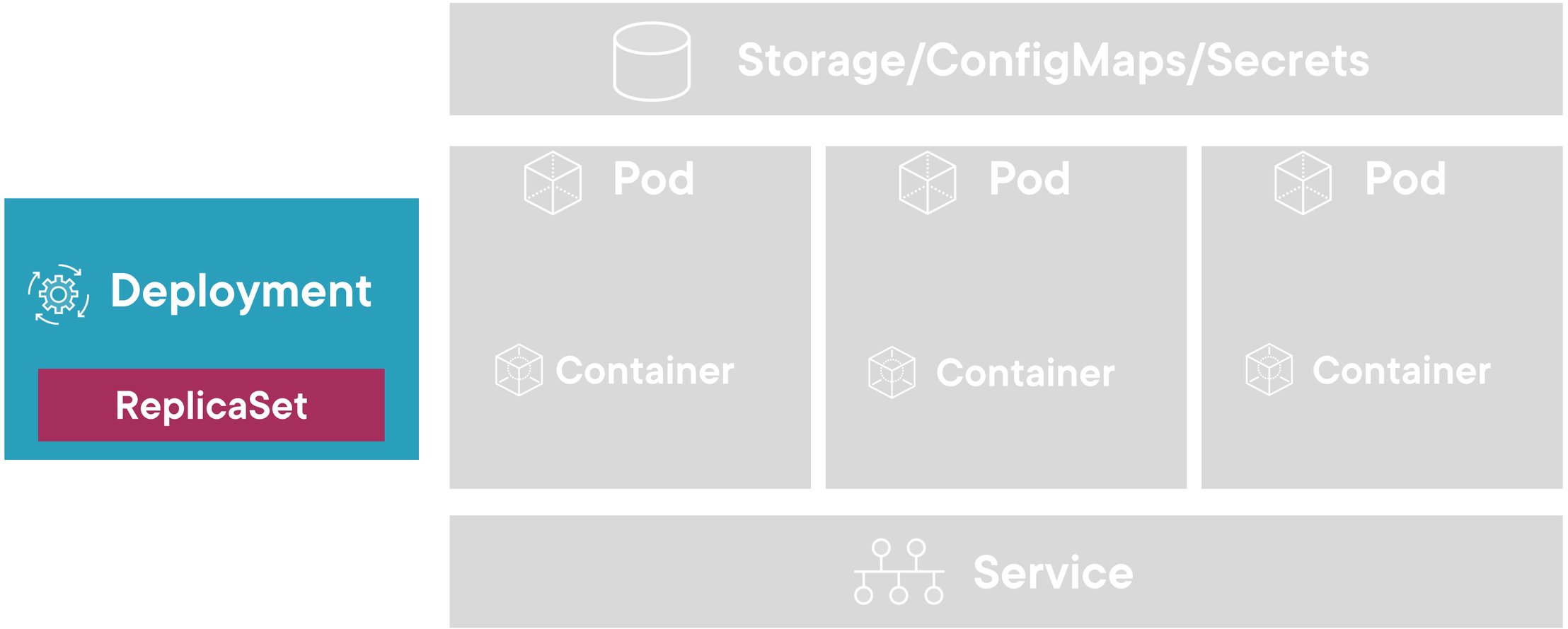
```
containers:
  - name: nginx
    image: nginx
```

Changing a Deployment Image

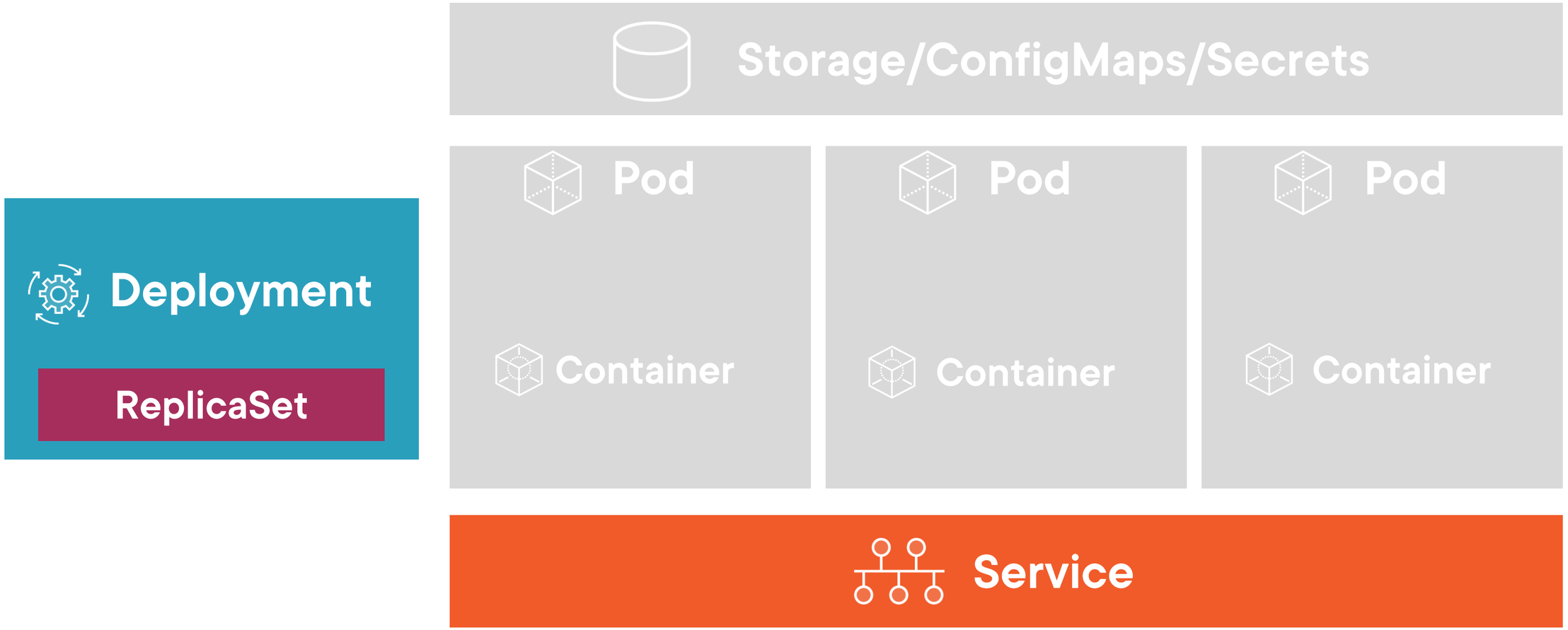
Use the *kubectl set image* command change a Deployment's image

