

Creating Services



Dan Wahlin

WAHLIN CONSULTING

@danwahlin www.codewithdan.com



Module Overview

Services Core Concepts

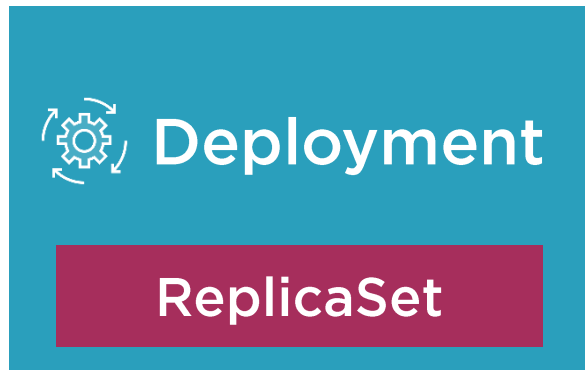
Service Types

Creating a Service with kubectl

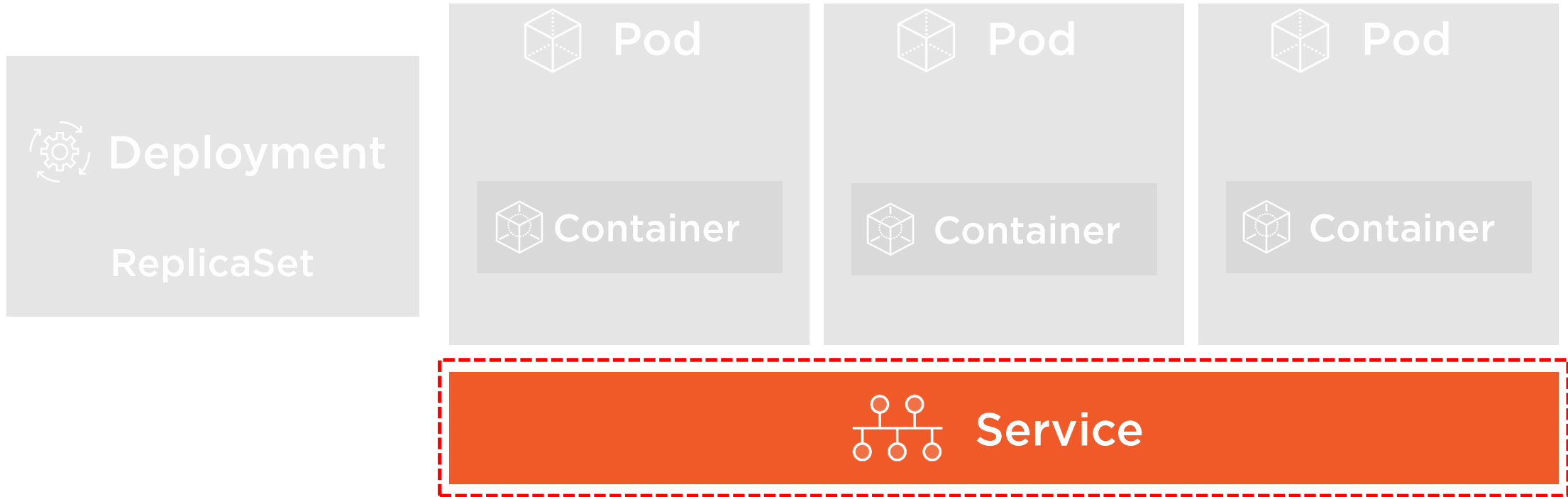
Creating a Service with YAML



You Are Here



You Are Here



Services Core Concepts

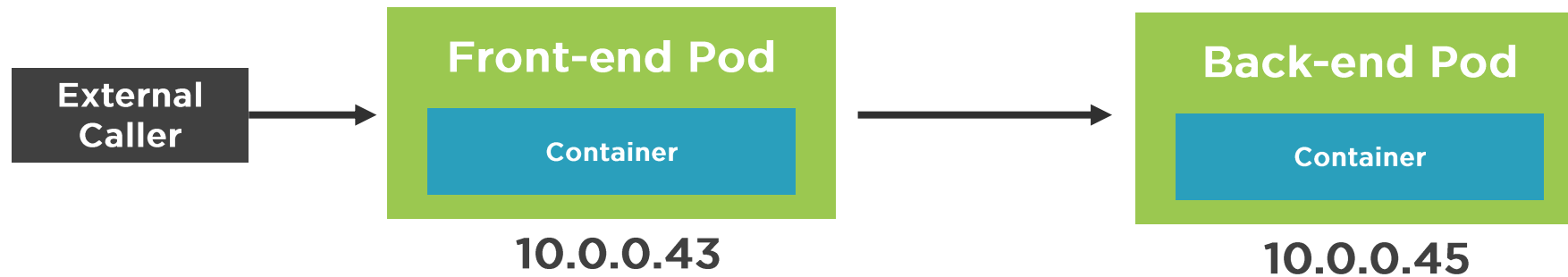


