kubernetes

Running Applications

Kubernetes

Course Objectives

In this module, you will learn:

- Pods
- Replica-sets
- Deployments
- Services

kubernetes

Pods

Running Applications

Pods

- Pods are the smallest unit of deployment
- Usually pods represent a single container
- But there are certain use cases where we may need one pod to have more than one containers.

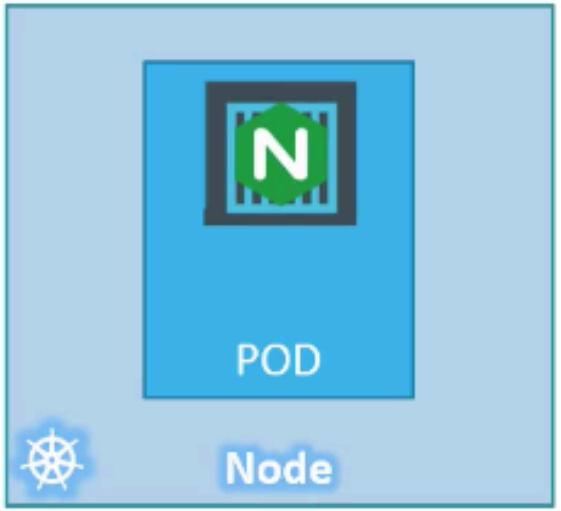
Create Pods using kubectl

```
kubectl run nginx --image nginx
```

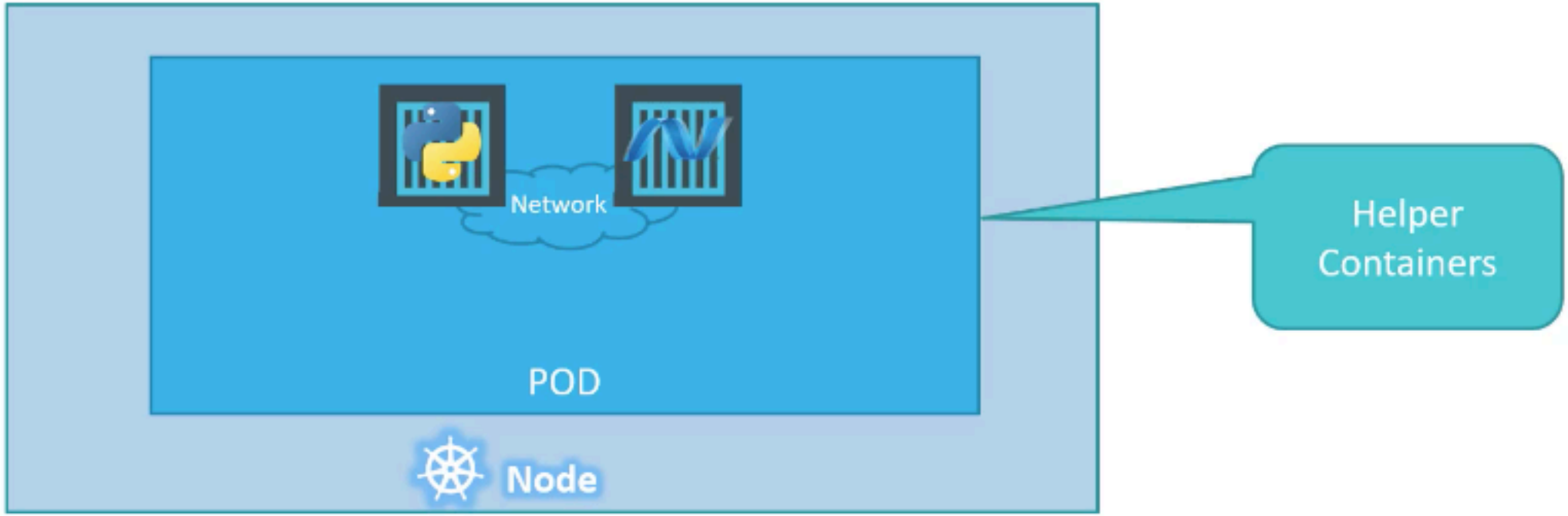
```
kubectl get pods
```

```
C:\Kubernetes>kubectl get pods
NAME                     READY     STATUS              RESTARTS   AGE
nginx-8586cf59-whssr     0/1       ContainerCreating   0          3s
```

```
C:\Kubernetes>kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-8586cf59-whssr     1/1       Running   0          8s
```



Multi-Container PODs



kubernetes

Creating Pods with YAML

Running Applications

YAML in Kubernetes - Structure

pod-definition.yml

```
apiVersion:
kind:
metadata:



spec:
```

YAML in Kubernetes

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:


spec:
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

YAML in Kubernetes

pod-definition.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
      app: myapp
spec:
```

String

String

Dictionary

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

YAML in Kubernetes

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

List/Array

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

YAML in Kubernetes

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

```
kubectl create -f pod-definition.yml
```

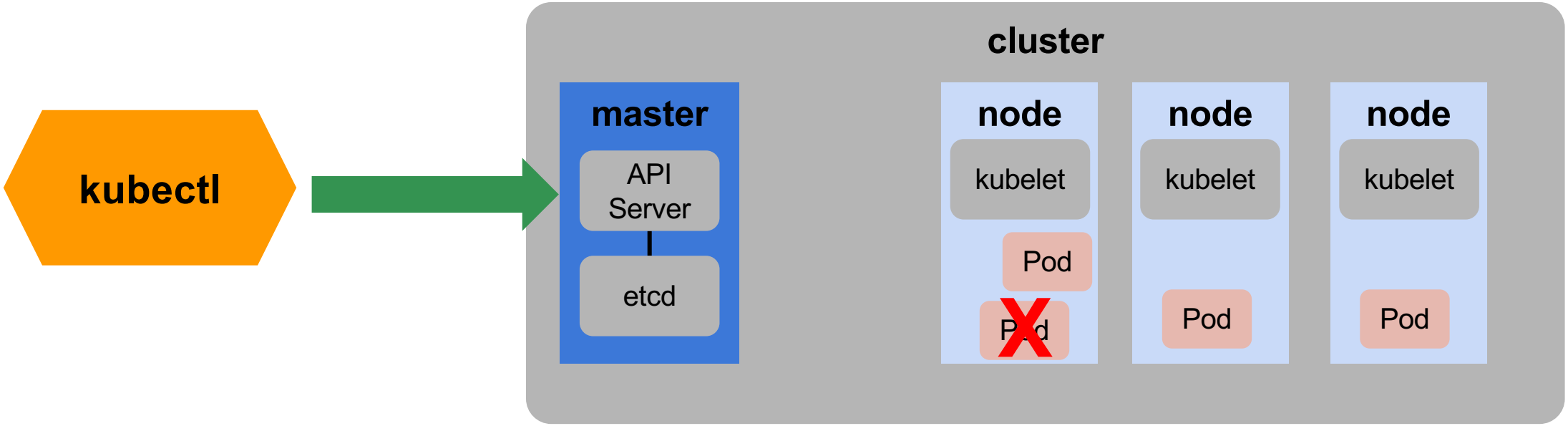
kubernetes

Replica Set

Running Applications

Kubernetes Terms

- Replication controllers are used to create multiple instances of a pod
- Guarantee pods are healthy and the right number exist
- It replaces any pod that fails



replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
```

```
> kubectl create -f replicaset-definition.yml
replicaset "myapp-replicaset" deleted
```

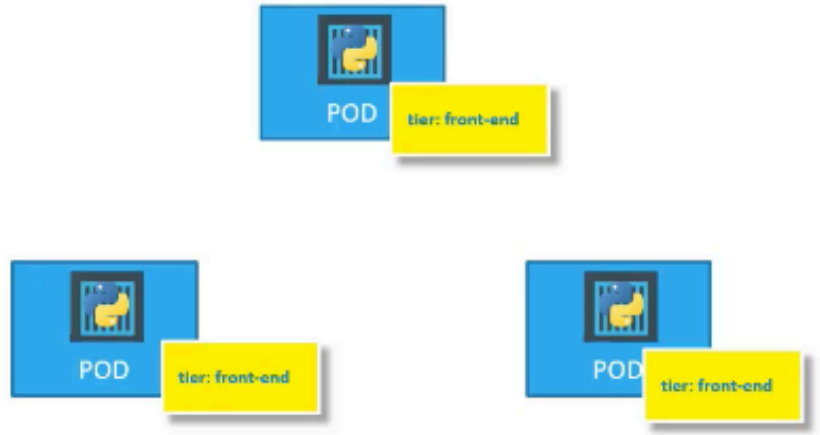
```
> kubectl get replicaset
NAME              DESIRED   CURRENT   READY     AGE
myapp-replicaset  3         3         3         19s
```

```
> kubectl get pods
NAME                      READY     STATUS    RESTARTS   AGE
myapp-replicaset-9dd19    1/1       Running   0          45s
myapp-replicaset-9jtpx    1/1       Running   0          45s
myapp-replicaset-hq84m    1/1       Running   0          45s
```