Working with Blue/Green Deployments

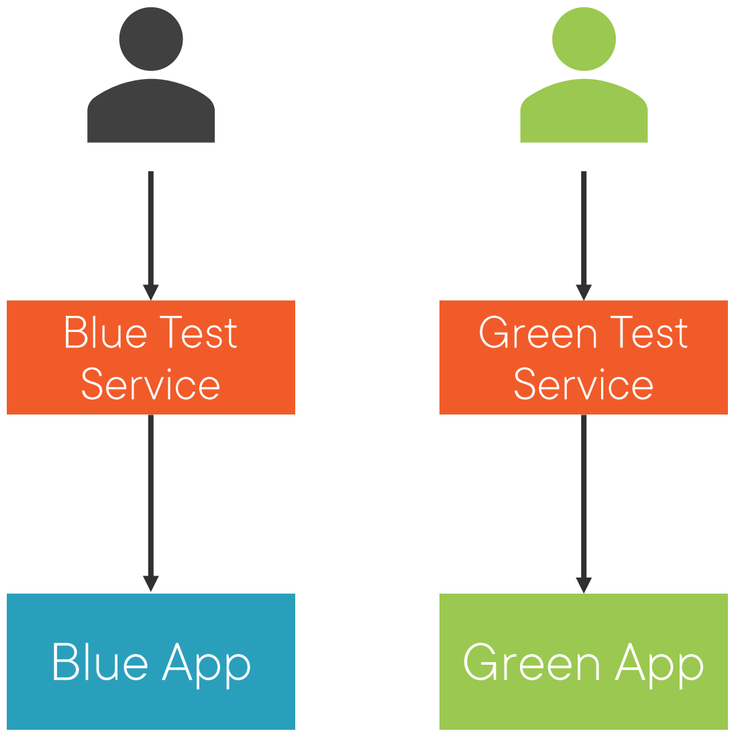
Blue/Green Deployments and Services



Blue/Green Deployments and Services



Blue/Green Deployments



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

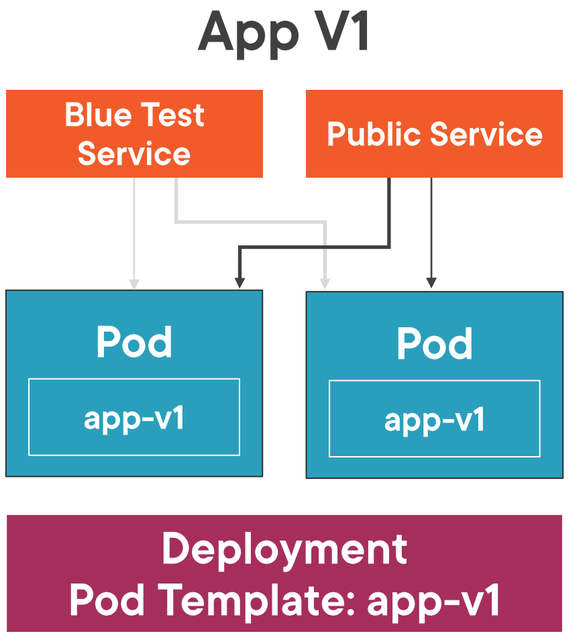
Run two identical production environments at the same time

