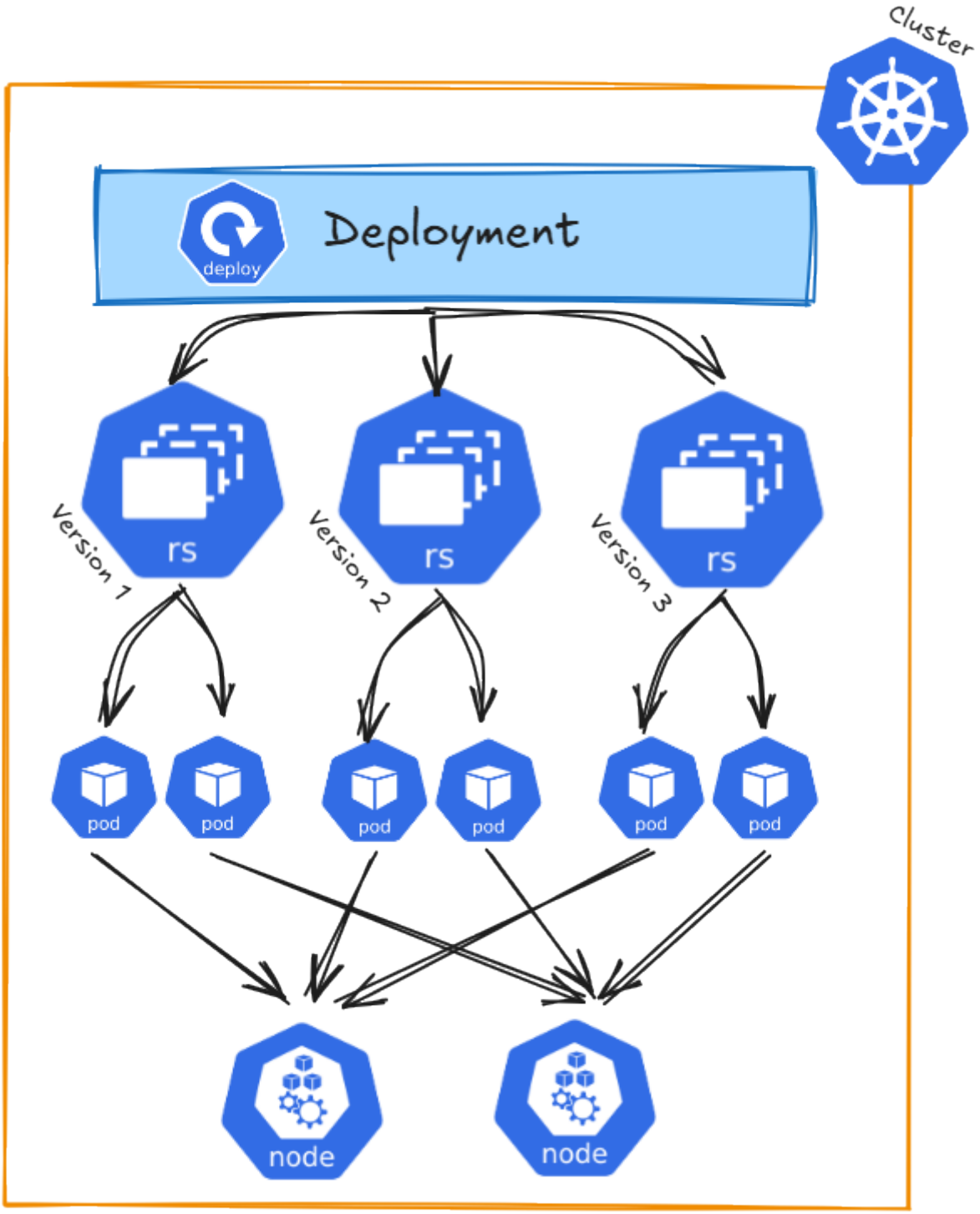
Deployment / ReplicaSet

Useful Links

- Kubernetes Official Documentation
- Using kubectl to Create a Deployment

Architecture



Detailed Description

Kubernetes `Deployments` serve as a blueprint for running your application in a cluster. Building on `ReplicaSets`, they ensure your application remains in the desired state by maintaining the defined number of instances.