Labels & Selectors

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
         app: myapp
         type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx

  replicas: 3
  selector:
    matchLabels:
       type: front-end
```



Scaling in Replica Set


```
> kubectl replace -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 -f replicaset-definition.yml
```

```
> kubectl scale --replicas=6 replicaset myapp-replicaset
```

TYPE NAME

replicaset-definition.yml

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: myapp-replicaset
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx
  replicas: 6
  selector:
    matchLabels:
      type: front-end
```

Commands

```
> kubectl create -f replicaset-definition.yml
```

```
> kubectl get replicaset
```

```
> kubectl delete replicaset myapp-replicaset
```

*Also deletes all underlying PODs

```
> kubectl replace -f replicaset-definition.yml
```

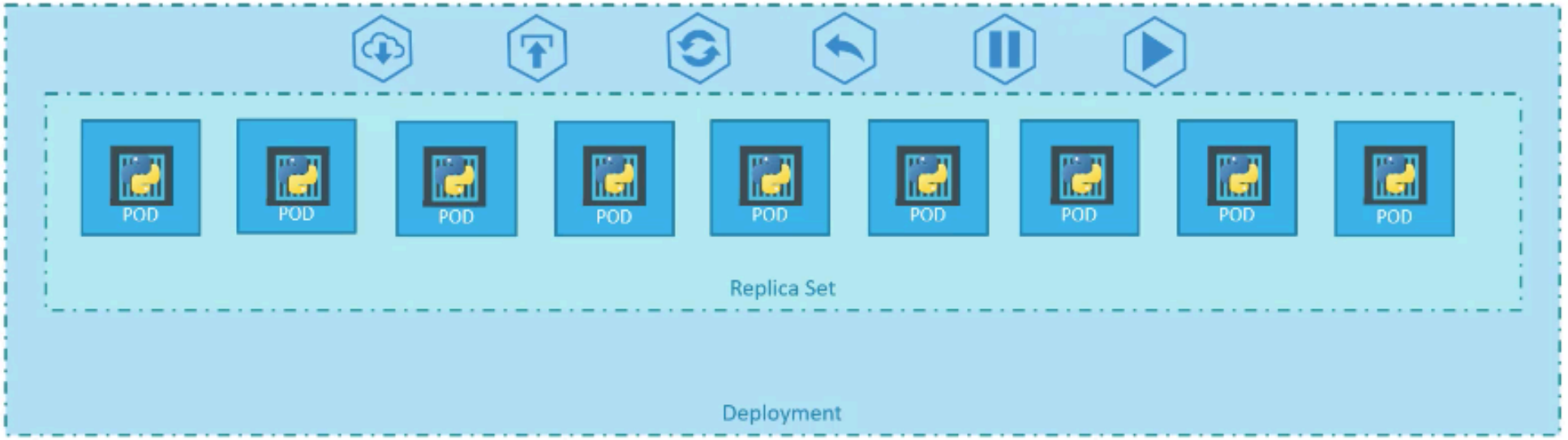
```
> kubectl scale -replicas=6 -f replicaset-definition.yml
```

kubernetes

Deployments

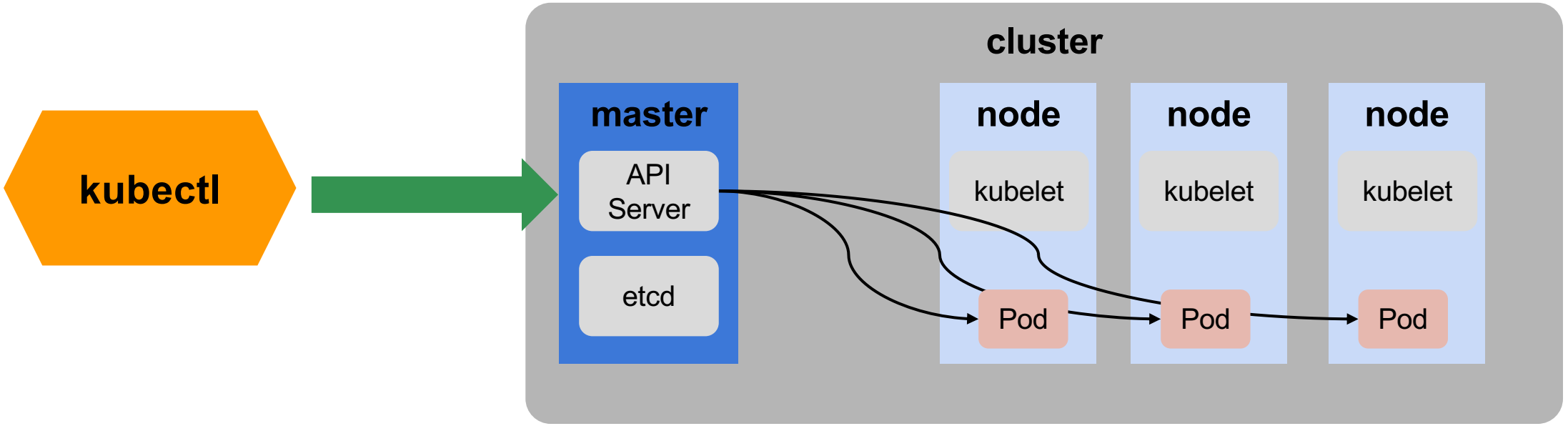
Running Applications

Understanding Deployments



Kubernetes Deployments

- Deployments are configurations that define service resources
- Use kubectl to send config files to the master
  - Can also send individual commands using kubectl, i.e.
  - kubectl create deployment hello-server --image=gcr.io/myproject/events:1.0
  - The master then decides how to deploy the pods



kubectl create deployment

- Deployments support declarative management of applications
  - For the following, Minikube should be running

```
$ kubectl get pods
No resources found in default namespace.
$
$ kubectl get deployments
No resources found in default namespace.
$
$ kubectl create deployment spaceinvaders --image drehnstrom/space-invaders --replicas=3
deployment.apps/spaceinvaders created
$
$ kubectl get deployments
NAME            READY   UP-TO-DATE   AVAILABLE   AGE
spaceinvaders   3/3     3            3           17s
$
$ kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
spaceinvaders-5bfc47ddf9-bkhqm   1/1     Running   0          26s
spaceinvaders-5bfc47ddf9-mv959   1/1     Running   0          26s
spaceinvaders-5bfc47ddf9-tv42b   1/1     Running   0          26s
$
```

Use `kubectl scale` to Add Instances

```
$ kubectl scale --replicas=6 deployment/spaceinvaders
deployment.apps/spaceinvaders scaled
$
$ kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
spaceinvaders-5bfc47ddf9-45wsm   1/1     Running   0          11s
spaceinvaders-5bfc47ddf9-b4jcq   1/1     Running   0          11s
spaceinvaders-5bfc47ddf9-bkhqm   1/1     Running   0          6m10s
spaceinvaders-5bfc47ddf9-mv959   1/1     Running   0          6m10s
spaceinvaders-5bfc47ddf9-rlg9g   1/1     Running   0          11s
spaceinvaders-5bfc47ddf9-tv42b   1/1     Running   0          6m10s
$
```

Deployments Combine Pods with Replica Sets

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: devops-deployment
  labels:
    <Some code omitted to save space>
spec:
  replicas: 3
  selector:
    <Some code omitted to save space>
  template:
    <Some code omitted to save space>
    spec:
      containers:
      - name: devops-demo
        image: drehnstrom/devops-demo:latest
        ports:
        - containerPort: 8080
```

Kind of resource: Deployment

We want 3 replicas of the pod

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

Can Control the Resources Your Pods require and are allowed to consume

```
apiVersion: apps/v1
kind: Deployment

***CODE OMITTED FOR SPACE ***

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

The minimum amount of resources required for each pod

The maximum amount of resources a pod is allowed to consume

Creating a Deployment from a Configuration File

- Deploy a service based on a configuration file

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

- Show the running pods

```
kubectl get pods
```

- Show all the deployments

```
kubectl get deployments
```

- Show details of a deployment

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

Creating an Autoscaler

- To dynamically scale up and down, create an autoscaler
  - Specify min and max number of pods and some metric to monitor

```
kubectl autoscale deployment devops-deployment --min=5 --
max=10 --cpu-percent=60
```