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

Traffic routed from blue to green when checks pass

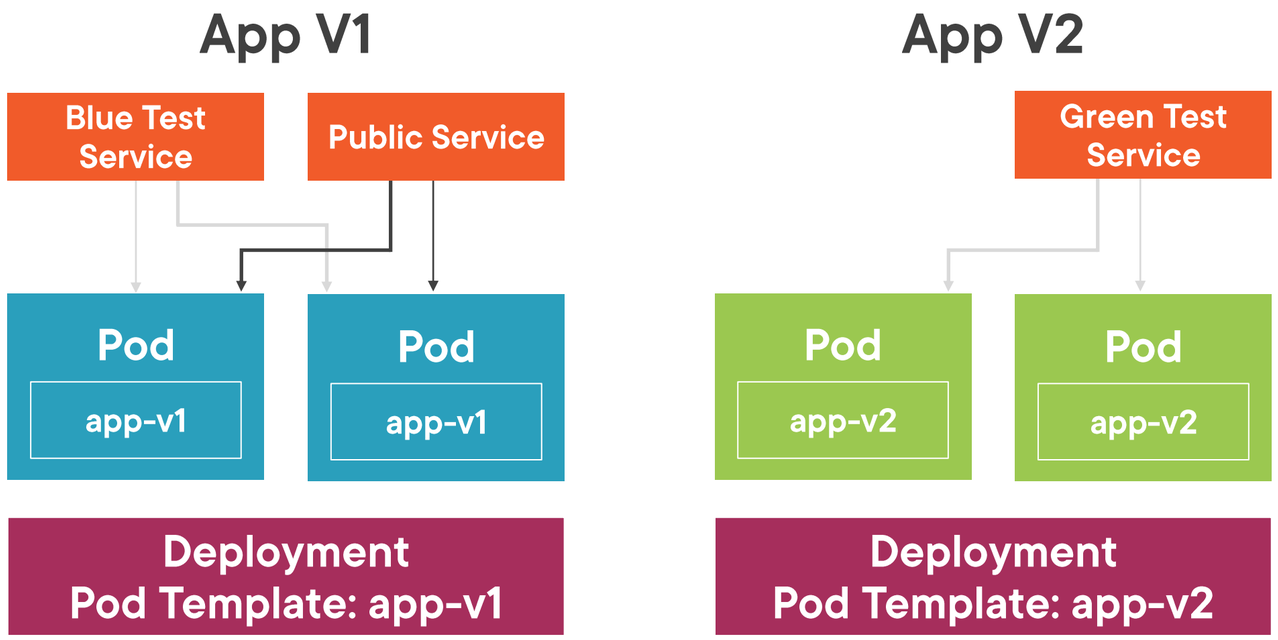
Blue/Green Deployments

1 **Create BLUE Deployment and Services**



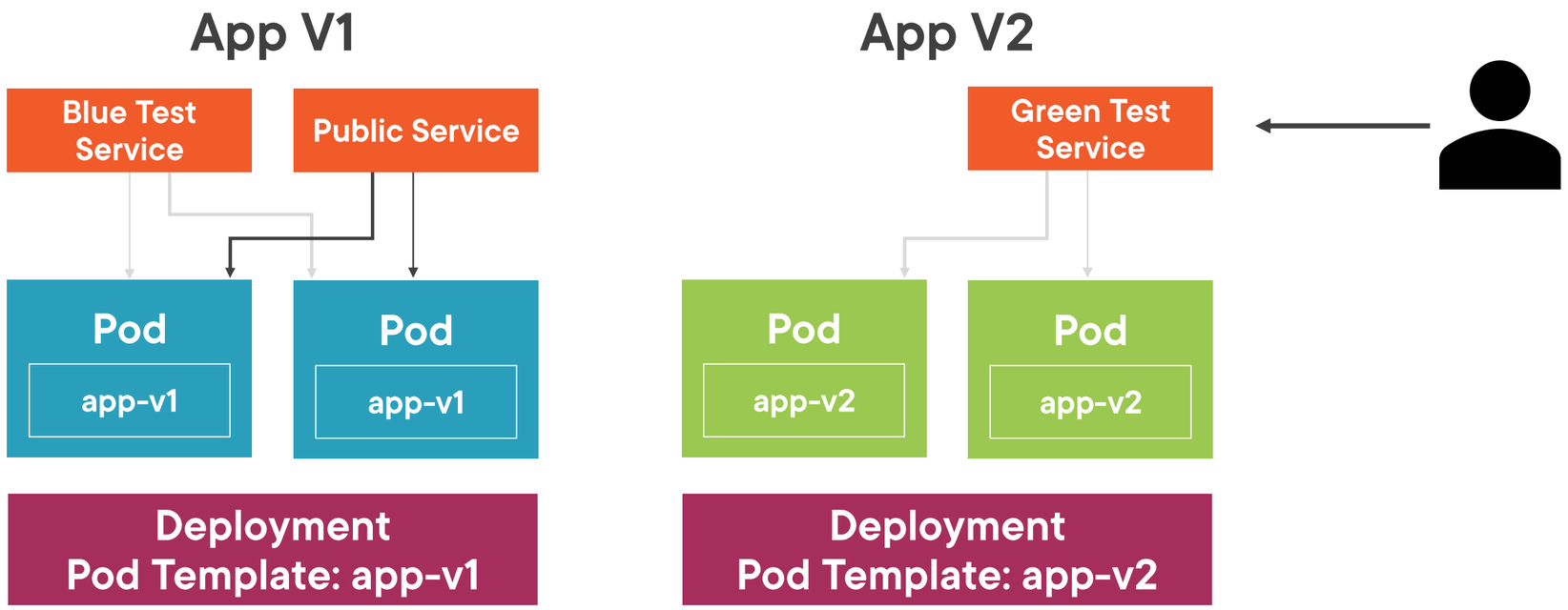
Blue/Green Deployments

2 Create GREEN Deployment and Service



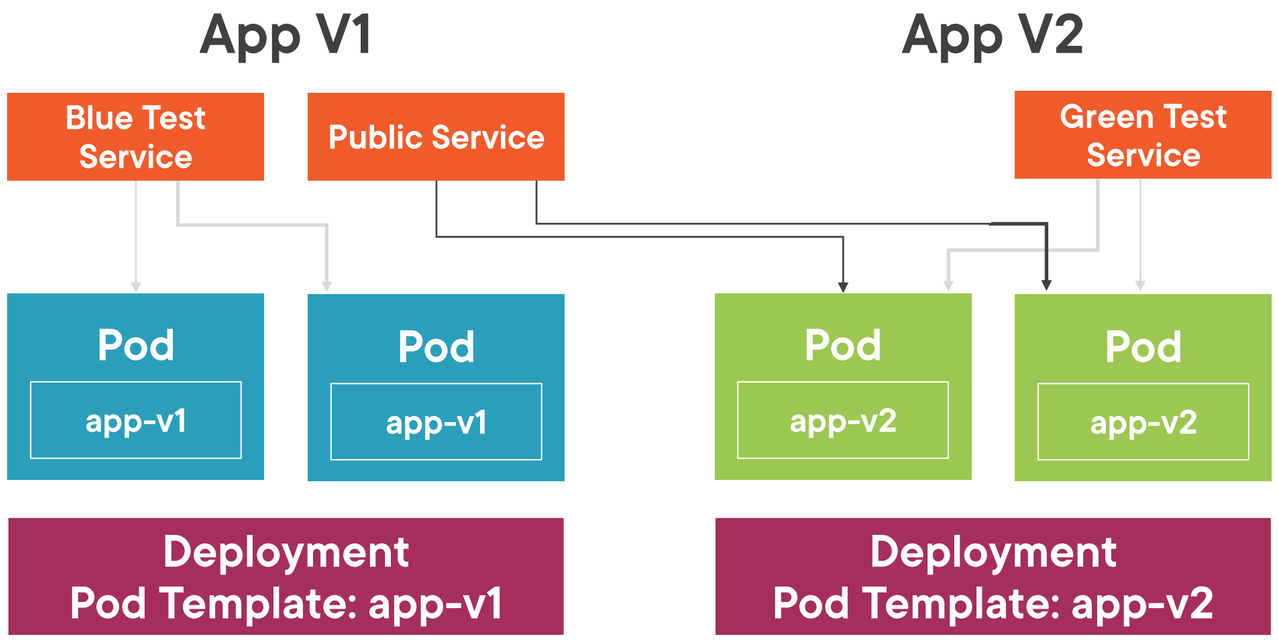
Blue/Green Deployments

3 **Test GREEN Pods**



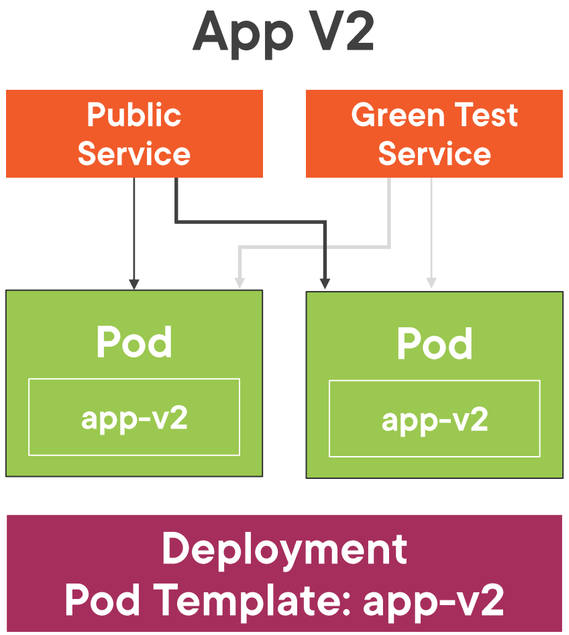
Blue/Green Deployments

4 **Change public Service from BLUE to GREEN**

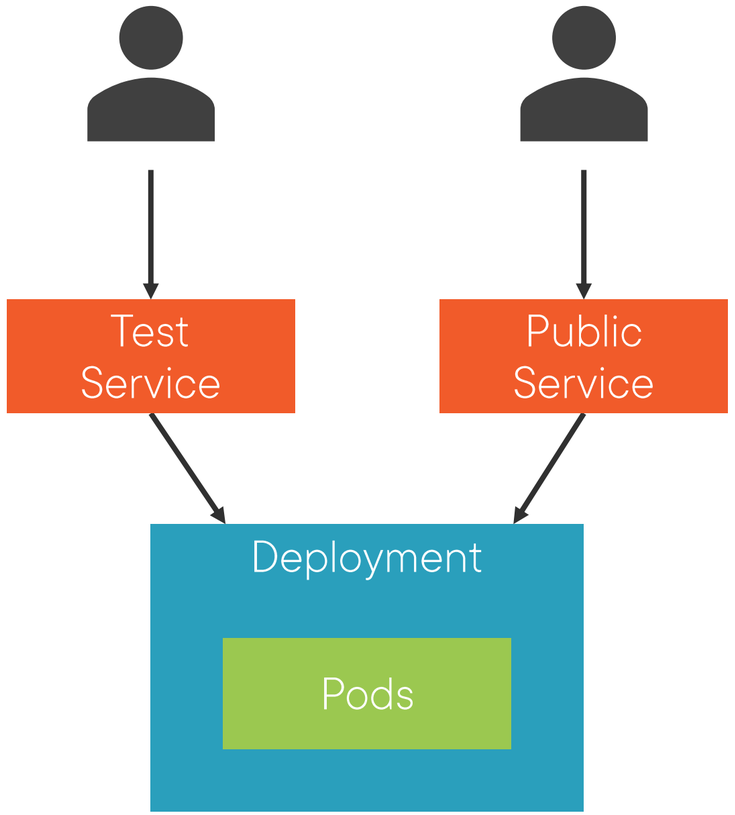


Blue/Green Deployments

5 **Remove BLUE Deployment and Service**



Blue/Green Deployment Resources

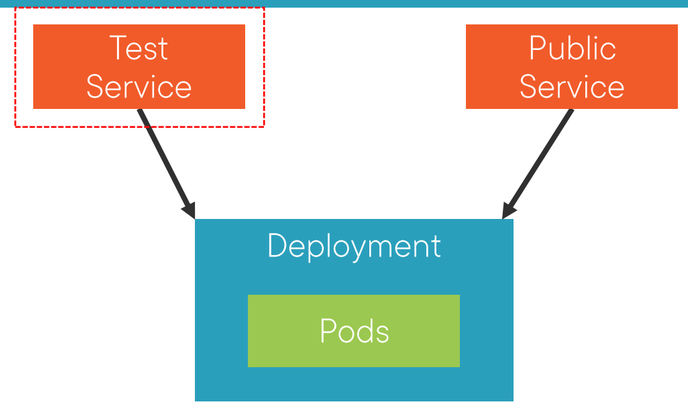


A Blue/Green Deployment involves 3 main Kubernetes resources:

- Test Service
- Public Service
- Deployment

Defining a Test Service

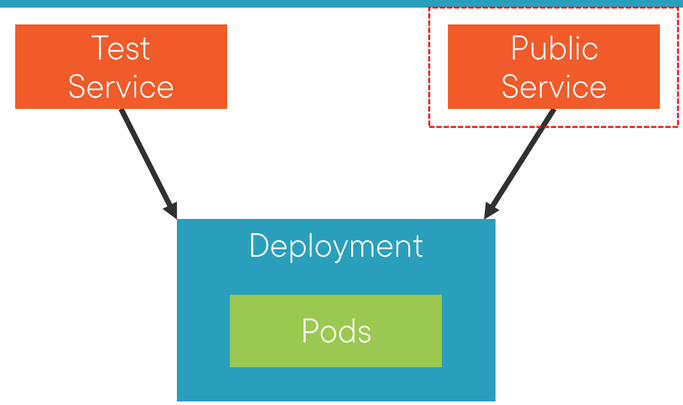
```
kind: Service
apiVersion: v1
metadata:
  name: blue-test-service
  labels:
    env: blue
spec:
  type: LoadBalancer
  selector:
    app: nginx
    role: blue
  ports:
    - port: 9000
      targetPort: 80
```



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

Defining a Public Service

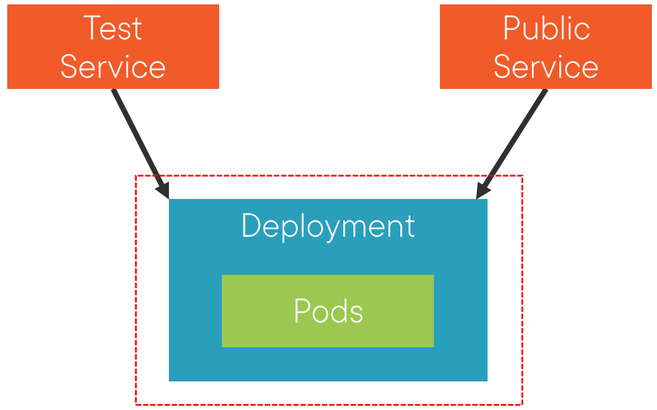
```
kind: Service
apiVersion: v1
metadata:
  name: public-service
  labels:
    env: prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    role: blue
  ports:
    - port: 80
      targetPort: 80
```



- Environment for the service (prod 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: blue-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
      role: blue
  template:
    metadata:
      labels:
        app: nginx
        role: blue
    spec:
      containers:
      - name: blue
```



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

Changing From Blue to Green

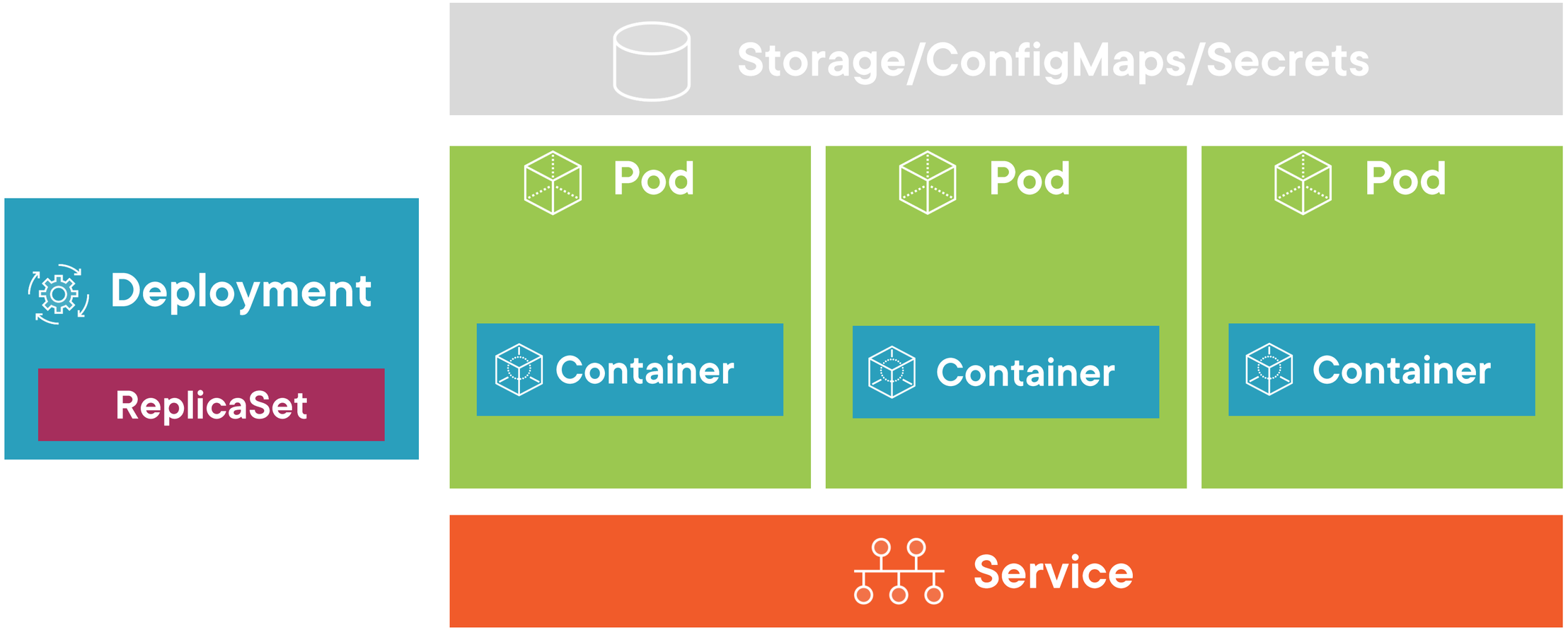
Once a GREEN deployment has been successfully rolled out and tested, change the public service's selector to "green"


