

A ReplicaSet is a declarative way to manage Pods



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



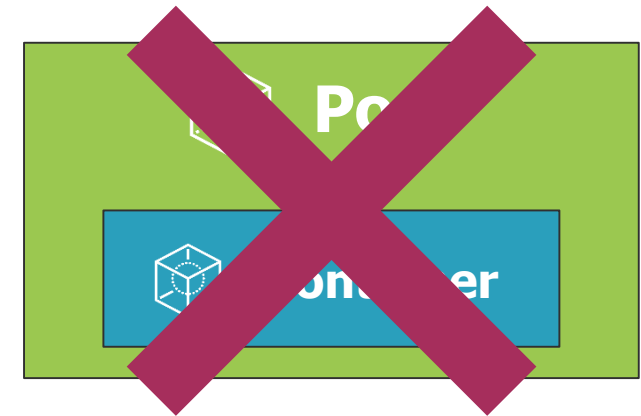
Pods represent the most basic resource in Kubernetes

Can be created and destroyed but are never re-created

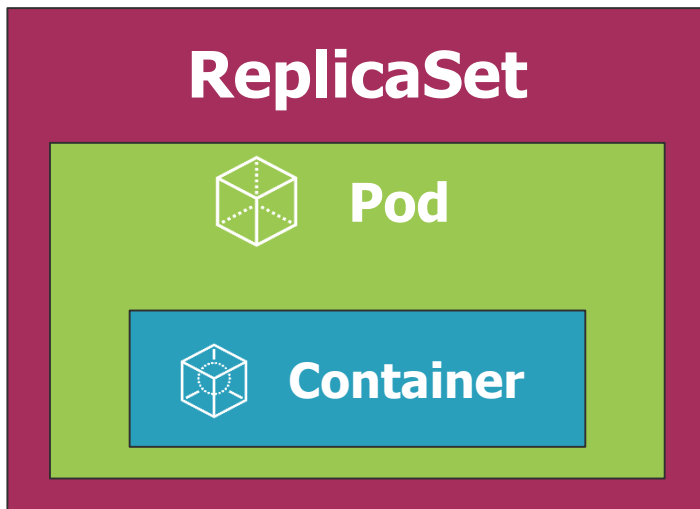
What happens if a Pod is destroyed?

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

Pods,
Deployments,
and ReplicaSets



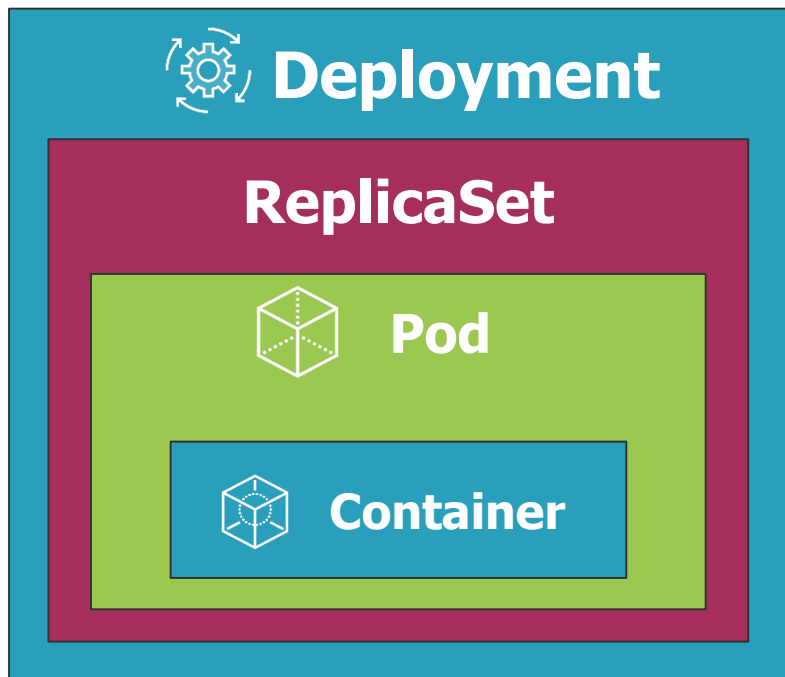
The Role of ReplicaSets



ReplicaSets act as a Pod controller:

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

The Role of Deployments



A Deployment manages Pods:

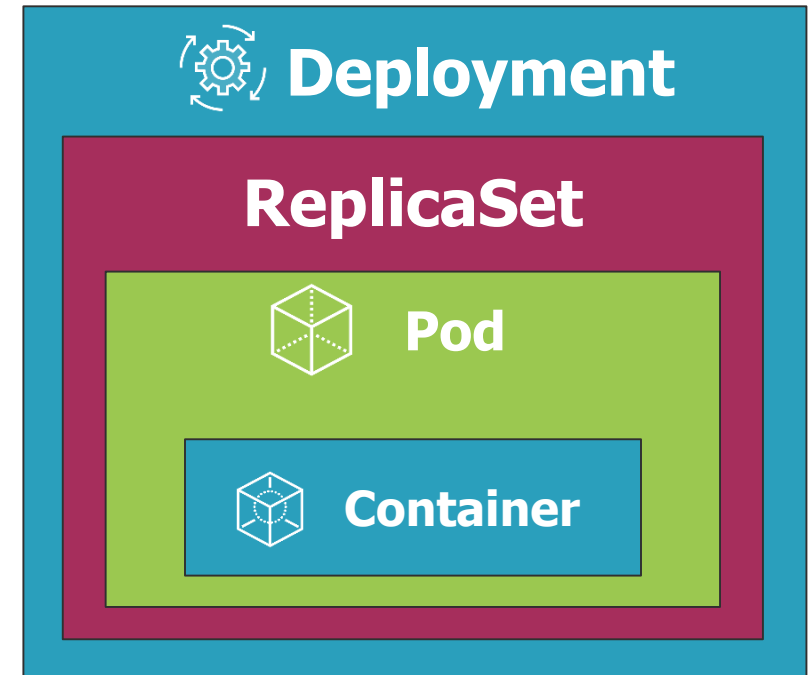
- Pods are managed using ReplicaSets
- Scales ReplicaSets, which scale Pods
- Supports zero-downtime updates by creating and destroying ReplicaSets
- Provides rollback functionality
- Creates a unique label that is assigned to the ReplicaSet and generated Pods
- YAML is very similar to a ReplicaSet

Defining a Deployment with YAML



Deployment

+ kubectl =



Defining a Deployment (From a High-Level)

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

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



Defining a Deployment

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

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




```
apiVersion: apps/v1

kind: Deployment

...

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

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

Create a Deployment

```
kubectl create -f file.deployment.yml
```

Creating a Deployment

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



Creating or Applying Changes

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

```
# Alternate way to create or apply changes to a  
# Deployment from YAML
```

```
kubectl apply -f file.deployment.yml
```

```
# Use --save-config when you want to use  
# kubectl apply in the future
```

```
kubectl create -f file.deployment.yml --save-config
```

Store current
properties in
resource's annotations



```
kubectl get deployments
```

Getting Deployments

List all Deployments



List all Deployments and their labels

```
kubectl get deployment --show-labels
```

Get all Deployments with a specific label

```
kubectl get deployment -l app=nginx
```

Deployments and Labels

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

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



Deleting a Deployment

To delete a Deployment use **kubectl delete**

Will delete the Deployment and all associated Pods/Containers

```
# Delete Deployment
```

```
kubectl delete deployment [deployment-name]
```

Scale the Deployment Pods to 5

```
kubectl scale deployment [deployment-name] --replicas=5
```

Scale by referencing the YAML file

