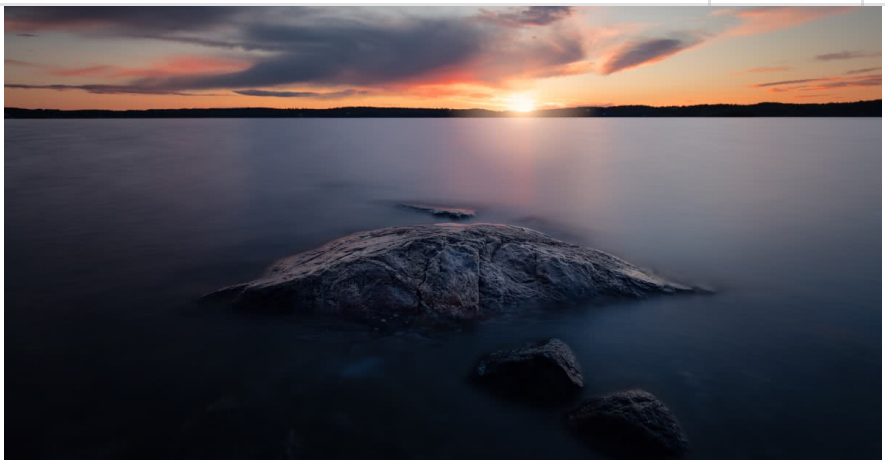
# Rolling Updates with Kubernetes Deployments

Learn how to use Kubernetes Deployment to perfom rolling update

NOVEMBER 14, 2016

TA-CHING CHEN

6 minute read



## Table of Contents

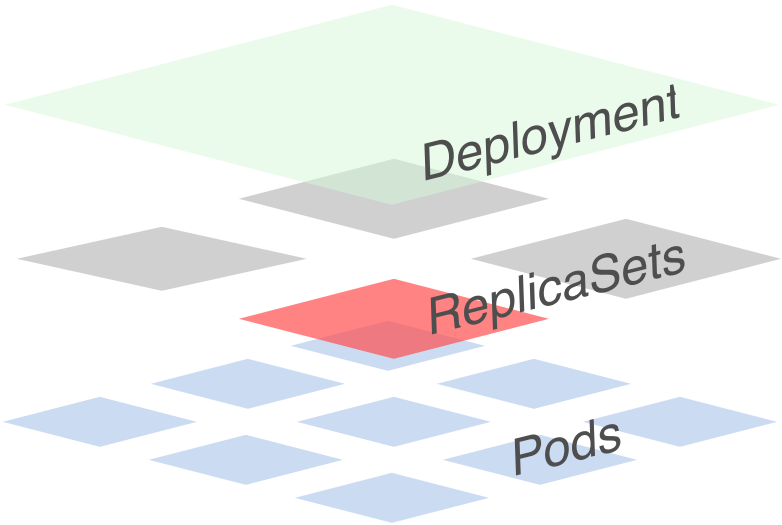


---

## Deployment

The newer version of Kubernetes, official suggests using `Deployment` instead of `Replication Controller(rc)` to perform a rolling update. Though, they are same in many ways, such as ensuring the homogeneous set of pods are always up/available and also they provide the ability to help the user to roll out

## Relationship among Pods, ReplicaSet and Deployment




A `Deployment` owns and manages one or more `ReplicaSets` . And `Replica Set` manages the basic units in Kubernetes - `Pods` .

Why Deployment manages multiple ReplicaSets? The answer is Kubernetes wants to support `rollback` mechanism. Kubernetes creates a new ReplicaSet each time after the new Deployment config is deployed and also keeps the old ReplicaSet. So that we can rollback to the previous state with old ReplicaSet. And there is only one ReplicaSet is in active state, which means DESIRED > 0.

```
NAME                          DESIRED   CURRENT   READY     AGE
websdk-backend-go-1144733912  0         0         0         42d
websdk-backend-go-1791900003  0         0         0         43d
websdk-backend-go-1986738532  0         0         0         43d
websdk-backend-go-3350542699  0         0         0         43d
websdk-backend-go-3961390587  0         0         0         41d
websdk-backend-go-4146660860  4         4         4         29d

$ kubectl get pod -l service=websdk-backend-go
NAME                                READY     STATUS    RESTARTS   AGE
websdk-backend-go-4146660860-7rj8h  2/2       Running   0          29d
websdk-backend-go-4146660860-9gq81  2/2       Running   0          14d
websdk-backend-go-4146660860-r7dqk  2/2       Running   0          14d
websdk-backend-go-4146660860-sb9lc  2/2       Running   0          29d
```