```
# 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

Working with Canary Deployments

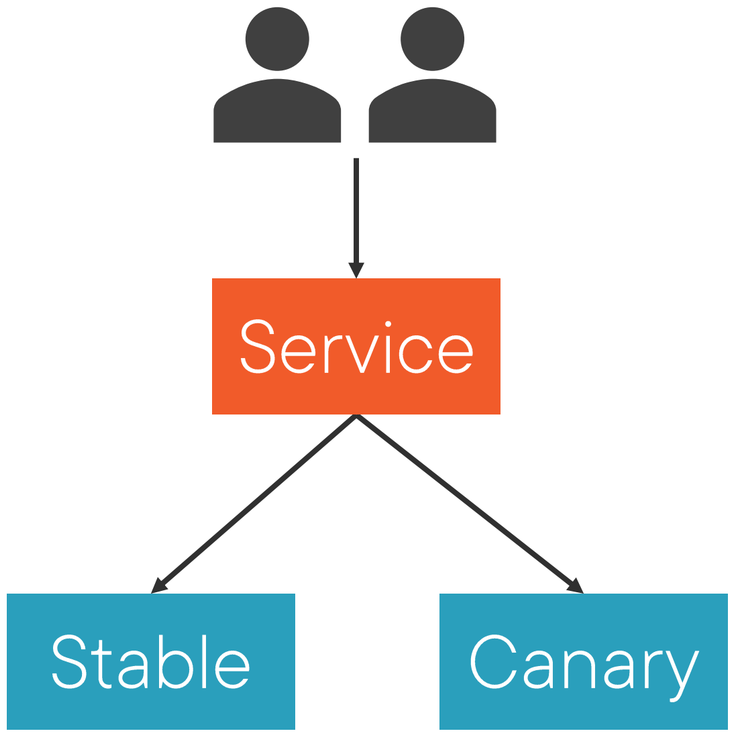
Canary Deployments and Services



"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

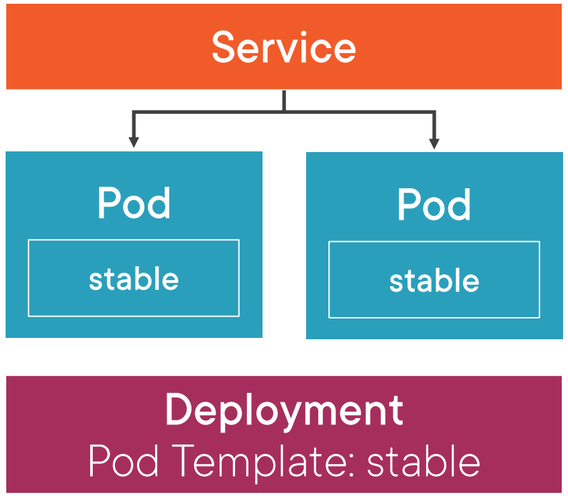
Canary Deployment runs alongside the existing stable Deployment

Canary Deployment is setup to receive minimal traffic

Canary Deployments

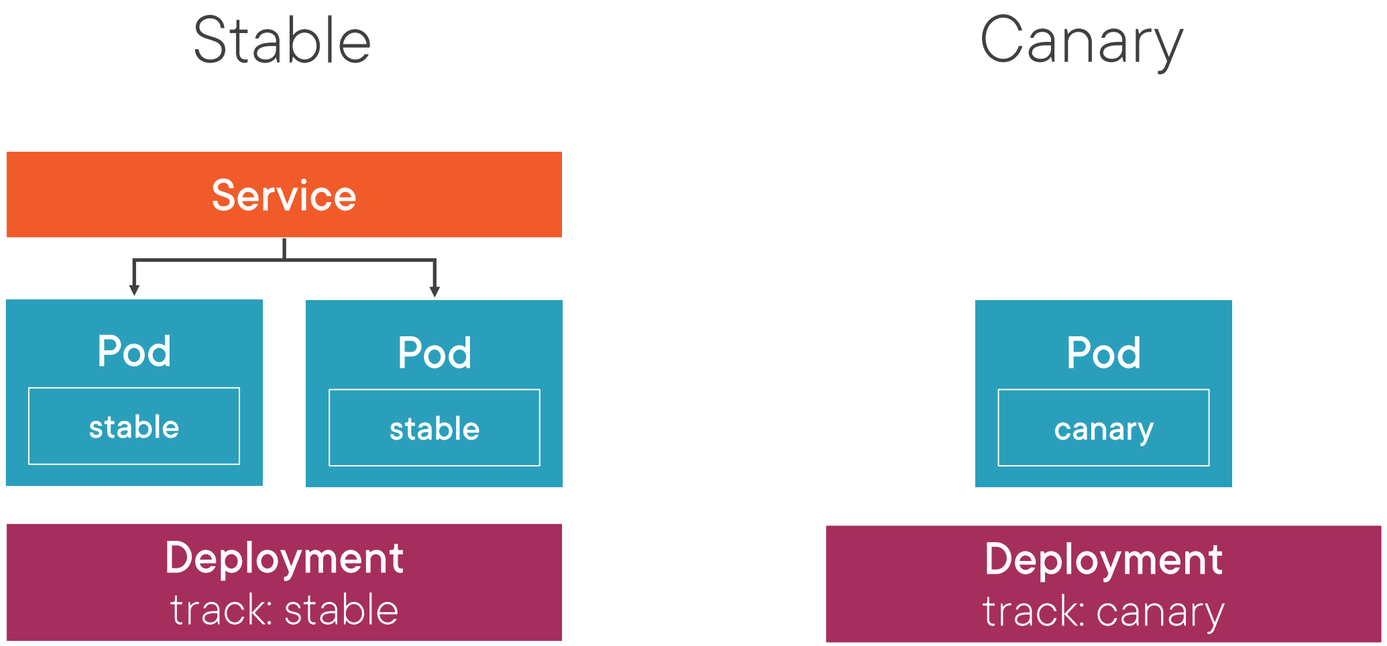
1 **Create Stable Deployment and Service**