Adding the Autoscaler to Configuration

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

Deleting Deployments and Resources

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

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

- Can also delete resources individually when created at the command line

```
kubectl delete hpa devops-autoscaler
kubectl delete services devops-loadbalancer
```

kubernetes

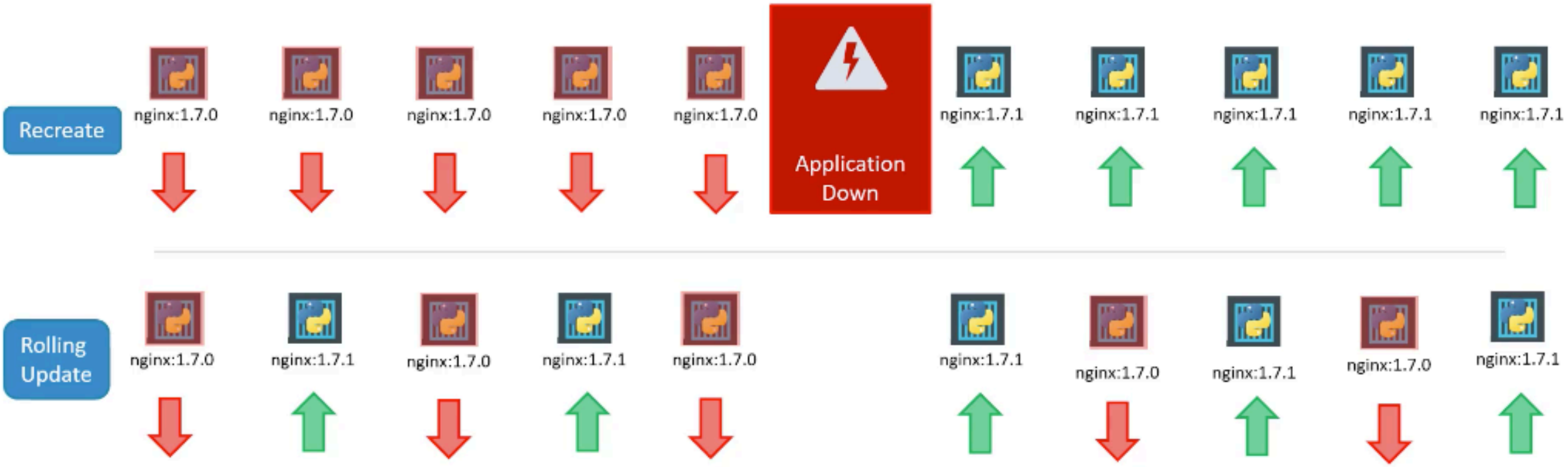
Deployments – Updates & Rollback

Running Applications

Rolling Updates

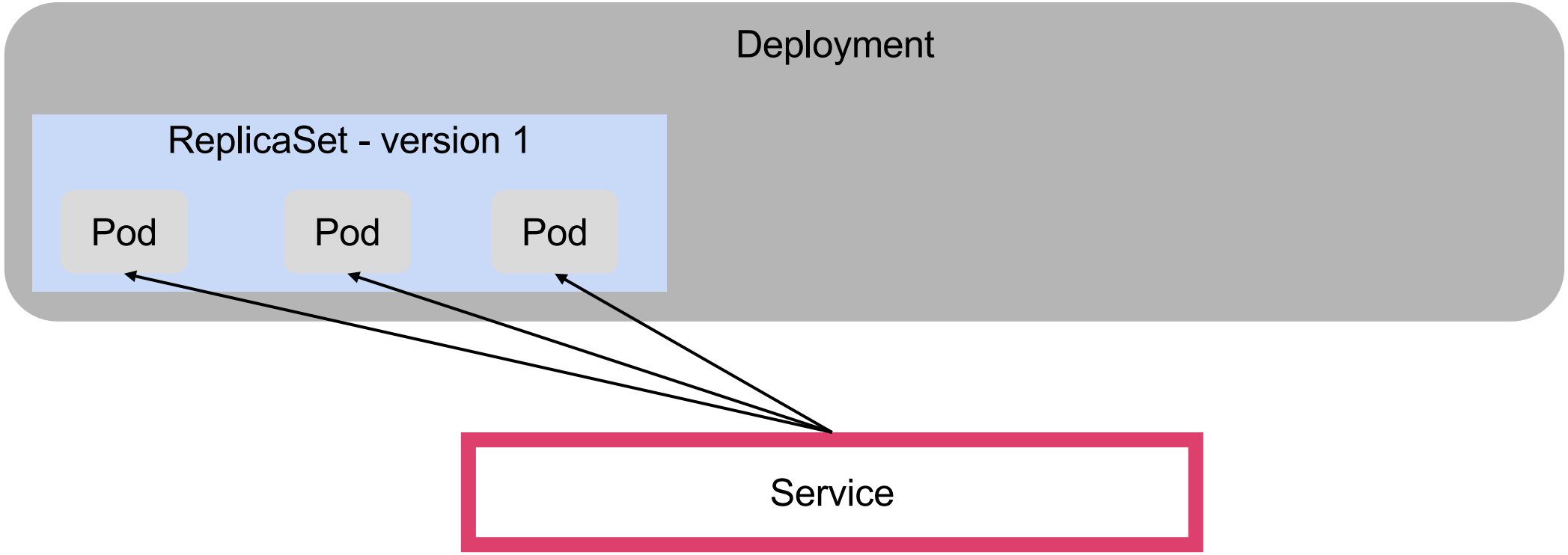
- Cloud services typically have multiple instances behind a load balancer
  - Multiple replicas of a pod in a Kubernetes cluster
- Rolling deployments update instances incrementally
  - One at a time, 10% at a time, etc.
  - Allows services to be updated with no downtime
- Supported by managed instance groups
- Supported by Kubernetes using the apply command
  - Simply change the container image and re-apply the configuration
  - Can also roll back the update

