`https://www.bmc.com/blogs/kubernetes-deployment/ ##### deployment full-blog

vi nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: sample-nginx-1

labels:

pot: sports

spec:

replicas: 4

selector:

matchLabels:

pot: sports

template:

metadata:

labels:

pot: sports

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

Resources:

Limits:

Cpu: 100m

Memory: 100Mi

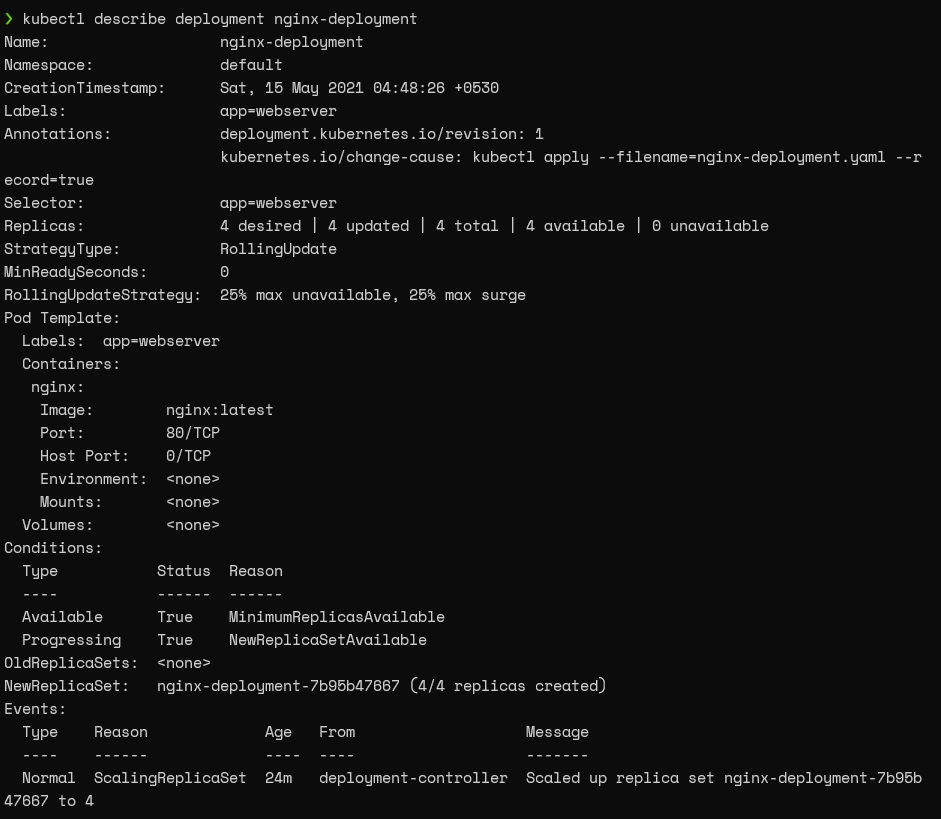
kubectl apply -f nginx-deployment.yaml



kubectl get all



kubectl describe deployment nginx-deployment

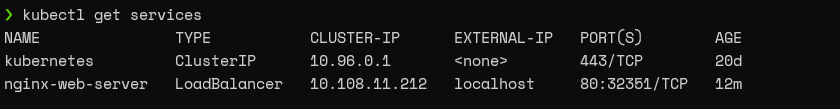


***Exposing the ReplicaSet***

kubectl expose deployment nginx-deployment --type=LoadBalancer --name=nginx-web-server