Stable



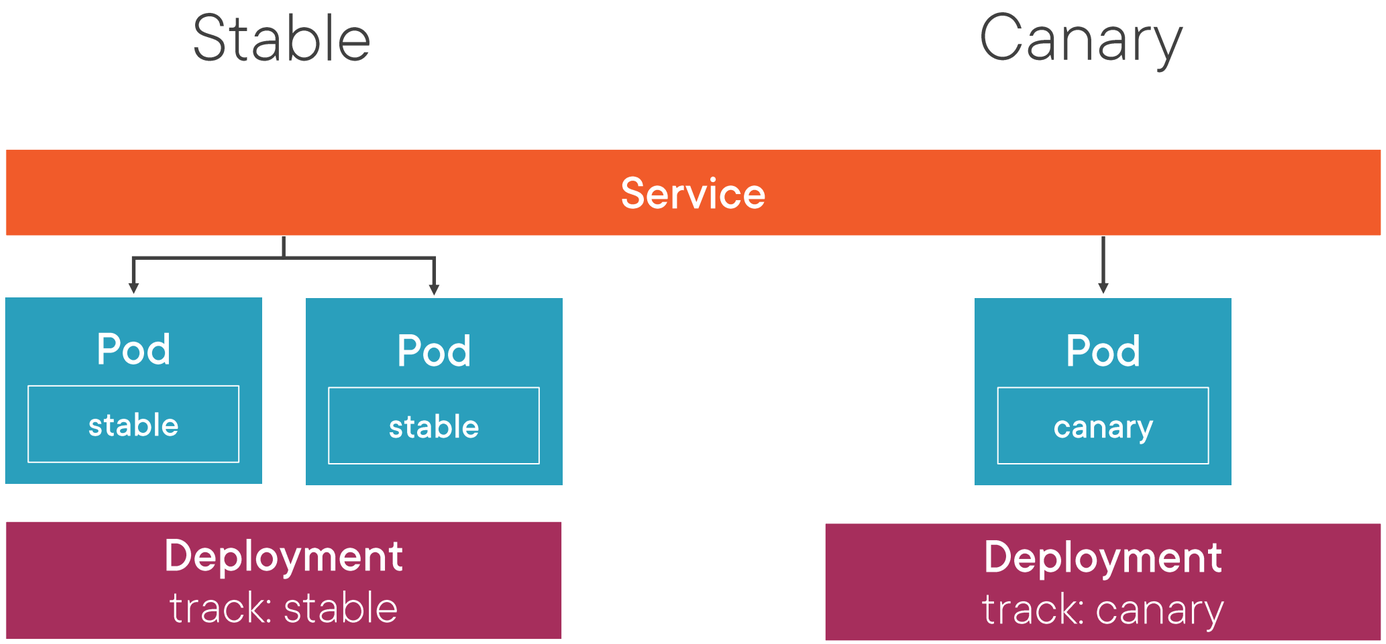
Canary Deployments

2 **Create Canary Deployment**



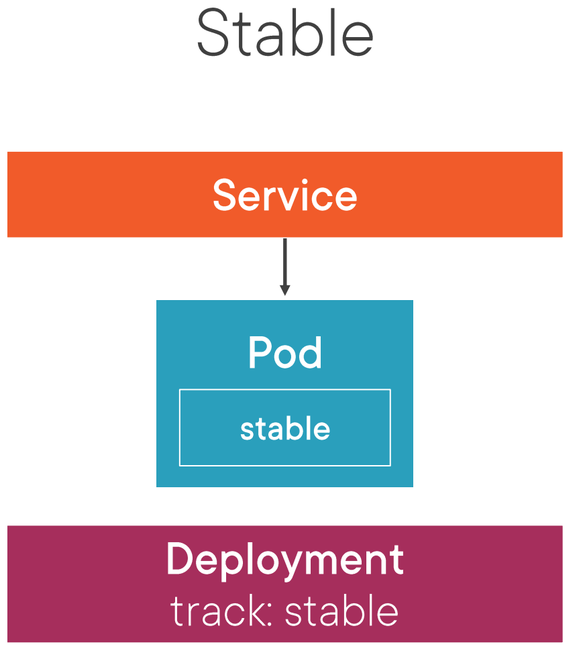
Canary Deployments

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

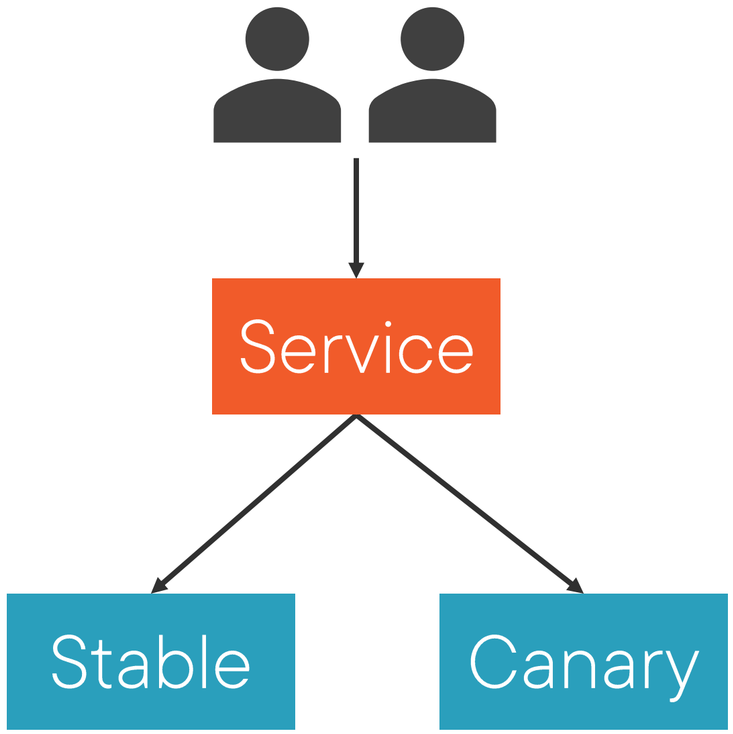


Canary Deployments

4 **Canary becomes stable Deployment**



Canary Deployment Resources

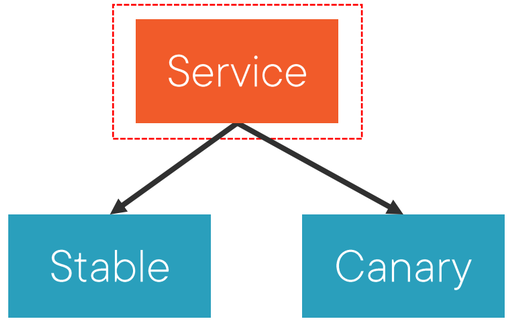


A Canary Deployment involves 3 main Kubernetes resources:

- Service
- Stable Deployment
- Canary Deployment

Defining a Service

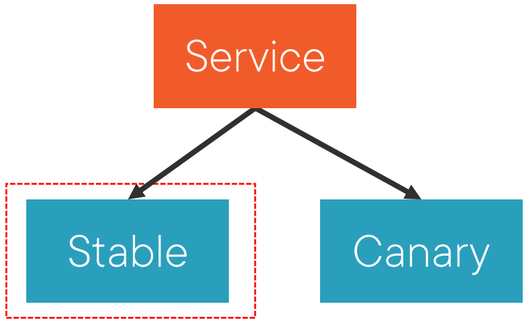
```
kind: Service
apiVersion: v1
metadata:
  name: stable-service
  labels:
    app: customer-app
spec:
  type: LoadBalancer
  selector:
    app: customer-app
  ports:
    - port: 80
      targetPort: 80
```



◀ **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: customer-app
      track: stable
  template:
    metadata:
      labels:
        app: customer-app
        track: stable
    spec:
      containers:
      - name: stable-app
        image: stable-app
```

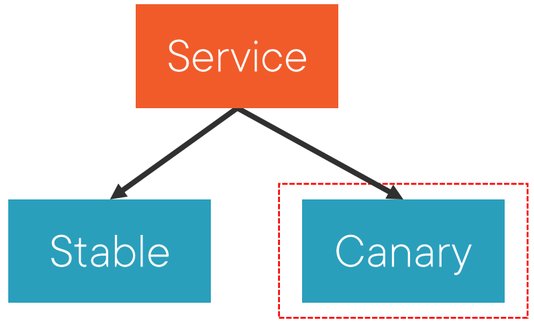


- Create stable replicas

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

Defining a Canary Deployment

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



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

- Pod labels (recall that app:customer-app 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]
```

Canary Deployments in Action

Exam Scenarios – Task 1

```
Cluster: ckad0012
Namespace: dev
Doc links: Services, Deployments
```

Task

1. Create a Deployment named **nginx-deploy**. The Deployment should use the **nginx:stable-alpine** image and create **4** replicas.