Deployment Strategy



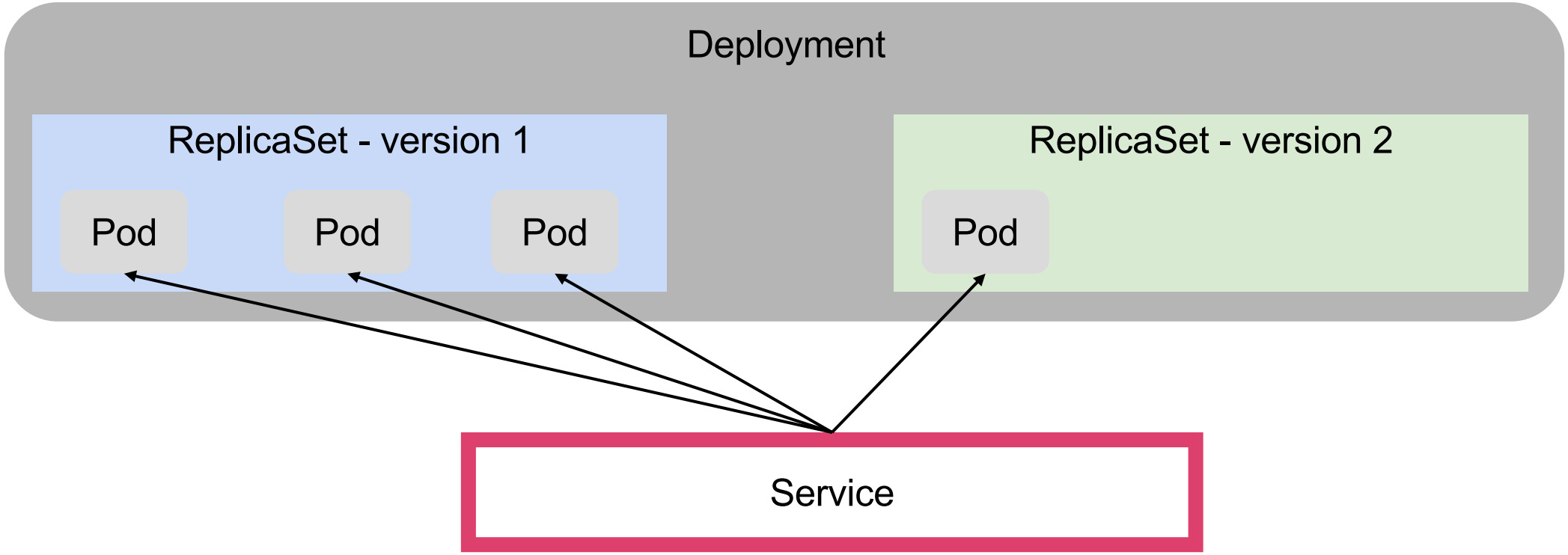
Rolling Update with Kubernetes

- Initially a deployment creates a single ReplicaSet



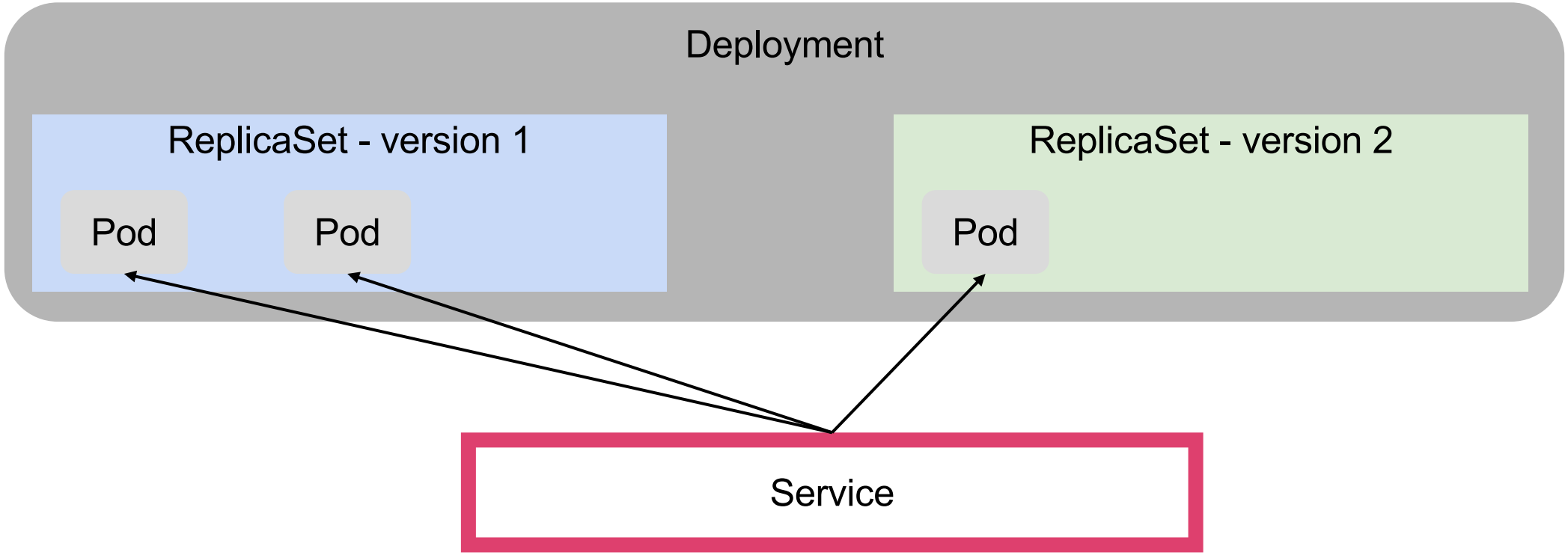
Rolling Update with Kubernetes (continued)

- When a container update is applied
  - A new ReplicaSet is created in the same deployment



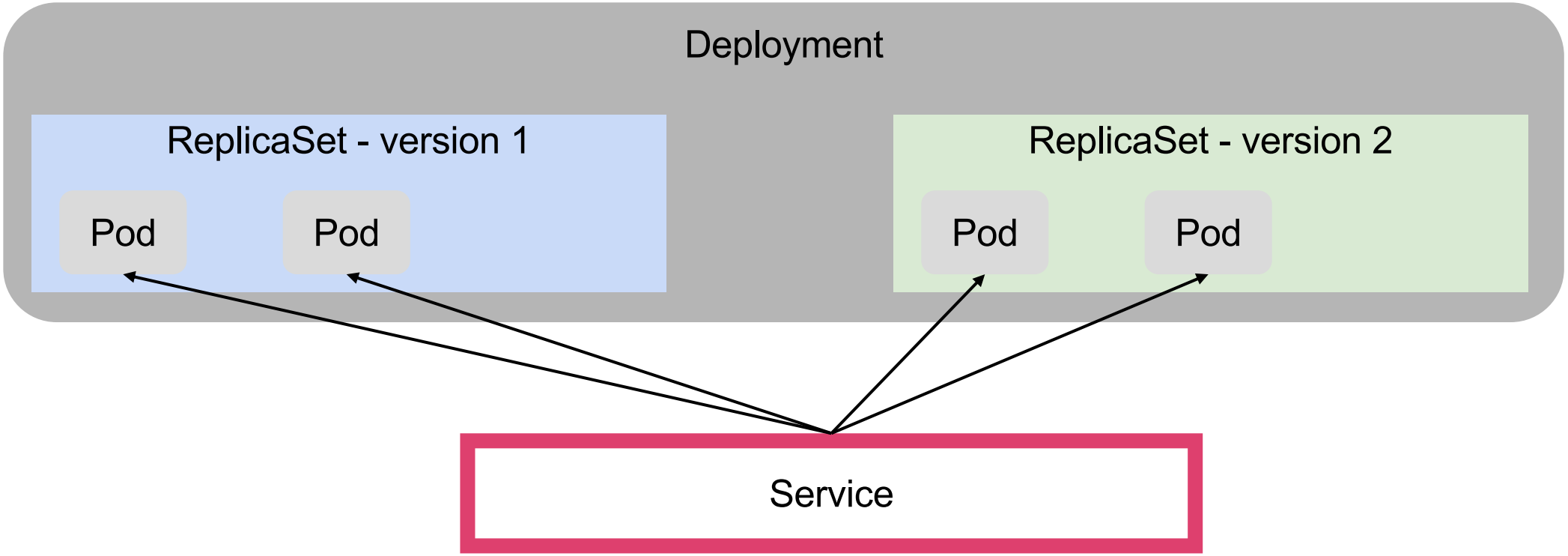
Rolling Update with Kubernetes (continued)

- A pod in the old ReplicaSet is deleted



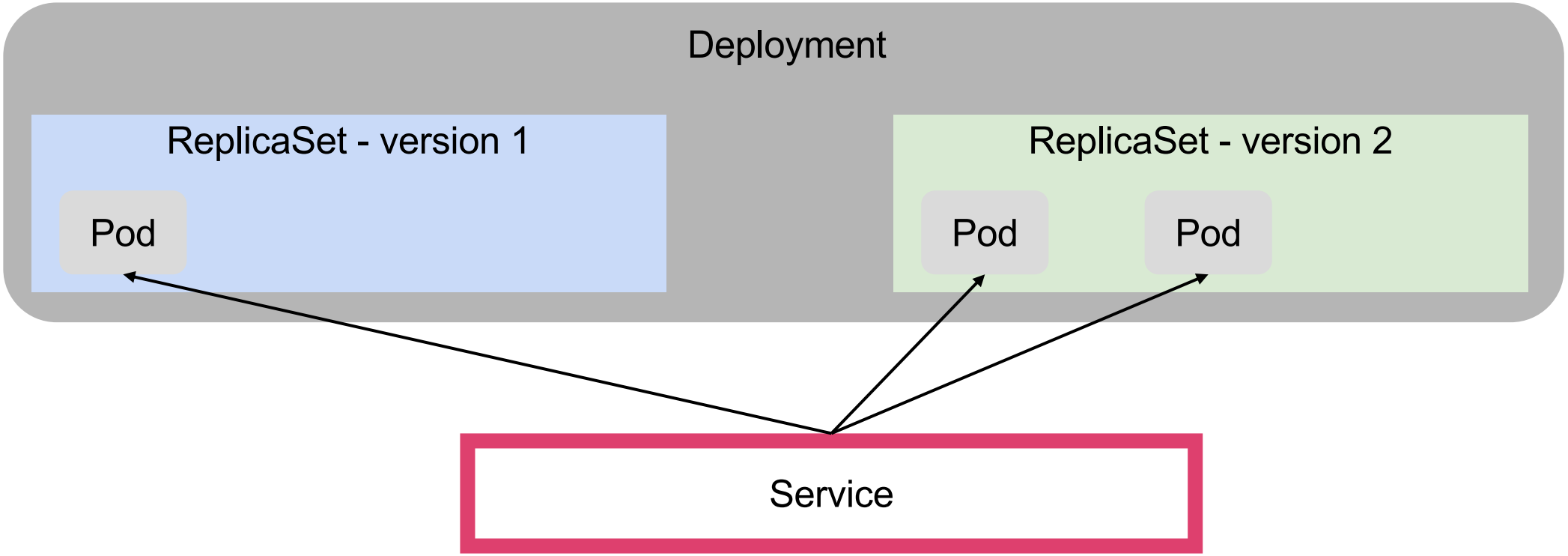
Rolling Update with Kubernetes (continued)