A Service provides a single point of entry for accessing one or more Pods



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!



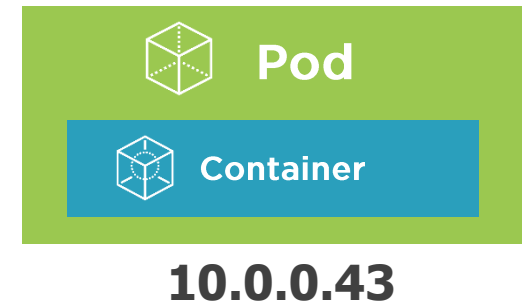
Pods are "mortal" and may only live a short time (ephemeral)

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

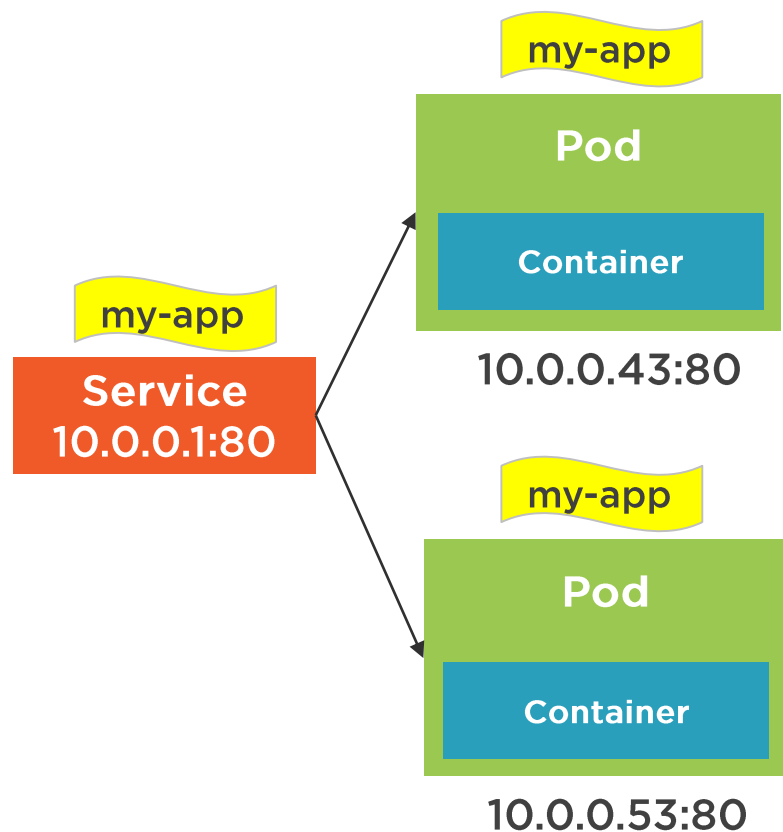
Pods can be horizontally scaled so each Pod gets its own IP address