ReplicaSets take care of the following:

- Ensuring the desired number of Pods are always running
- Replacing failed Pods automatically to maintain the specified replicas

On top of that, a Deployment adds features, such as:

- Automatically rolling out new versions of your application
- Rolling back to a previous version if something goes wrong
- Managing updates with strategies like rolling updates or recreating Pods

When a deployment in Kubernetes performs an upgrade (for example, if you change the image or other pod specifications), a rolling upgrade strategy is created by default. This manages the ReplicaSets to ensure minimal downtime and meet the constraints set by maxUnavailable and maxSurge. The Deployment maintains a current ReplicaSet (for the existing pods) and creates a new ReplicaSet for the updated pods. During the update, pods are gradually scaled down in the old ReplicaSet and scaled up in the new ReplicaSet.

`revisionHistoryLimit` The Number of old replicas to retain for rollback.

`strategy` Defines the deployment strategy, Default `RollingUpdate` or `Recreate`.

- Recreate: All existing Pods are destryoed before new ones are created
- RollingUpdate: Updates Pods in a rolling update fashion (maxUnavailable; maxUnavailable)

`maxUnavailable` The maximum number of pods that can be unavailable (not running or ready) during the update. Default: 25% Ensures a certain number of old pods remain running during the update to handle requests.

Example: If there are 4 replicas in a Deployment and maxUnavailable is set to 1:

- At most, 1 pod can be unavailable during the update.
- Kubernetes will ensure at least 3 pods (old or new) are running at any given time.

`maxSurge` The maximum number of extra pods that can be created beyond the desired replicas during the update. Default: 25% Determines how many new pods can be created during the update to replace old pods.

Example: If there are 4 replicas in a Deployment and maxSurge=1:

- Up to 5 pods (4 original + 1 new) can be running at once during the update.

Command Reference Guide

> Remeber to use dry-run and tee to check the configuration of each command first.
> `--dry-run=client -o yaml | tee nginx-deployment.yaml`

Create a ReplicaSet using a YAML file (declarative method)

nginx-replicaset.yaml

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginxdemos/hello
        ports:
        - containerPort: 80
```

```
# Apply replicaset
kubectl apply -f nginx-replicaset.yaml

# Get ReplicaSet informatoin
kubectl get replicaset nginx-replicaset -o wide

# Get detailed ReplicaSet information
kubectl describe replicaset/nginx-replicaset
```

```
# Check the current Pods running
kubectl get pods

# Delete a Pod of the ReplicaSet
FIRST_POD=$(kubectl get pods -l app=nginx -o jsonpath='{.items[0].metadata.name}')
kubectl delete pod $FIRST_POD

# Recheck running Pods
kubectl get pods

# Change image in yaml to nginxdemos/hello:v0.2 and apply replicaset again
kubectl apply -f nginx-replicaset.yaml



# You will encounter that the replicaset was updated - but the pods are still using the old image
kubectl describe replicaset/nginx-replicaset
kubectl describe pod/<podname>