- This is repeated



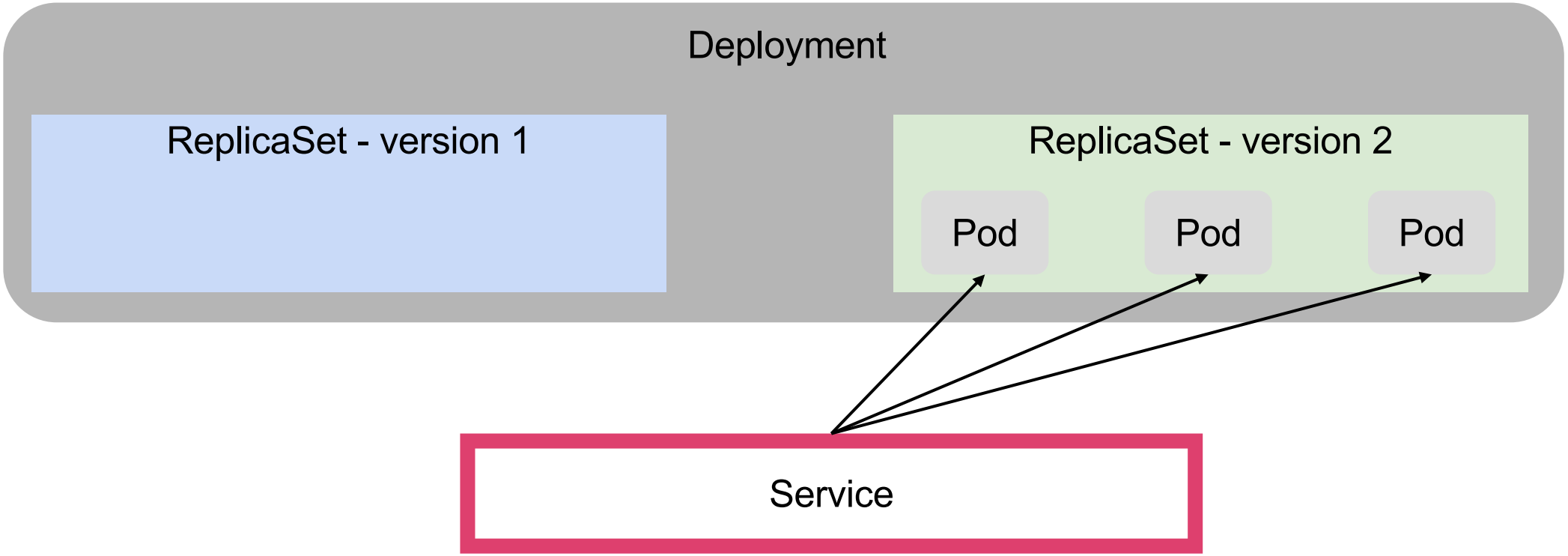
Rolling Update with Kubernetes (continued)

- This is repeated



Rolling Update with Kubernetes (continued)

- Until all pods have been updated

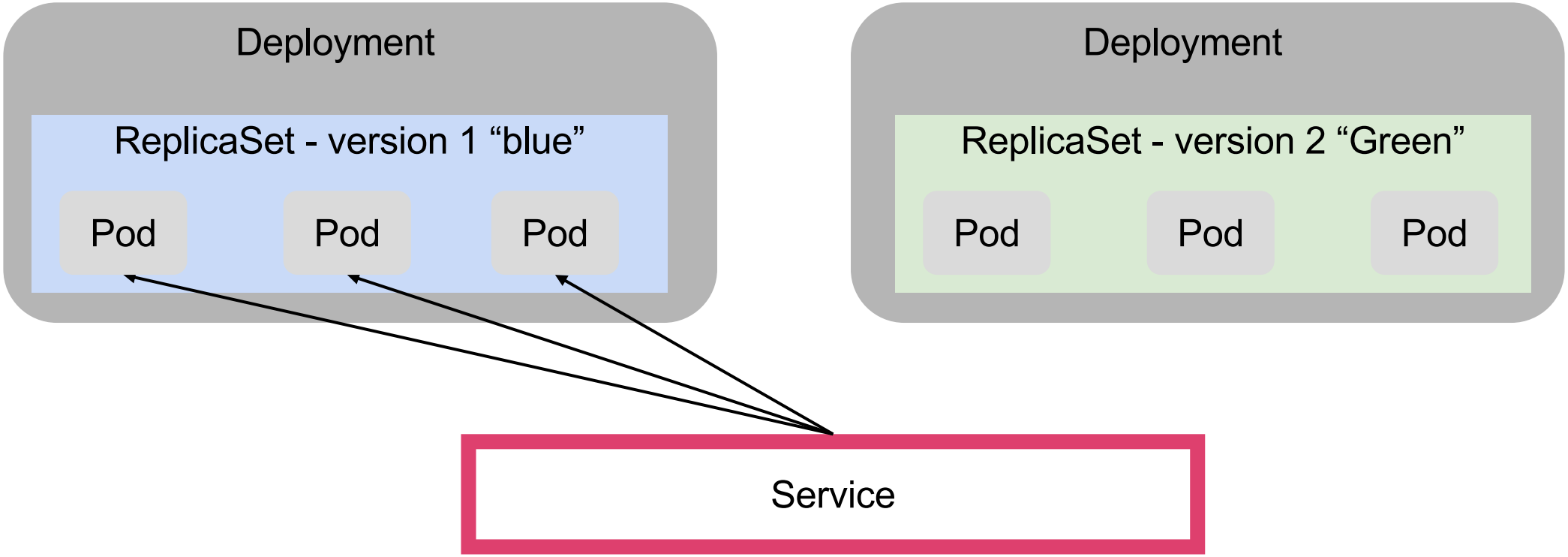


Blue/Green Deployment

- Allows new revisions to be deployed with less risk and no downtime
- There are two copies of the production environment
  - Blue environment is taking requests
  - Green environment is idle
- When deploying a new version:
  - Update new version to green environment leaving the blue environment in place
  - Test the green environment
  - When testing is complete, move the workload to the green environment
  - Green is now blue; can turn the old environment off
- Blue/green deployments also make rolling back to old versions easy