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

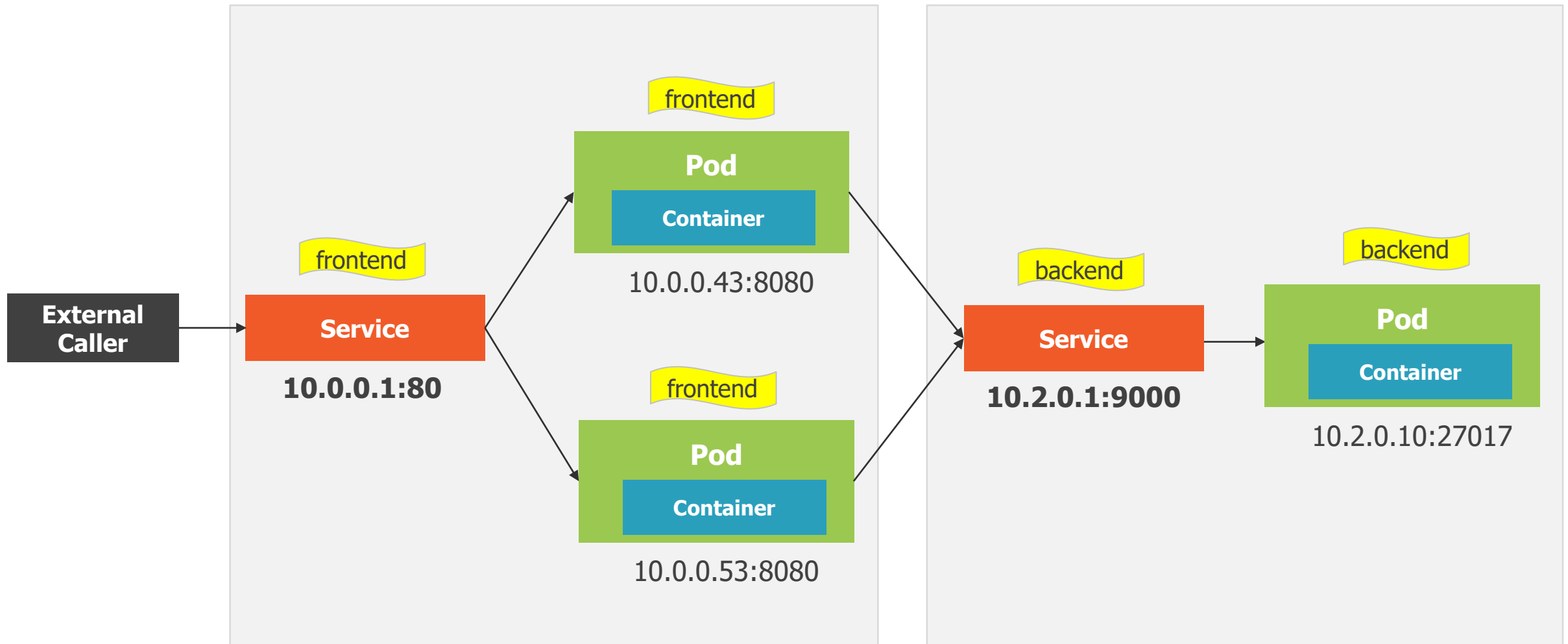
Layer 4 (TCP/UDP over IP)