## Hands-On

Let's create a Deployment with the following deployment yaml file `nginx.yaml` .

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  template:
    metadata:
      labels:
        service: http-server
    spec:
      containers:
      - name: nginx
        image: nginx:1.10.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
```

You can use `kubectl create` or `kubectl apply` to create nginx deployment.

```
$ kubectl create -f nginx.yaml
deployment "nginx" created
```

```
$ kubectl get deployment
NAME               DESIRED   CURRENT   UP-T
nginx              3         3         3
```

As I mentioned before, Deployment manages ReplicaSets and ReplicaSet manages Pods.

```
└─ Deployment: <name>
   └─ ReplicaSet: <name>-<rs>
      └─ Pod: <name>-<rs>-<randomString>
```

Kubernetes will create the replicaset for us after the creation of deployment.

```
$ kubectl get rs
NAME                DESIRED   CURRENT   AGE
nginx-3322722759    3         3         8m
```

And the Replica Set will create pods after its been created.

```
$ kubectl get pod -l "service in (http-serve
NAME                     READY     STATUS
nginx-3322722759-7vp34   1/1       Running
nginx-3322722759-ip5w2   1/1       Running
nginx-3322722759-q97b7   1/1       Running
```

## Rolling Update

In order to support rolling update, we need to configure the update strategy first.

So we add following part into `spec`

```
  # indicate which strategy we want for roll
  type: RollingUpdate
  rollingUpdate:
    maxSurge: 1
    maxUnavailable: 1
```

- minReadySeconds:
  - the bootup time of your application, Kubernetes waits specific time til the next pod creation.
  - Kubernetes assume that your application is available once the pod created by default.
  - If you leave this field empty, the service may be unavailable after the update process cause all the application pods are not ready yet
- maxSurge:
  - amount of pods more than the desired number of Pods
  - this fields can be an absolute number or the percentage
  - ex. maxSurge: 1 means that there will be at most 4 pods during the update process if replicas is 3
- maxUnavailable:
  - amount of pods that can be unavailable during the update process
  - this fields can be a absolute number or the percentage
  - this fields cannot be 0 if `maxSurge` is set to 0
  - ex. maxUnavailable: 1 means that there will be at most 1 pod unavailable during the update

The final `nginx.yaml` would be like the following

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-test
spec:
  replicas: 10
  selector:
    matchLabels:
      service: http-server
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
  minReadySeconds: 5
  template:
    metadata:
      labels:
        service: http-server
    spec:
      containers:
      - name: nginx
        image: nginx:1.10.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 80
```

Lets apply the new nginx.yaml

```
$ kubectl apply -f nginx.yaml --record
```

Now, for example, if we want to update the docker image, we have three ways to perform the rolling update.

- `set image`

```
# example
$ kubectl set image deployment nginx nginx=r
```

- replace

Modify the container image version in nginx.yaml (1.10.2)

```
spec:
  containers:
  - name: nginx
    # newer image version
    image: nginx:1.11.5
    imagePullPolicy: IfNotPresent
    ports:
    - containerPort: 80
```

Using `replace` here instead of apply

```
# format
$ kubectl replace -f <yaml> --record
# example
$ kubectl replace -f new-nginx.yaml --recor
```

- edit

```
# format
$ kubectl edit deployment <deployment> --rec
# example
$ kubectl edit deployment nginx --record
```