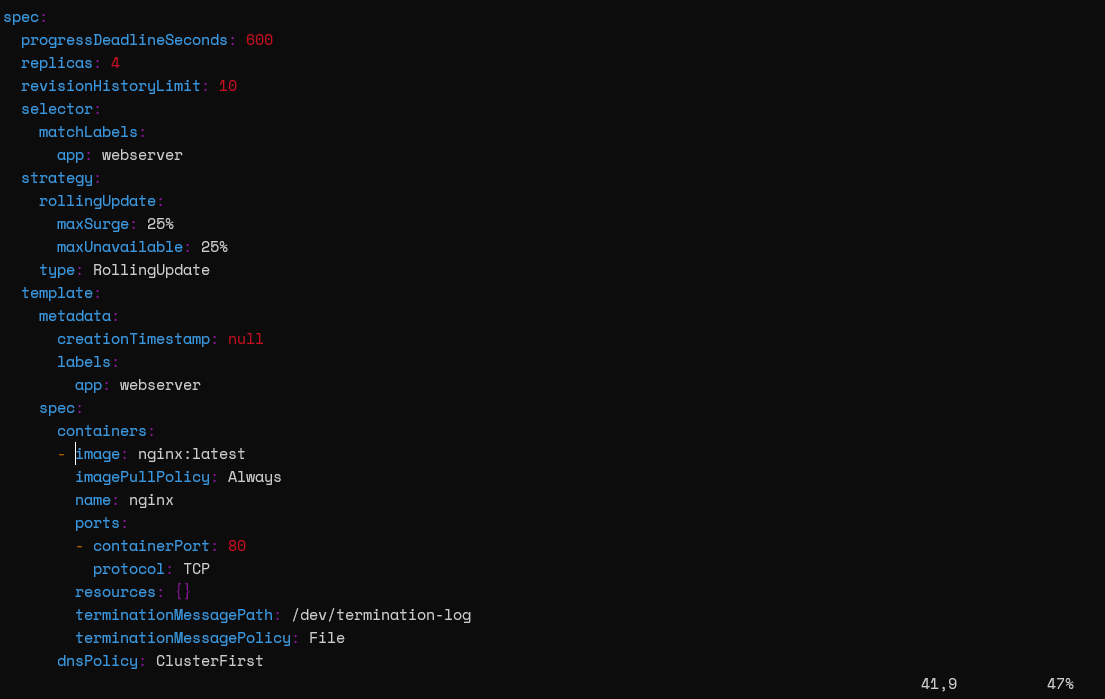
Load Balancer

kubectl get services



### Changing the deployment configuration file

kubectl edit deployment nginx-deployment

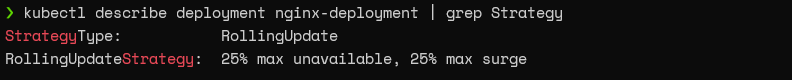


Deployment strategies

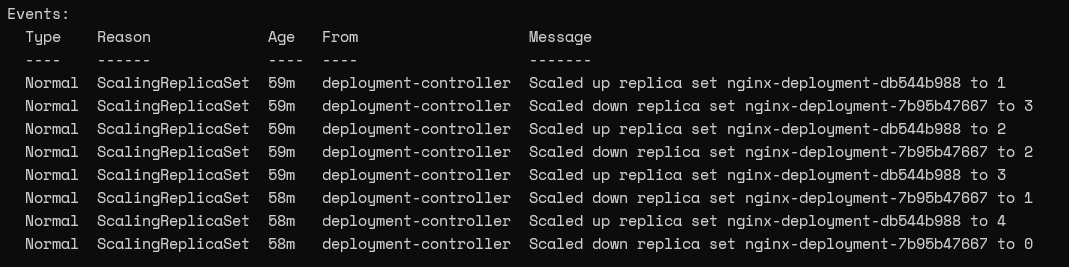
Kubernetes uses two deployment strategies called “Recreate” and “RollingUpdate” to recreate pods. We can define those strategies in .spec.strategy.type field. The RollingUpdate strategy is the default for deployments.

* **Recreate**will delete all the existing pods before creating the new pods.
* **RollingUpdate**will recreate the pods in a rolling update fashion. Moreover, it will delete and recreate pods gradually without interrupting the application availability. This strategy utilizes the maxUnavailable and maxSurge values to control the rolling update process..
  + **maxUnavailable**defines the maximum number of pods that can be unavailable in the update process.
  + **maxSurge**defines the maximum number of pods that can be created.

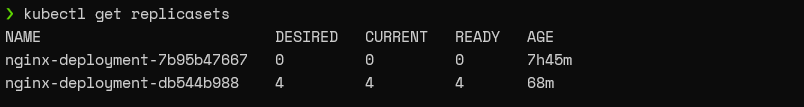
kubectl describe deployment sample-nginx-1| grep Strategy



kubectl describe deployment sample-nginx-1



kubectl get replicasets



Discovering the Kubernetes deployment details

When managing a Kubernetes cluster, the initial step would be to check for a successful deployment. For this purpose, we can use the kubectl rollout status and kubectl get deployment commands.

* **kubectl rollout status** informs the user if the deployment was successful.
* **kubectl get deployment** shows the desired and updated number of replicas, the number of replicas running, and their availability. As mentioned previously, we can use the kubectl describe command to a complete picture of the deployment.

kubectl rollout status deployment sample-nginx-1

kubectl get deployment sample-nginx-1



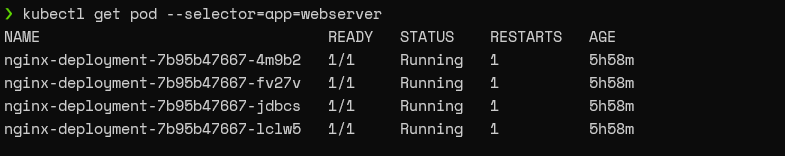
By default, Kubernetes will automatically append a pod-template-hash value to the ReplicaSet name. However, do not rename the ReplicaSet as it will break the deployment.

kubectl get replicaset



The kubectl get pod command can be used to get only the information about the pods related to the deployment while defining a selector. In this instance, we will be using the “app:webserver” label as the selector.

kubectl get pod --selector=pot=sports



Scaling deployments

As the Deployments rely on ReplicaSets to manage the pods, we can scale up or down the number of pods. This scaling can be done either:

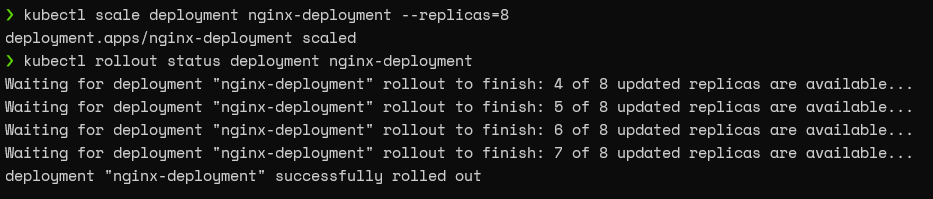
* Manually
* By configuring an auto-scaling rule

Manually

We can use the scale command with the replica parameter to scale the deployment to the desired number. For instance, we will use the following command to scale up our deployment from 4 pods to 8 pods.