Services are not ephemeral

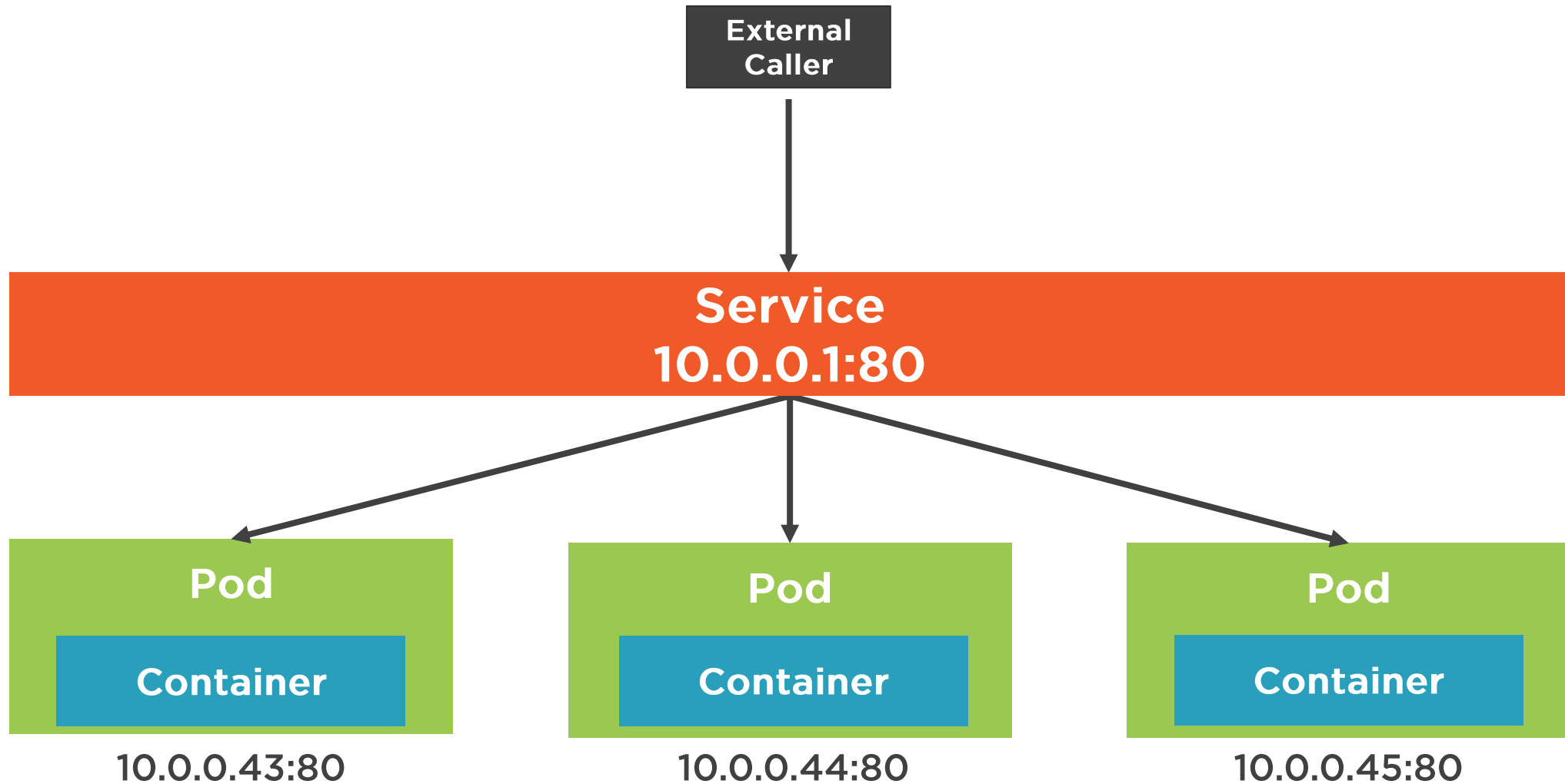
Creates endpoints which sit between a Service and Pod