This command opens the editor, and you just need to change the image version in it.

```
# Please edit the object below. Lines beginr
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    kubectl.kubernetes.io/last-applied-confi
...
    spec:
      containers:
      - image: nginx:1.10.2
        imagePullPolicy: IfNotPresent
        name: nginx
...
```

### Rollout Status

```
$ kubectl rollout status deployment nginx
```

### Pause Rolling Update

```
$ kubectl rollout pause deployment <deployme
```

### Resume Rolling Update

```
$ kubectl rollout resume deployment <deploym
```

### Rollback

After the image update, your colleague finds the service become unstable you may want to go back to the previous version. Unfortunately, he/she dunno how the previous config looks like. Well, you don't need the time machine, just let rollback to do its job.

typed, so that you can distinguish between the revisions.

```
$ kubectl apply -f nginx.yaml --record
deployment "nginx" configured

$ kubectl set image deployment nginx nginx=r
deployment "nginx" image updated

$ kubectl rollout history deployment ngin
deployments "nginx":
REVISION  CHANGE-CAUSE
1   kubectl apply -f nginx.yaml --record
2   kubectl set image deployment nginx ngin
```

Now, lets go back to revision 1

```
# to previous revision
$ kubectl rollout undo deployment <deploymer
# to specific revision
$ kubectl rollout undo deployment <deploymer
# exmaple
$ kubectl rollout undo deployment nginx --to
```

All revision history is stored in the ReplicaSets that deployment controls. If you want to keep more revision history, please set `.spec.revisionHistoryLimit` in yaml to specify the number of old ReplicaSets to retain to allow rollback. (set this field at the first time apply)

```
...
spec:
  replicas: 10
  selector:
    matchLabels:
      service: http-server
  strategy:
```

```
      maxUnavailable: 1
  minReadySeconds: 5
  revisionHistoryLimit: 10
...
```

```
$ kubectl rollout history deployment/nginx
deployments "nginx":
REVISION  CHANGE-CAUSE
2   kubectl set image deployment nginx nginx
3   kubectl set image deployment nginx nginx
4   kubectl set image deployment nginx nginx
5   kubectl set image deployment nginx nginx
```

## Troubleshooting

- Please add labels to the `spec.template.metadata.labels`

```
The Deployment "nginx" is invalid.

* spec.selector: Required value
* spec.template.metadata.labels: Invalid val
```

## Reference

- http://kubernetes.io/docs/user-guide/deployments/
- http://kubernetes.io/docs/user-guide/kubectl/kubectl_rollout_history/
- https://youtu.be/wVMXjDoeRS4?t=3381
- http://kubernetes.io/docs/user-guide/kubectl/kubectl_rolling-update/

## See Also

- Kubernetes - Two Steps Installation
- Kubernetes - Installation
- Kubernetes - Pod
- Kubernetes - High Availability
- Adopting Container and Kubernetes in Production



CATEGORY | KUBERNETES



**Idan Adar** • 4 years ago

Hello,

What is more recommended? using "kubectl replace" followed by "kubectl apply" or using "kubectl set image"?
This, assuming a Jenkins pipeline to update the image used in the pod with a newer image tag.

I am leaning towards "kubectl set image"

new pod was instantiated, it will replace an existing pod after 10 seconds?

58 · Reply · Share ›

**Ta Ching Chen** Mod → Idan Adar
• 4 years ago • edited

Hi Idan,

Thank you for your question

1. use `kubectl set image` or `kubectl replace`

It depends on your situation.

In most cases, use `kubectl set image` in the delivery pipeline is a better way to update images. It's more clear and helps others to understand what image was updated between two build jobs.

However, in my current company, we use Git to version the difference between two bots' auto-commits and a tool that merges config and template into K8S yaml. In this case, we use `kubectl replace` instead. Though it lowers the transparency in pipeline, but it makes sure that deployment's config is always up-to-date.

2. minReadySeconds

As long as the `maxUnavailable` is set to zero, no existing pod will be replaced until the new pod was instantiated after `minReadySeconds`

‹ Kubernetes - Assigning Pod to Nodes

Building Minimal Docker Image for Go Applications ›