```
kubectl scale -f file.deployment.yml --replicas=5
```

Scaling Pods Horizontally

Update the YAML file or use the **kubectl scale** command

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



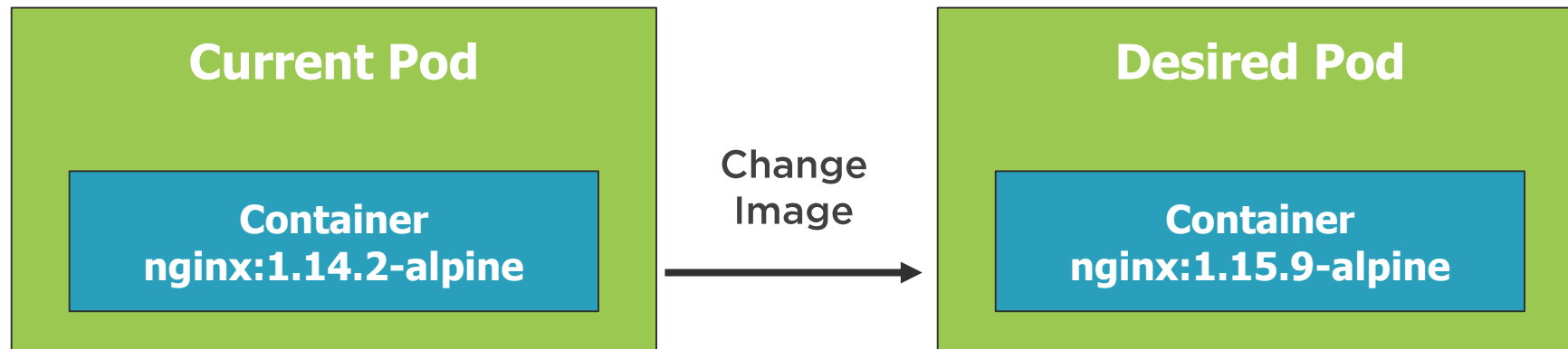
```
kubectl create -f nginx.deployment.yml --save-config  
kubectl describe [pod | deployment] [pod-name | deployment-name]  
kubectl apply -f nginx.pod.yml  
kubectl get deployments --show-labels  
kubectl get deployments -l app=my-nginx  
kubectl scale -f nginx.deployment.yml --replicas=4
```

kubectl Deployments Commands

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



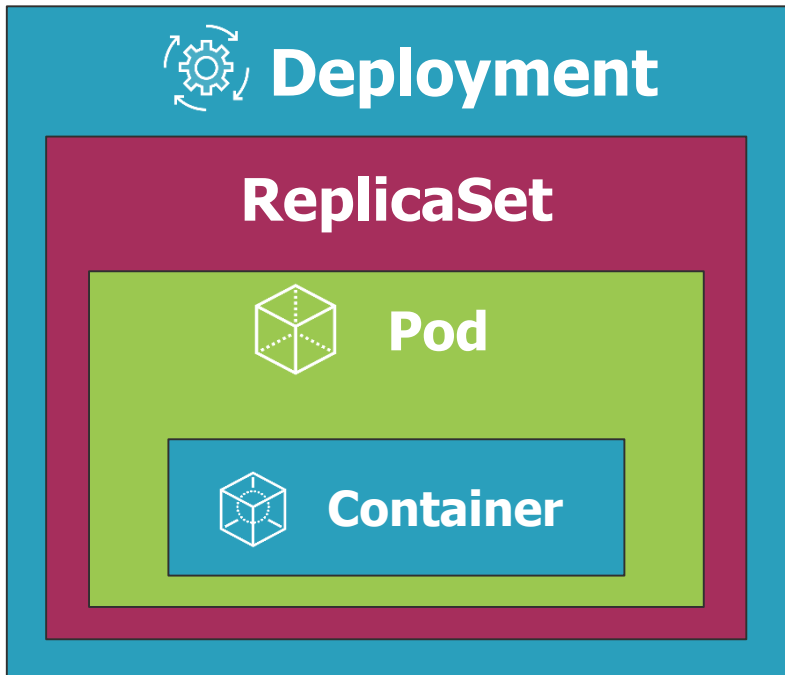
How Do You Update Existing Pods?



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



Deployment Options



One of the strengths of Kubernetes is zero downtime deployments

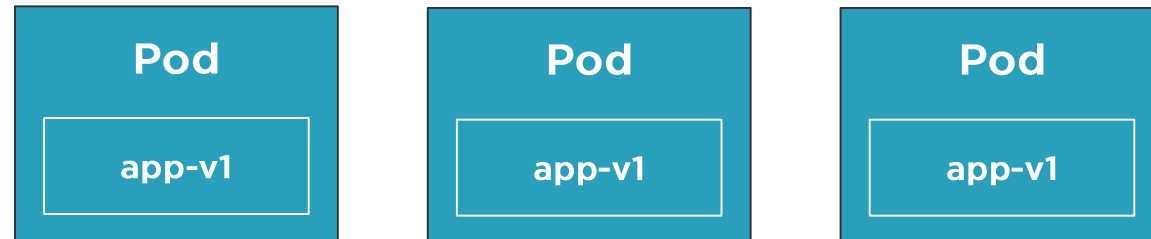
Update an application's Pods without impacting end users

Several options are available:

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

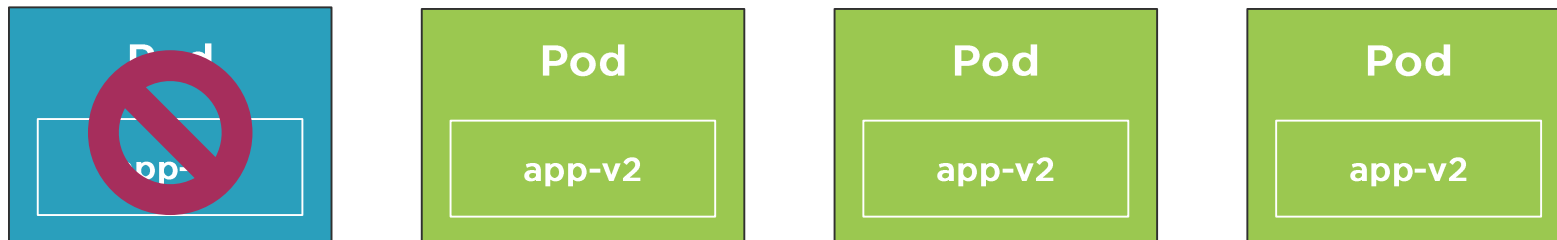
Rolling Deployments

Initial Pod State



Rolling Deployments

Delete Pod



Updating a Deployment

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

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