Calling Services



Services and Pod Load Balancing



Service Types



Service Types



Service

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

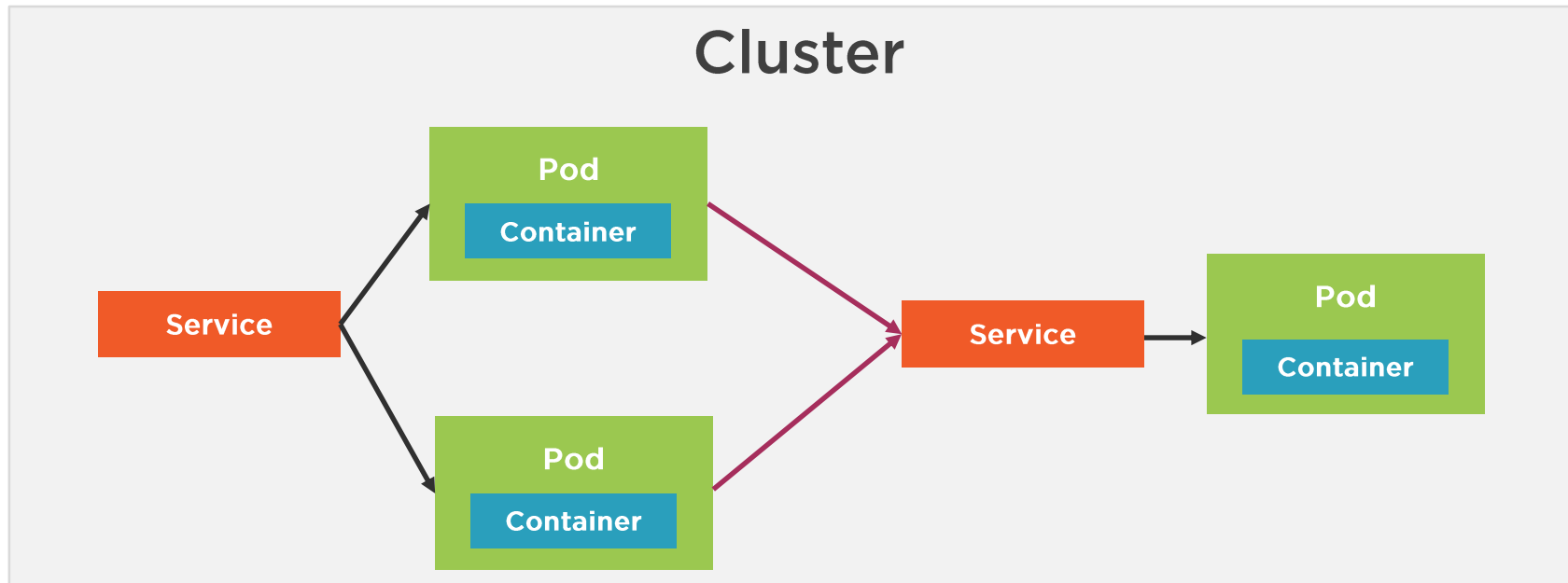


ClusterIP Service

Service IP is exposed internally within the cluster

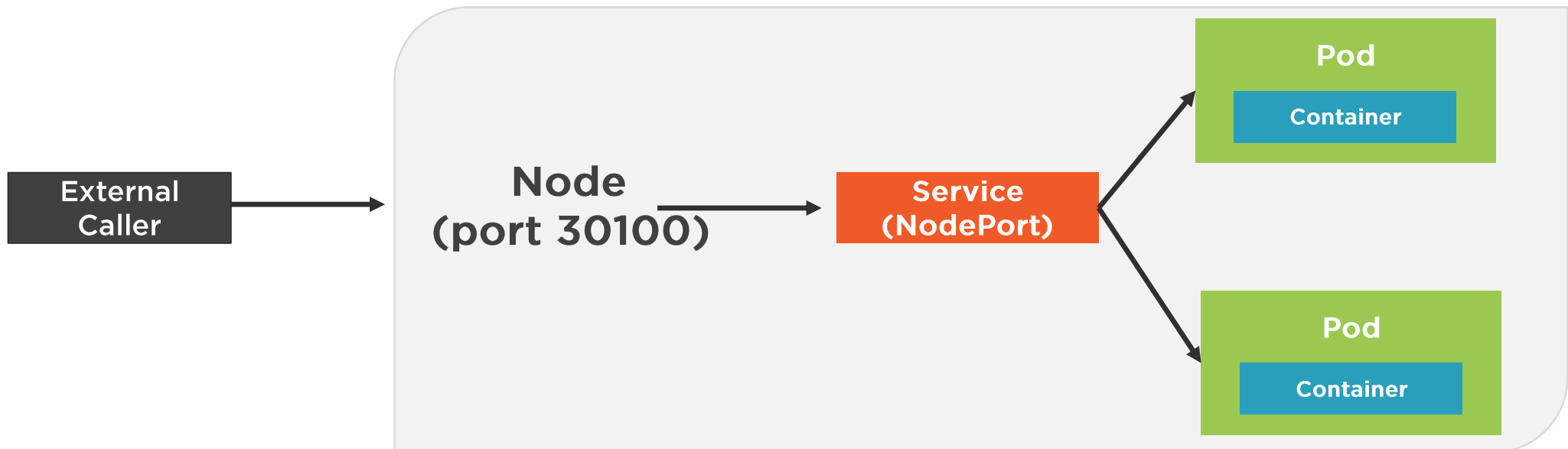
Only Pods within the cluster can talk to the Service

