KarthiKeyan Shanmugam Follow

Dec 8, 2017 · 5 min read · Listen

Save

Kubernetes tutorial — Create deployments using YAML file

This is in continuation on Kubernetes article series.In the last post, we have learnt how to create & deploy the app to the Kubernetes cluster.Now in this post,we are

going to learn how to create application deployment using yaml file also we can check on how to create services to control how application communicates.

Step #1.Create an nginx deployment

Using *Deployment* controller we can provide declarative updates for Pods and ReplicaSets. Create deployment.yaml file in your current folder like the below to describe the nginx deployment.

Kubernetes manifest file defines a desired state for the cluster, including what container images should be running.For example, this YAML file describes a Deployment that runs the nginx:latest Docker image

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

In this example:

- A Deployment named *nginx-deployment* is created, indicated by the metadata: name field.
- The Deployment creates *1 replicated Pods,* indicated by the replicas field.
- The Pod template's specification, or template: spec field, indicates that the Pods run one container, nginx, which runs the nginx Docker Hub latest image (you can also specify the version ex.1.7.9.)
- The Deployment opens port 80 for use by the Pods.

Step #2.Create Deployment based on the YAML file

- Based on the deployment described in deployment.yaml created in the previous step,create the deployment using *kubectl create* command

```
kubectl create -f deployment.yaml
```

Image — Kubectl — Create deployment command

- Now that deployment is created,lets check the deployment information using *kubectl get deployment* command :

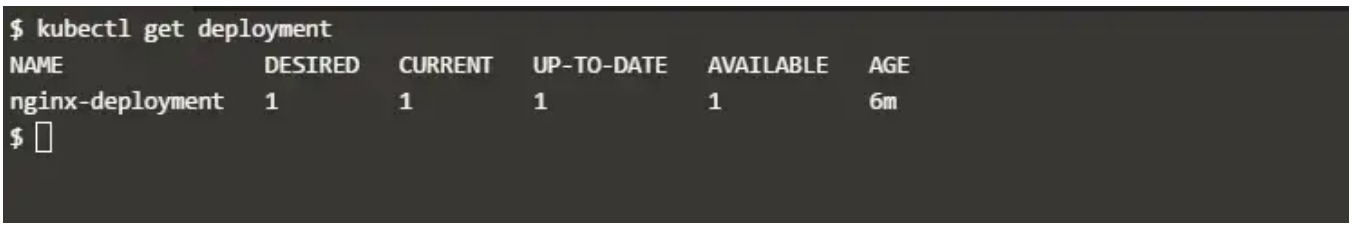

Image — Kubectl — get deployment command

When you inspect the Deployments in your cluster, the following fields are displayed:

- `NAME` lists the names of the Deployments in the cluster.
- `DESIRED` displays the desired number of *replicas* of the application, which you define when you create the Deployment. This is the *desired state*.
- `CURRENT` displays how many replicas are currently running.
- `UP-TO-DATE` displays the number of replicas that have been updated to achieve the desired state.
- `AVAILABLE` displays how many replicas of the application are available to your users.
- `AGE` displays the amount of time that the application has been running.

- Describe the deployment using *kubectl describe deployment* command to check the details on the deployment

```
$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 15 Nov 2017 09:52:27 +0000
```

Open in app ↗

Image- Kubectl — describe deployment command

- Check the list of pods created by the deployment by using *kubectl get pods* command:

```
$ kubectl get pods -l app=nginx
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-2947857529-bnxx0   1/1       Running   0          11m
$
```

Image — Kubectl — get pods command

Step #3.Create service

Kubernetes has powerful networking capabilities that control how applications communicate. These networking configurations can also be controlled via YAML.The Service selects all applications with the label *ngnix*. As multiple replicas, or instances, are deployed, they will be automatically load balanced based on this common label. The Service makes the application available via a NodePort.

Kubernetes `Service` is an abstraction which defines a logical set of `Pods` and a policy by which to access them (micro-service). The set of `Pods` targeted by a `Service` is determined by a `Label Selector`

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-svc
 labels:
 app: nginx
spec:
```

```
type: NodePort
ports:
- port: 80
nodePort: 30080
selector:
app: nginx
```

69

Step #4.Deploy service

- Use *kubectl create command* to create new service based on the service.yaml file created in the previous step
- Check the details of all the Service objects deployed

```
$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.0.0.1     <none>        443/TCP        38m
nginx-svc    NodePort    10.0.0.50    <none>        80:30080/TCP   6s
```

Image — Kubectl — get Svc command command

- Use describe command to discover more details about the configuration of all the Service objects deployed

```
$ kubectl describe svc nginx-svc
Name:                     nginx-svc
Namespace:                default
Labels:                   app=nginx
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP:                       10.0.0.50
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30080/TCP
```

Image — Kubectl — describe svc command command

Use curl command to Issuing requests to the port *30080*

```
$ curl host01:30080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
```

Image — curl command to issue request

Step #5.Update nginx deployment to have 4 replicas

Modify deployment.yaml to update the nginx deployment to have 4 replicas.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

Step #6.Apply the updated nginx deployment to have 4 replicas

- Apply the changes using kubectl apply command

```
kubectl apply -f deployment.yaml
```

- Now that deployment is created,lets check the deployment information using *kubectl get deployment* command :

```
$ kubectl get deployment
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   4         4         4            4           26m
$
```

Image — Kubectl — get deployment command

- When you inspect the Deployments in your cluster, the following fields are displayed:
- `NAME` lists the names of the Deployments in the cluster.
- `DESIRED` displays the desired number of *replicas* of the application, which you define when you create the Deployment. This is the *desired state*.
- `CURRENT` displays how many replicas are currently running.
- `UP-TO-DATE` displays the number of replicas that have been updated to achieve the desired state.
- `AVAILABLE` displays how many replicas of the application are available to your users.
- `AGE` displays the amount of time that the application has been running.
- Describe the deployment using *kubectl describe deployment* command to check the details on the deployment

```
$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Wed, 15 Nov 2017 09:52:27 +0000
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision=1
                        kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"extensions/v1beta1","kind":"Depl
oyment","metadata":{"annotations":{},"name":"nginx-deployment","namespace":"default"},"spec":{"replicas"...
Selector:               app=nginx
Replicas:               4 desired | 4 updated | 4 total | 4 available | 0 unavailable
```

Image — Kubectl — describe deployment command

- Check the list of pods created by the deployment by using *kubectl get pods* command:

```
$ kubectl get pods -l app=nginx
NAME                                READY     STATUS    RESTARTS   AGE
nginx-deployment-2947857529-4q9tj   1/1       Running   0          8m
nginx-deployment-2947857529-bnxx0   1/1       Running   0          34m
nginx-deployment-2947857529-jtvvs   1/1       Running   0          8m
nginx-deployment-2947857529-vz471   1/1       Running   0          8m
$
```

Image — Kubectl — get Pods command

- As all the Pods have the same label selector, they'll be load balanced behind the Service NodePort deployed. Issuing requests to the port will result in different containers processing the request

Like this post? Don't forget to share it!

Additional Resources :

- Kubectl cheat sheet

Originally published at @upnxtblog.

Kubernetes