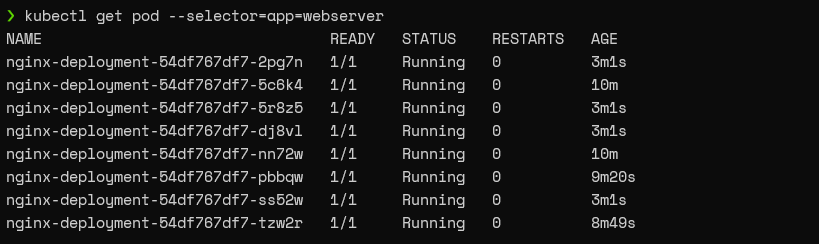
kubectl scale deployment sample-nginx-1 --replicas=5

kubectl rollout status deployment sample-nginx-1



If we look at the pods associated with this deployment, we can see that it has eight pods now.

kubectl get pods --selector=pot=sports



### Autoscaling

The best practice for scaling deployments would be to configure an auto-scaling rule so that the pods will scale according to predefined thresholds.

So, let’s go ahead and create an autoscaling rule for our deployment, which will scale according to the CPU load of the node.

kubectl autoscale deployment sample-nginx-1 --min=5 --max=10 --cpu-percent=70

Autoscale Deployment

According to the above configuration, if the CPU load is greater than 70%, the deployment will scale until the maximum number of pods is reached (maximum ten pods).

## Managing Kubernetes deployments

## Performing Rolling Update on a deployment

Let’s assume that we need to change the Nginx server version in our deployment to target our application to a specific server version. We can do this by either:

* Using the kubectl set image command
* Changing the deployment configuration file

Using the set image command

The set image command can be used with the container name of the template and the required image name to update the pods.

kubectl set image deployment nginx-deployment nginx=nginx:1.19.10

Image Deployment

**Pausing & resuming deployments**

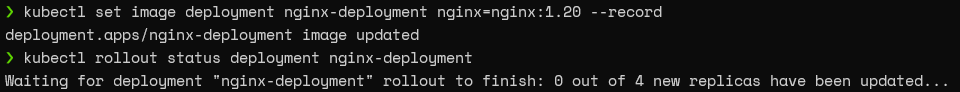
kubectl rollout pause deploy nginx-deployment

Pause Deployment

Now, if we update the Nginx image in the paused status, the controller will accept the change, yet it will not trigger the new ReplicaSet rollout. If we look at the rollout status, it will indicate a pending change.

kubectl set image deployment nginx-deployment nginx=nginx:1.20 --record

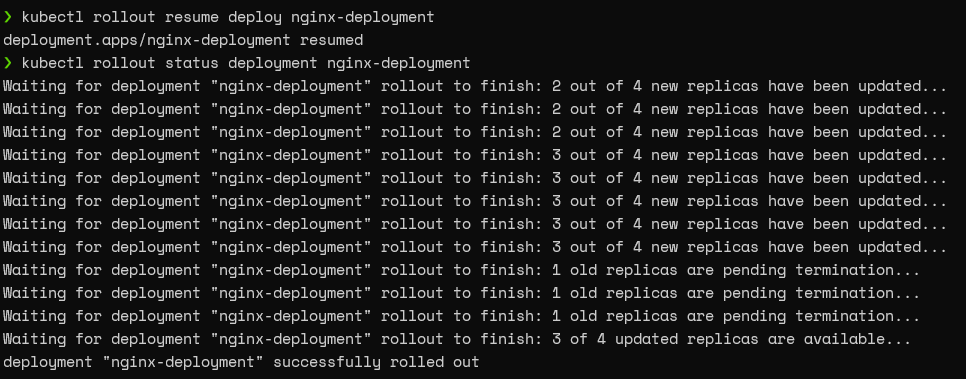
kubectl rollout status deployment nginx-deployment



You can simply run the “rollout resume deploy” command to resume the deployment.

kubectl rollout resume deploy nginx-deployment

kubectl rollout status deployment nginx-deployment

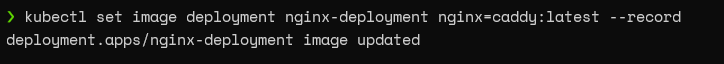


**Rolling back a deployment**

Kubernetes also supports rolling back deployments to the previous revision. This is a crucial feature enabling users to undo changes in deployment.

Let’s assume that we have updated our configuration with an incorrect image. (We will be using the caddy webserver in this example.)

kubectl set image deployment nginx-deployment nginx=caddy:latest --record



kubectl rollout history deployment nginx-deployment