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



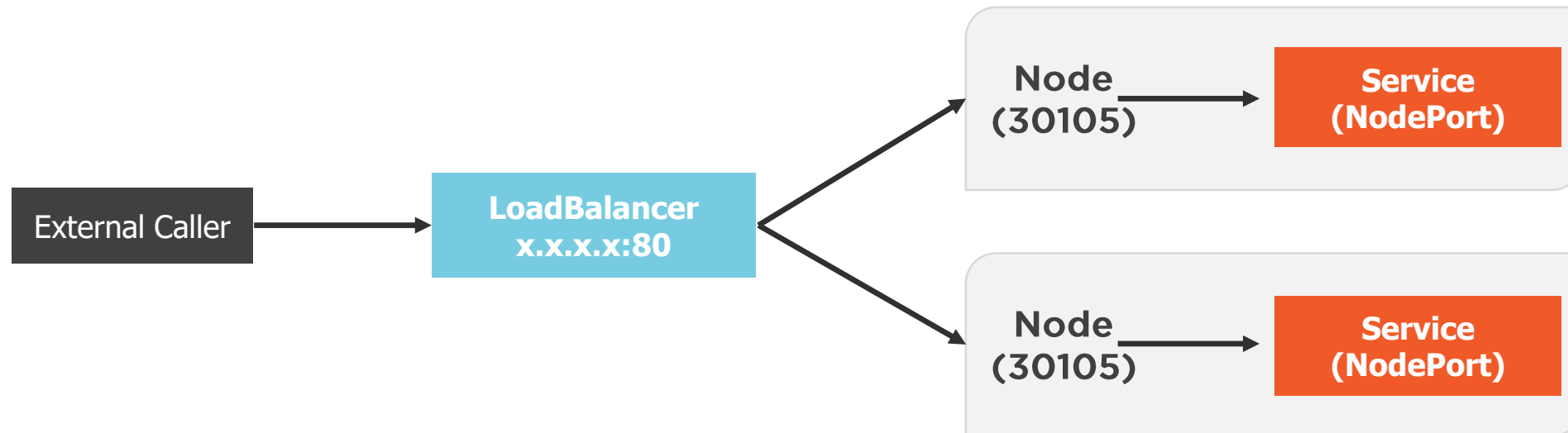
LoadBalancer Service

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

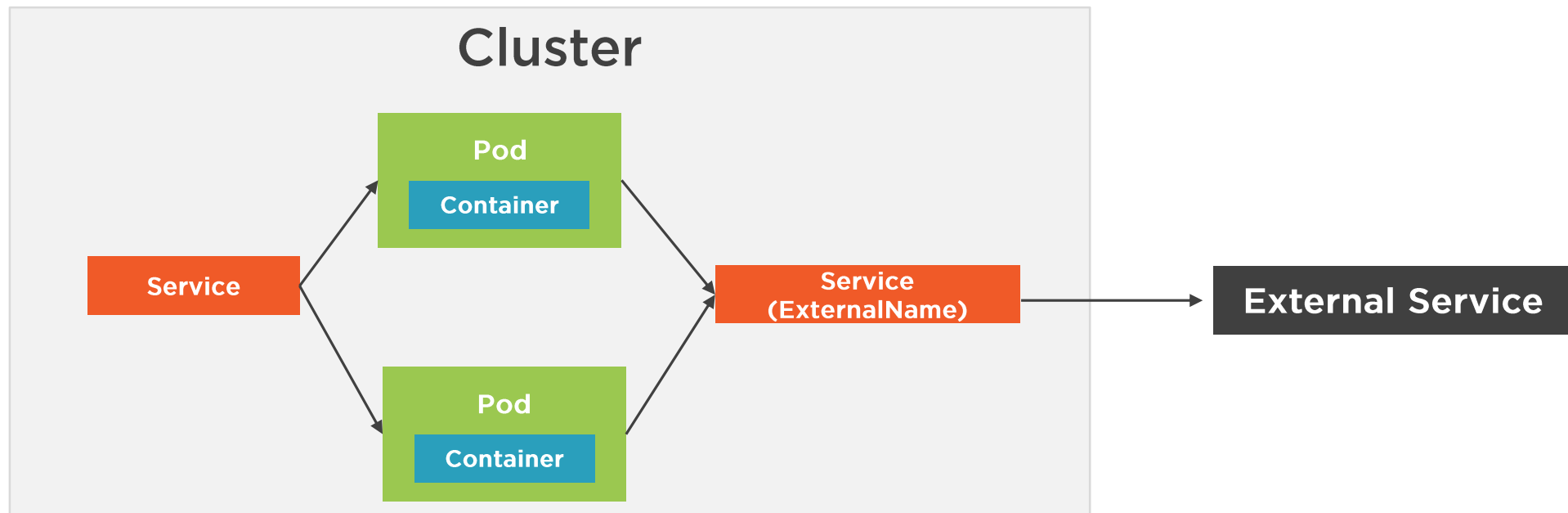