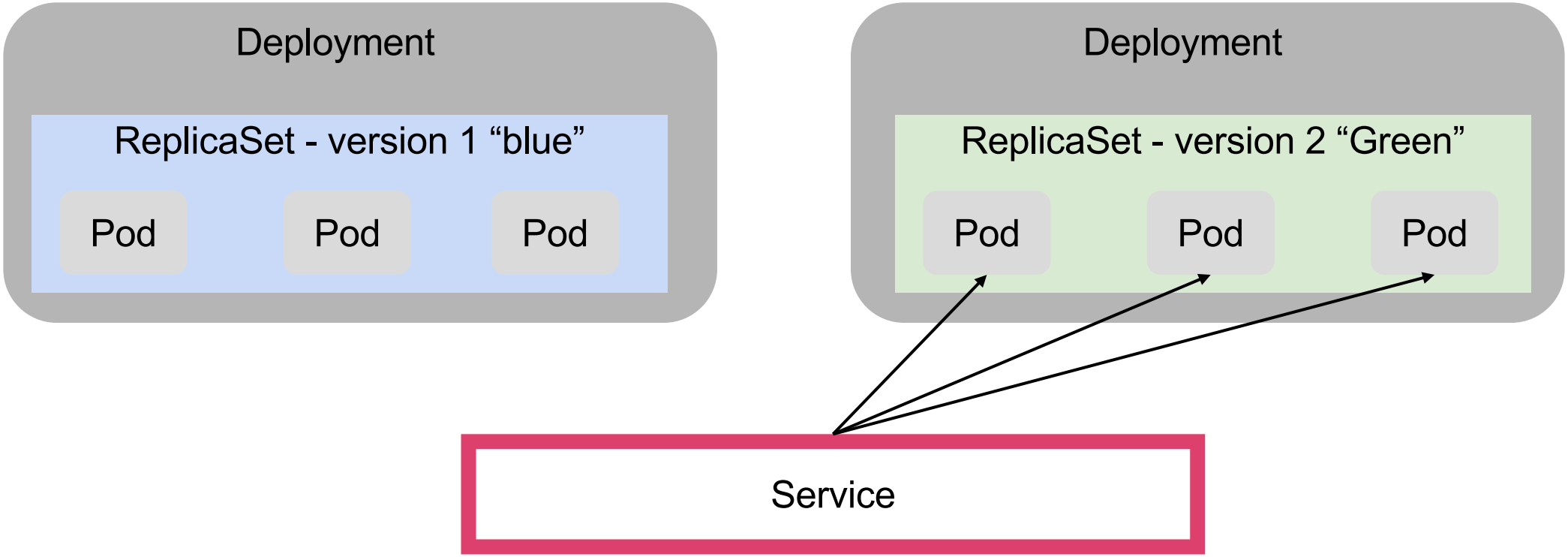
Blue/Green Deployments in Kubernetes

- Service routes all traffic to one version



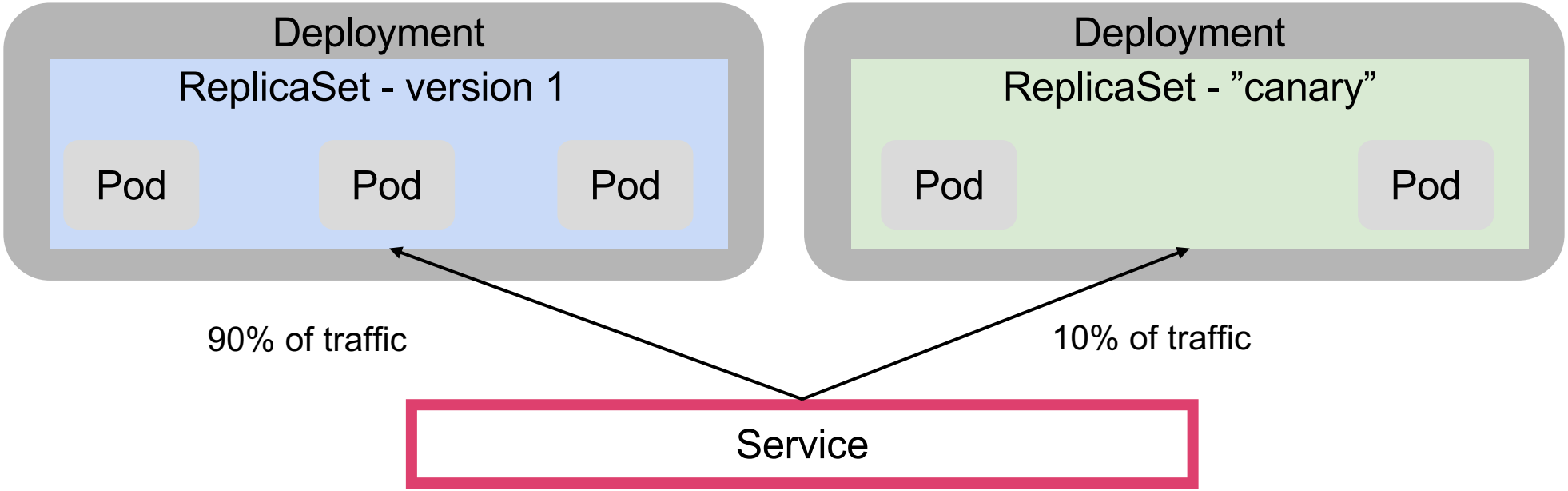
Blue/Green Deployments in Kubernetes (continued)

- Can switch the version quickly



Canary Releases

- A new version of a service is put into production alongside old versions
  - A small subset of select traffic is routed to the canary release
- Canary releases help developers know how a new version will perform
- Canary releases are easy to pull back if they fail their testing



Canary Release in Kubernetes

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

1. Services use selectors to determine what pods to route traffic to.
2. Create a new deployment with a new container, but use the same labels as the current deployment.
3. If the current deployment has 3 replicas and the new deployment has 1, then the new deployment gets about 25% of the traffic.

More advanced control possible with an Ingress controller or a service mesh (Istio)

- Outside the scope of this discussion

Kubectl apply

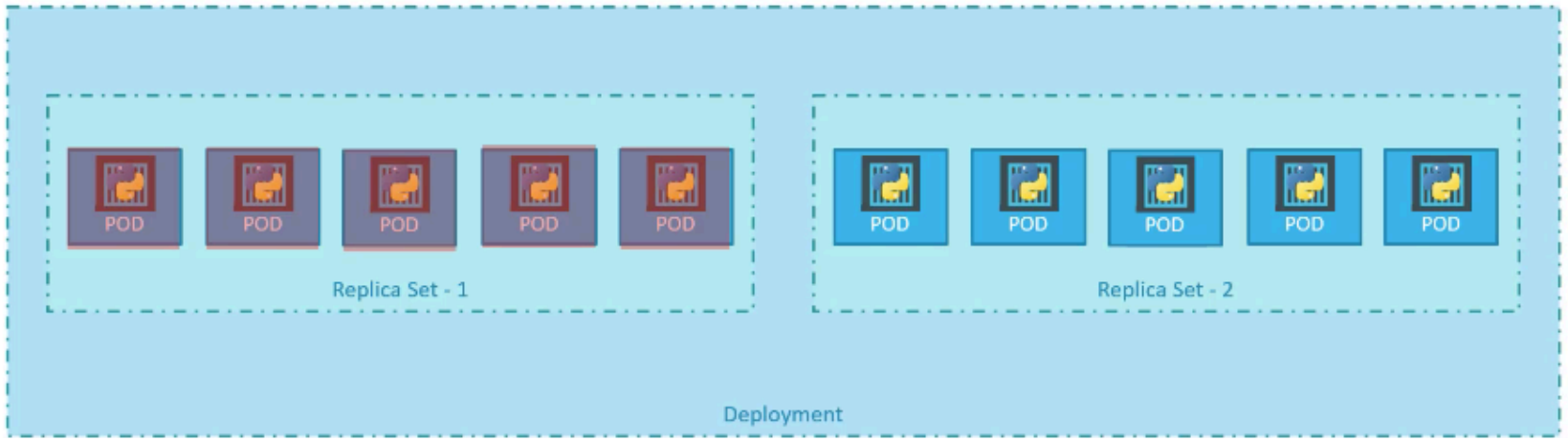
```
> kubectl apply -f deployment-definition.yml
deployment "myapp-deployment" configured
```

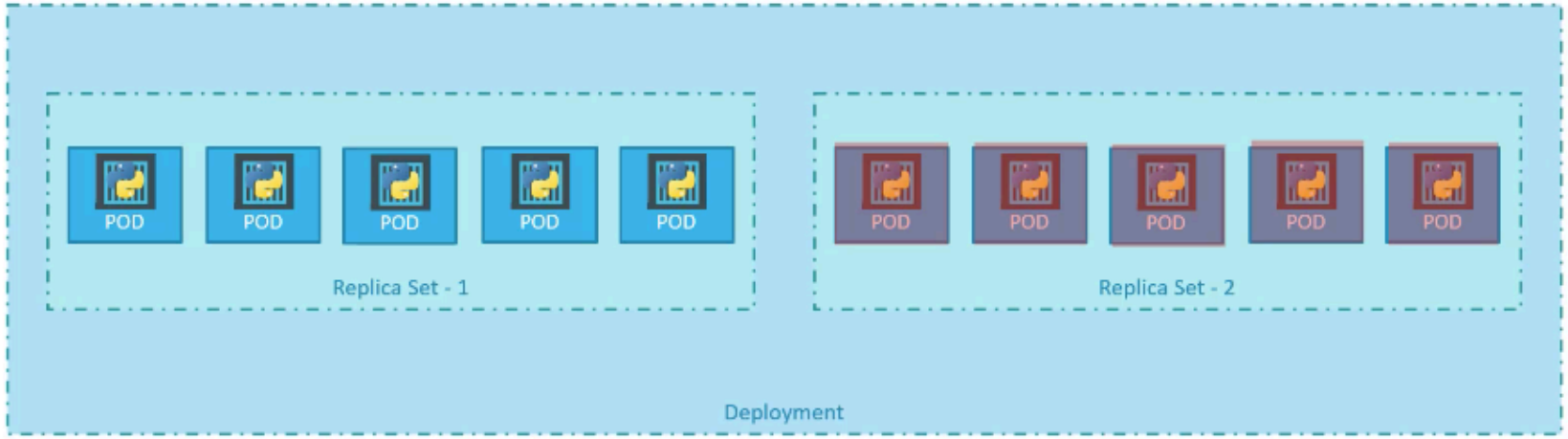
```
> kubectl set image deployment/myapp-deployment \
                  nginx=nginx:1.9.1
deployment "myapp-deployment" image is updated
```

deployment-definition.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
      app: myapp
      type: front-end
spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.7.1
  replicas: 3
  selector:
    matchLabels:
      type: front-end
```

Rollback






```
> kubectl rollout undo deployment/myapp-deployment
deployment "myapp-deployment" rolled back
```

Rollout Command

```
> kubectl rollout status deployment/myapp-deployment
Waiting for rollout to finish: 0 of 10 updated replicas are available...
Waiting for rollout to finish: 1 of 10 updated replicas are available...
Waiting for rollout to finish: 2 of 10 updated replicas are available...
Waiting for rollout to finish: 3 of 10 updated replicas are available...
Waiting for rollout to finish: 4 of 10 updated replicas are available...
Waiting for rollout to finish: 5 of 10 updated replicas are available...
Waiting for rollout to finish: 6 of 10 updated replicas are available...
Waiting for rollout to finish: 7 of 10 updated replicas are available...
Waiting for rollout to finish: 8 of 10 updated replicas are available...
Waiting for rollout to finish: 9 of 10 updated replicas are available...
deployment "myapp-deployment" successfully rolled out
```