2. Create a **NodePort** service named **nginx-svc** and associate it with the Pods created by the **nginx-deploy** Deployment. The service should expose port **9000** and a target port of 80.

3. Edit the **nginx-deploy** Deployment and change the number of replicas to **6**.

4. Ensure the service is associated with the **nginx-deploy** Pods by running the following command and checking that it returns HTML content. Replace **<node-port>** with the **nginx-svc** node port value.

   ```
   $ curl localhost:<node-port-value>
   ```

Exam Scenarios – Task 2

```
Cluster: ckad0012
Namespace: dev
Doc links: Services, Deployments
```

Task

The current folder contains Kubernetes YAML files to create Blue/Green deployments and services. Images you'll use are also available on the system. View the files in the current folder using:

```
$ ls -l
```

1. View the selectors in the **blue.svc.yml** and **green.svc.yml** files. Add a **role: blue** selector to **public.svc.yml**.

2. Create the resources in Kubernetes using the YAML files available in the current folder. Run the following command to verify that the **blue** Pods are accessible using the public service.

   ```
   $ curl localhost
   ```

3. Change the public service's **role** selector from **blue** to **green**. Run the following command to verify that the **green** Pods are accessible using the public service.

   ```
   $ curl localhost
   ```

Exam Scenarios – Task 3

```
Cluster: ckad0012
Namespace: dev
Doc links: Services, Deployments
```

Task

Your system has **ckad:stable** and **ckad:canary** images available and the current folder contains YAML files to be used to complete the task.

1. Create resources in Kubernetes using the YAML files available in the current folder. List all the running Pods.

2. Scale the **stable** Deployment to **6** replicas and the **canary** Deployment replicas to **2**.

3. Modify the **canary** Deployment so that the Service will direct traffic to the Pods.

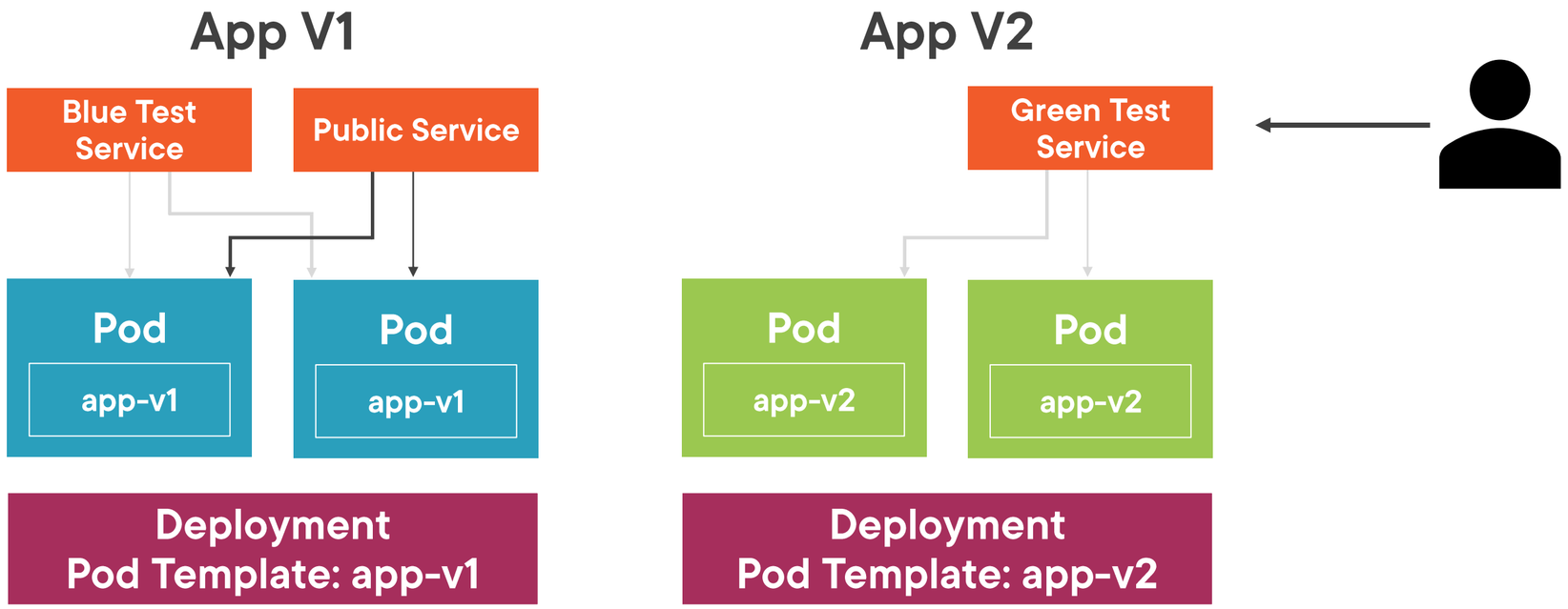
4. Create a temporary Pod named **temp-pod** as an interactive TTY. Use the **alpine** image for the Pod and set **restart** to **Never**.

5. Install **curl** into the **temp-pod** container using **apk add curl**, and run the following command until the output from a canary Pod is displayed:

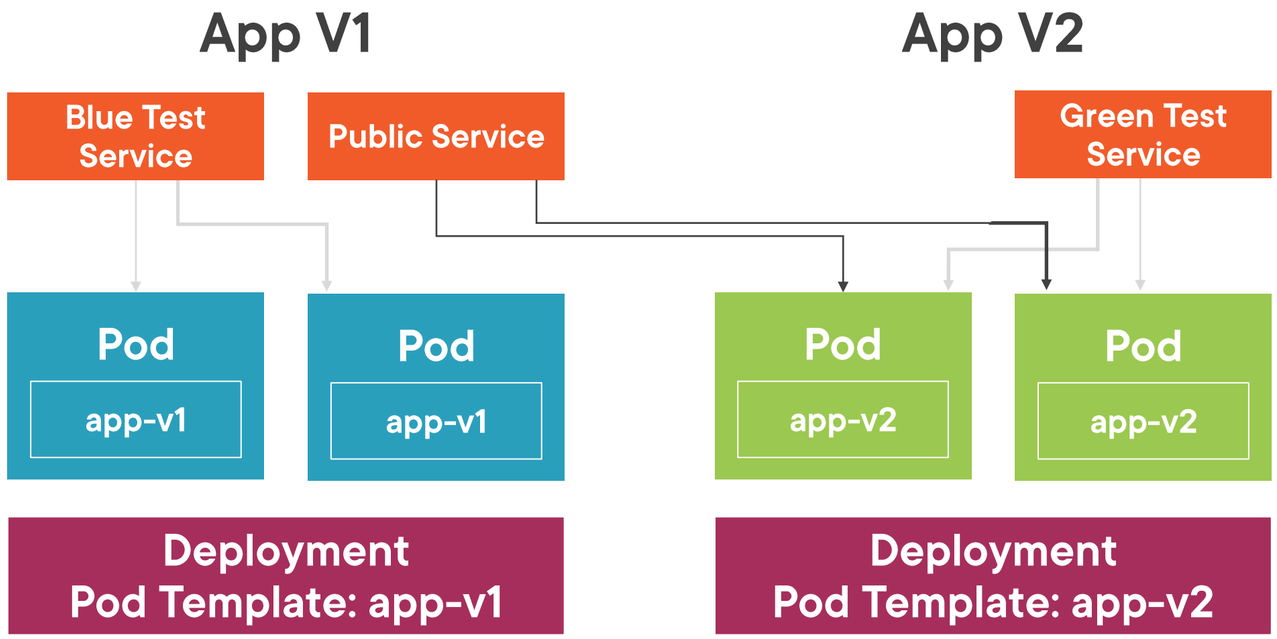
   ```
   $ curl <service-cluster-ip>
   ```