# You have to kill and recreate the pods, so the new ones will be created with the new image. (see Hint section)
FIRST_POD=$(kubectl get pods -l app=nginx -o jsonpath='{.items[0].metadata.name}')
kubectl scale rs nginx-replicaset --replicas=0
kubectl scale rs nginx-replicaset --replicas=3
kubectl describe pod/$FIRST_POD
```

Create a Deplyoment (imperative method)

nginx-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx-deployment
  name: nginx-deployment
spec:
  replicas: 25
```

```
    selector:
      matchLabels:
        app: nginx-deployment
    strategy:
      rollingUpdate:
        maxSurge: 25%
        maxUnavailable: 25%
      type: RollingUpdate
    template:
      metadata:
        creationTimestamp: null
        labels:
          app: nginx-deployment
      spec:
        containers:
        - image: nginxdemos/hello:0.4
          imagePullPolicy: Always
          name: hello
          ports:
          - containerPort: 80
            protocol: TCP
          resources: {}
  status: {}
```

```
# Create nginx deployment with the default of one replica
kubectl create deployment nginx-deployment --image=nginxdemos/hello --port=80

# Create nginx deployment with three replicas
kubectl create deployment nginx-deployment --image=nginxdemos/hello --port=80  --replicas=3

# Check deployment
kubectl get deployment -o wide

# Get detailed deployment information
kubectl describe deployment

# Get ReplicaSet information created by deployment
kubectl get replicasets -o wide
```

```
# Get History for deployment
kubectl rollout history deployment/nginx-deployment

# Annotate inital history entry
kubectl annotate deployment/nginx-deployment kubernetes.io/change-cause="init nginx deployment"
kubectl rollout history deployment/nginx-deployment

# Scale up/down deployment (scale is not changing history)
kubectl scale deployment/nginx-deployment --replicas=2; watch kubectl get pods -o wide
kubectl scale deployment/nginx-deployment --replicas=20; watch kubectl get pods -o wide



# Run update and rollback
FIRST_POD=$(kubectl get pods -l app=nginx-deployment -o jsonpath='{.items[0].metadata.name}')
# Check image name
kubectl get pod $FIRST_POD -o jsonpath='{.spec.containers[0]}'
kubectl get deployment/nginx-deployment -o yaml > nginx-deployment.yaml
# check Update yaml modifications under code section
kubectl apply -f nginx-deployment.yaml && kubectl rollout status deployment/nginx-deployment
kubectl annotate deployment/nginx-deployment kubernetes.io/change-cause="update new version"
kubectl rollout history deployment/nginx-deployment
# Recheck image name after update
kubectl get pod $FIRST_POD -o jsonpath='{.spec.containers[0]}'
# Check replicaset
kubectl get replicaset

# ROllback to previouse revision
kubectl rollout undo deployment/nginx-deployment --to-revision=1 && kubectl rollout status deployment/nginx-deployment
kubectl rollout history deployment/nginx-deployment
# check pod image, as it was reverted to old revision
FIRST_POD=$(kubectl get pods -l app=nginx-deployment -o jsonpath='{.items[0].metadata.name}')
kubectl get pod $FIRST_POD -o jsonpath='{.spec.containers[0]}'

# Run into failed state (change image in yaml to  nginxdemos/hello:5.1)
kubectl apply -f nginx-deployment.yaml && kubectl rollout status deployment/nginx-deployment
kubectl annotate deployment/nginx-deployment kubernetes.io/change-cause="failed version"
kubectl rollout history deployment/nginx-deployment
kubectl rollout undo deployment/nginx-deployment --to-revision=3 && kubectl rollout status deployment/nginx-deployment
```

```
# check pod image, as it was reverted to healthy revision
FIRST_POD=$(kubectl get pods -l app=nginx-deployment -o jsonpath='{.items[0].metadata.name}')
kubectl get pod $FIRST_POD -o jsonpath='{.spec.containers[0]}'
kubectl rollout history deployment/nginx-deployment

# Pause from deplyoment
kubectl rollout pause deployment nginx-deployment
kubectl set image deployment/nginx-deployment nginx=nginx:1.22
kubectl get deployment nginx-deployment -o yaml | grep paused
kubectl rollout resume deployment nginx-deployment
kubectl rollout status deployment nginx-deployment



# Delete deployment
kubectl delete deployment/nginx-deployment
```

Hints

Deployments manage **ReplicaSets**, primarily due to historical reasons. There is no practical need to manually create ReplicaSets (or previously, ReplicationControllers), as Deployments, built on top of ReplicaSets, offer a more user-friendly and feature-rich abstraction for managing the application lifecycle, including replication, updates, and rollbacks.

ReplicaSets do not support auto updates. As long as required number of pods exist matching the selector labels, replicaset's jobs is done.

When a **rollback** is applied to a Deployment, Kubernetes creates a new history revision for the rollback. It doesn’t simply go back to an old revision but treats the rollback as a new change. This means the rollback gets its own revision number, while the previous revisions remain saved. This helps you keep track of all changes, including rollbacks.

Revision #28
Created 23 November 2024 09:26:35 by Admin
Updated 26 November 2024 20:50:35 by Admin