```
> kubectl rollout history deployment/myapp-deployment
deployments "myapp-deployment"
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl apply --filename=deployment-definition.yml --record=true
```

Commands

Create

```
> kubectl create -f deployment-definition.yml
```

Get

```
> kubectl get deployments
```

Update

```
> kubectl apply -f deployment-definition.yml
```

```
> kubectl set image deployment/myapp-deployment nginx=nginx:1.9.1
```

Status

```
> kubectl rollout status deployment/myapp-deployment
```

```
> kubectl rollout history deployment/myapp-deployment
```

Rollback

```
> kubectl rollout undo deployment/myapp-deploymen
```

kubernetes

Services

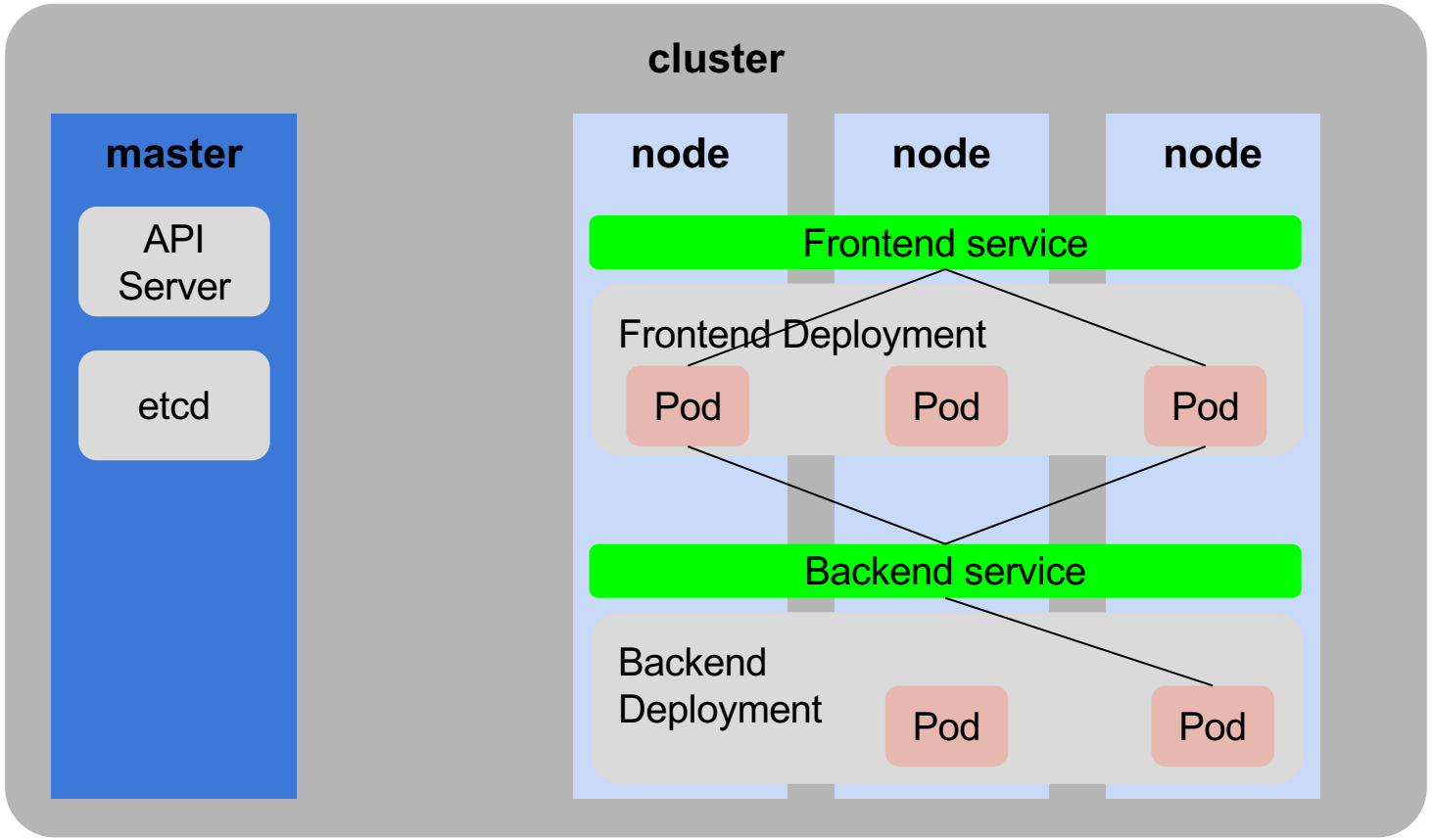
Running Applications

Types of Services

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

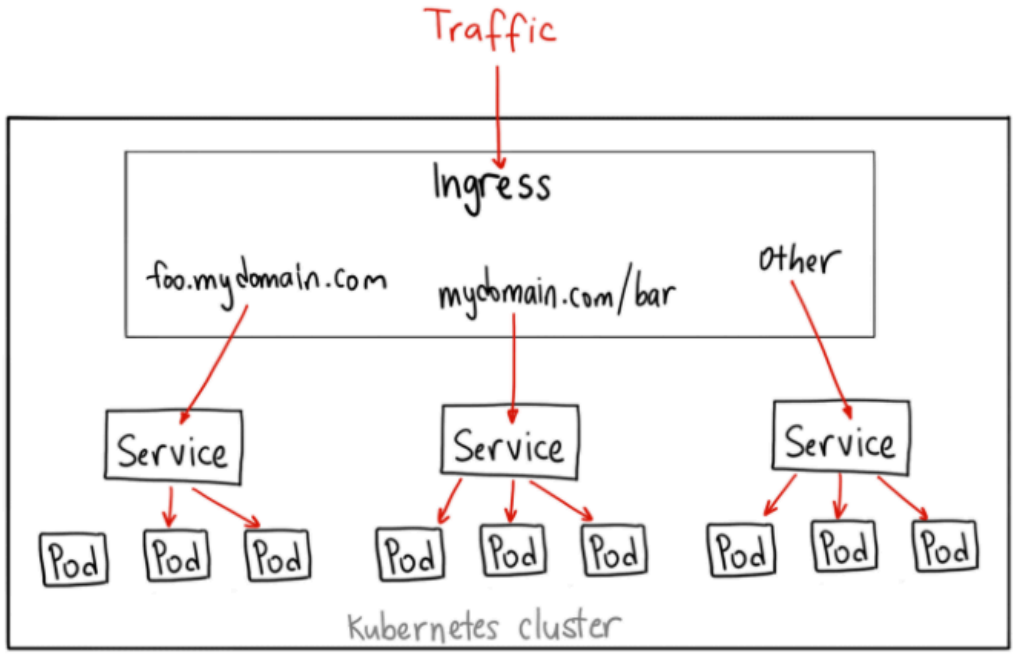
Types of Services (continued)

- A single application can have multiple services



Ingress

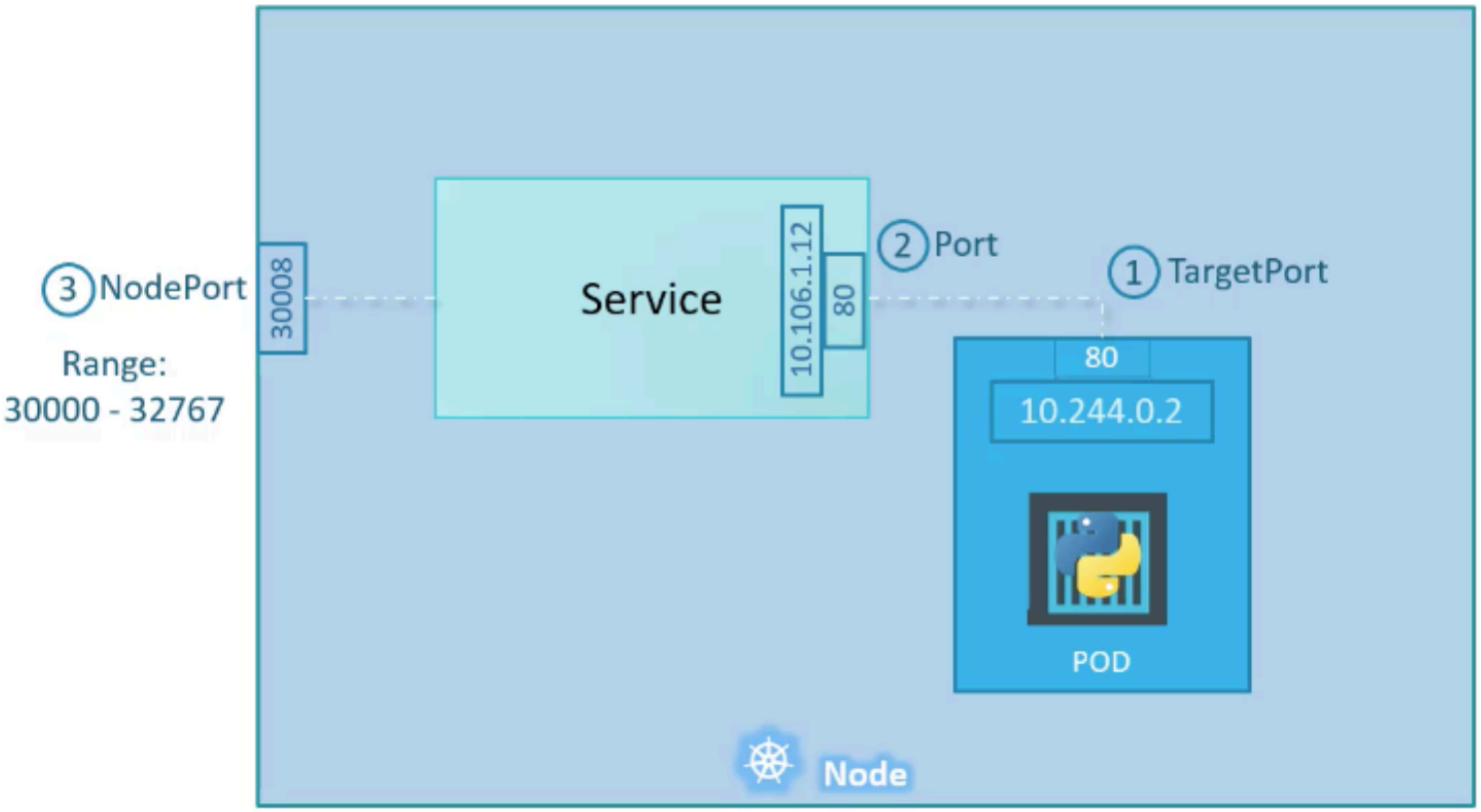
- Ingress is NOT a type of service.
  - Instead, it sits in front of multiple services and act as a "smart router" or entrypoint into your cluster.
- The default GKE ingress controller will spin up a HTTP(S) Load Balancer for you.
- This will let you do both path based and subdomain based routing to backend services.
- For example, you can send everything on `foo.yourdomain.com` to the foo service, and everything under the `yourdomain.com/bar/` path to the bar service.



Ingress

- When would you use this?
- Ingress is the most useful if you want to expose multiple services under the same IP address, and these services all use the same L7 protocol (typically HTTP).
- You only pay for one load balancer if you are using the native GCP integration, and because Ingress is “smart” you can get a lot of features out of the box (like SSL, Auth, Routing, etc)

Service - NodePort



service-definition.yml