Recap and Test Yourself

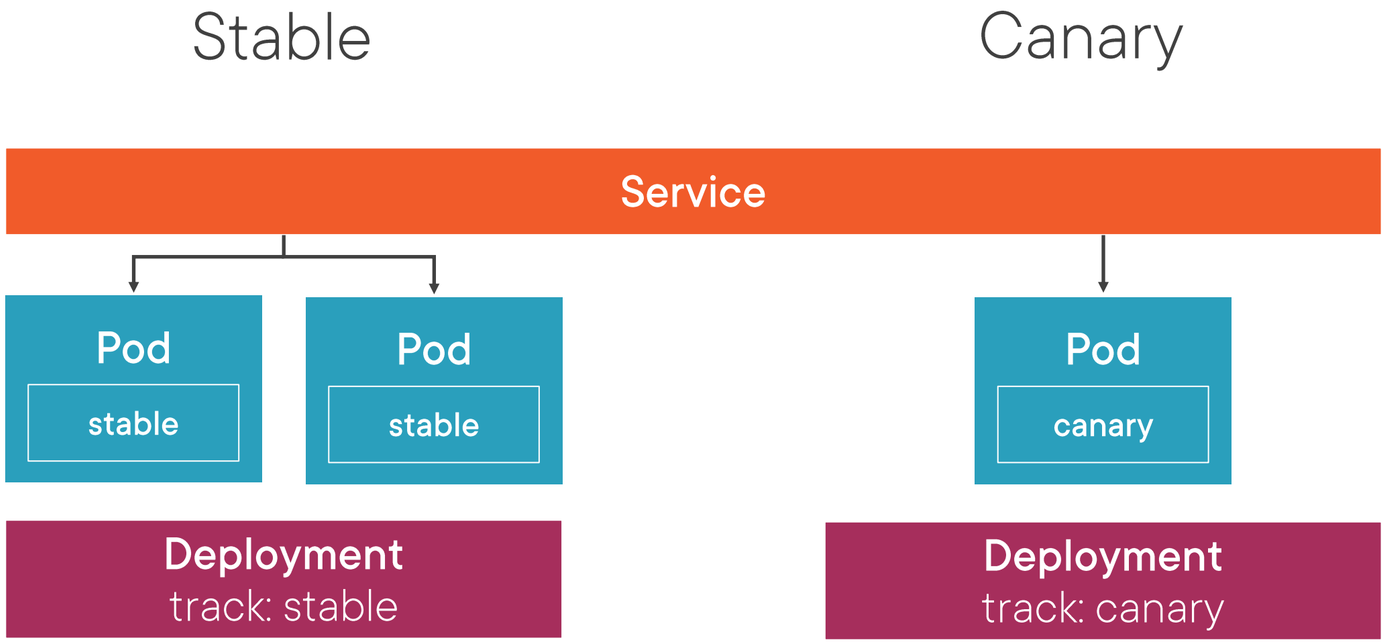
Blue/Green Deployments



Blue/Green Deployments



Canary Deployments



Top Three Take-home Points

Deployments can be created imperatively or declaratively.

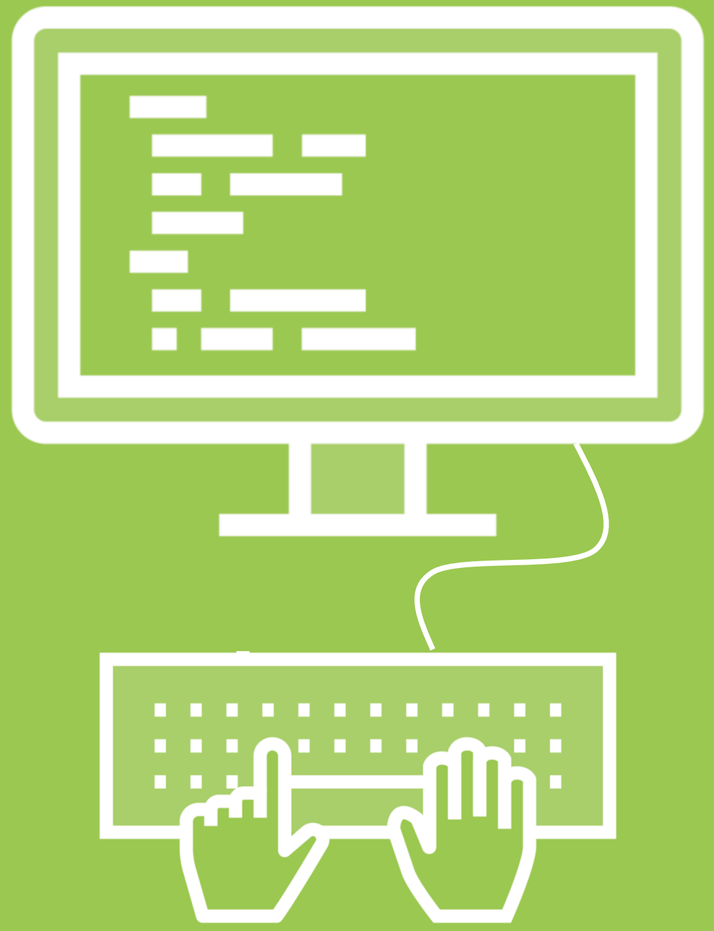
Blue/Green Deployments check the viability of a deployment before it's publicly available.

Canary Deployments deploy new versions of applications next to stable production versions.

Key kubectl Commands

Key kubectl commands to know for the exam.

```
k create deploy [deployment-name] --image=[image-name]:[tag]
  --dry-run=client -o yaml > deploy.yaml

k create –f ./[folder-name]

k set selector svc [service-name] 'key=value'

k scale deploy [deployment-name] --replicas=4

k set image deploy [deployment-name] [image-name]

KUBE_EDITOR="[editor-name]" k edit deploy [deployment-name]
```



GitHub Repo

https://github.com/nigelpoulton/ckad