ExternalName Service

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

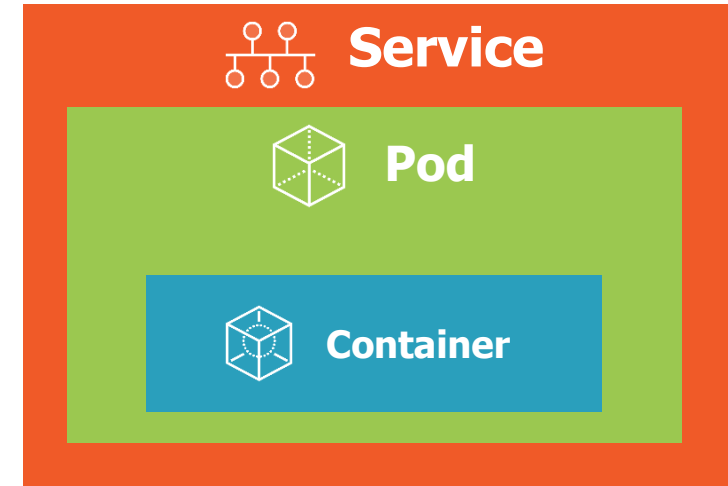


Defining a Service with YAML



Service

+ **kubectl** =



```
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`**




```
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
```

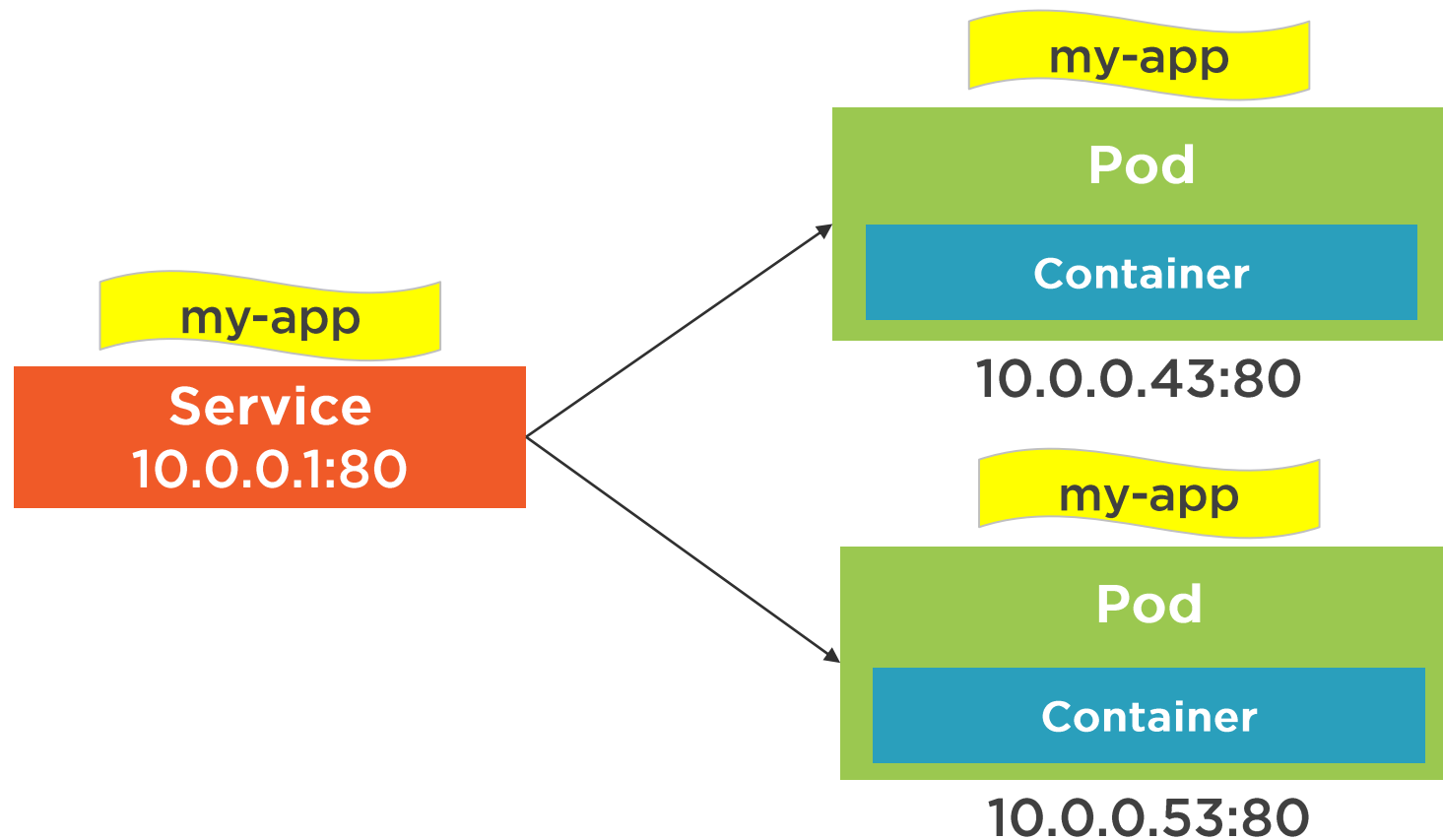
Testing a Service and Pod with curl

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)