```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
      *port: 80
       nodePort: 30008
```

Service - NodePort

service-definition.yml

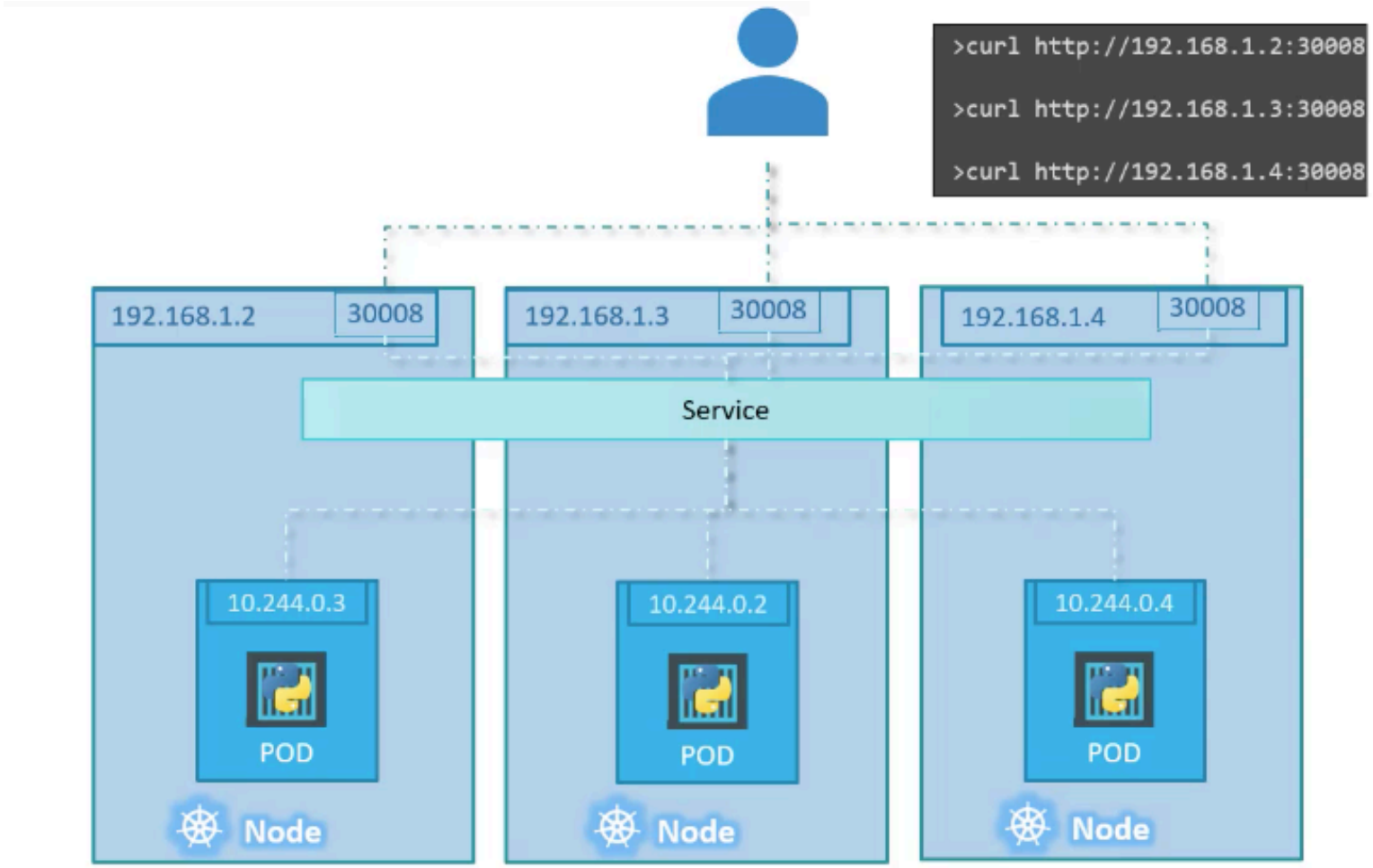
```yaml
apiVersion: v1
kind: Service
metadata:
    name: myapp-service

spec:
    type: NodePort
    ports:
     - targetPort: 80
       port: 80
       nodePort: 30008
    selector:
       app: myapp
       type: front-end
```

```
> kubectl create -f service-definition.yml
service "myapp-service" created
```

```
> kubectl get services
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1       <none>        443/TCP        16d
myapp-service   NodePort    10.106.127.123  <none>        80:30008/TCP   5m
```

```
curl http://192.168.1.2:30008
tml>
ead>
itle>Welcome to nginx!</title>
tyle>
  body {
      width: 35em;
      margin: 0 auto;
      font-family: Tahoma, Verdana, Arial, sans-serif;
  }
style>
head>
ody>
```

Service - NodePort



Adding the Load Balancer to Configuration

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

Accessing a Deployment with a Load Balancer

- Need a load balancer to route requests to the pods

```
kubectl expose deployment devops-deployment --port=80 --
target-port=8080 --type=LoadBalancer
```

- To get the load balancers public IP address, use the following command:

```
kubectl get services
```

Running a Load Balancer on Minikube—I

- To obtain an external IP address, the load balancer must be running on a platform that knows how to generate one, such as GCP and AWS
- Minikube does not know how generate an IP address
- Open a new terminal and execute the following command to create a route from the host to the deployment:

```
$ minikube tunnel
Status:
        machine: minikube
        pid: 37580
        route: 10.96.0.0/12 -> 192.168.49.2
        minikube: Running
        services: [space-lb]
    errors:
                minikube: no errors
                router: no errors
                loadbalancer emulator: no errors
```

Running a Load Balancer on Minikube—II

- Once the Minikube tunnel is running, go back to the original termial and execute the following command
  - The load balancer should now have an external IP address

```
$ kubectl get services
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.96.0.1        <none>           443/TCP        119m
space-lb     LoadBalancer   10.108.199.220   10.108.199.220   80:32365/TCP   46m
$
```

- Open a browser and point it to the external IP address of the load balancer service
  - What do you see?