131

Kubernetes Deployment Tutorial For Beginners

DEVOPSCUBE NOVEMBER 29, 2018 4 COMMENTS

Kubernetes deployment tutorial guide will explain the key concepts in a Kubernetes YAML specification with a Nginx example deployment.

Introduction:

In Kubernetes, pods are the basic units which get deployed in the cluster. Kubernetes deployment is an abstraction layer for the pods. The main purpose of the deployment is to maintain the resources declared in the deployment configuration to be in its desired state. A deployment can be a YAML or JSON declaration.

Key Things To Understand

1. A Deployment can schedule multiple pods. A pod as a unit cannot scale by itself.

2. A Deployments represents a single purpose with a group of PODs.

3. A single POD can have multiple containers and these containers inside a single POD shares the same IP and can talk to each other using localhost address.

4. To get access to a Deployment with one or many PODs, you need a Kubernetes Service endpoint mapped to the deployment using labels and selectors.

5. Deployment should have only stateless services. Any application needs to store the state should be deployed in Kubernetes StatefulSet.

131

Deployment YAML:

Kubernetes deployment Yaml contains the following main specifications.

1. apiVersion
2. Kind
3. metadata
4. spec

Now let's look at each specification in detail.

Note: *In Kubernetes, everything persistent is defined as an object. Example: Deployments, services, Replica Set, Configmap, Jobs etc*

ApiVersion

This specifies the API version of the Kubernetes deployment object. It varies between each Kubernetes version.

How To Use the Right API version: Kubernetes contains three API versions.

1. **Alpha**: This is the early release candidate. It might contain bugs and there is no guarantee that it will work in the future.
   Example: `scalingpolicy.kope.io/v1alpha1`
2. **Beta**: The API's become beta once its alpha tested. It will be in continuous development & testing until it becomes stable. Beta versions will most likely go into the Kubernetes main APIs.Example: `batch/v1beta1`
3. **Stable**: The APIs which does not contain alpha and beta goes into the stable category. Only stable versions are recommended to be used in production systems.
   Example: `apps/v1`

These APIs could belong to different API groups.

An example list of Kubernetes APIs from different API groups from Kubernetes version 1.10.6 is shown below. Deployment object belongs to `apps` API group. You can list these API on http://localhost:8001/ using the kubectl proxy.

131

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/admissionregistration.k8s.io",
    "/apis/admissionregistration.k8s.io/v1beta1",
    "/apis/apiextensions.k8s.io",
    "/apis/apiextensions.k8s.io/v1beta1",
    "/apis/apiregistration.k8s.io",
    "/apis/apiregistration.k8s.io/v1",
    "/apis/apiregistration.k8s.io/v1beta1",
    "/apis/apps",
    "/apis/apps/v1",
    "/apis/apps/v1beta1",
    "/apis/apps/v1beta2",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/autoscaling/v2beta1",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v1beta1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1beta1",
    "/apis/cloud.google.com",
    "/apis/cloud.google.com/v1beta1",
    "/apis/extensions",
    "/apis/extensions/v1beta1",
    "/apis/metrics.k8s.io",
    "/apis/metrics.k8s.io/v1beta1",
    "/apis/networking.k8s.io",
    "/apis/networking.k8s.io/v1",
    "/apis/policy",
    "/apis/policy/v1beta1",
    "/apis/rbac.authorization.k8s.io",
    "/apis/rbac.authorization.k8s.io/v1",
    "/apis/rbac.authorization.k8s.io/v1beta1",
    "/apis/scalingpolicy.kope.io",
    "/apis/scalingpolicy.kope.io/v1alpha1",
    "/apis/storage.k8s.io",
    "/apis/storage.k8s.io/v1",
    "/apis/storage.k8s.io/v1beta1"
    ]
}
```

131

Kind

Kind describes the type of the object/resource to be created. In our case its a deployment object. Following are the main list of objects/resources supported by Kubernetes.

```
componentstatuses
configmaps
daemonsets
deployments
events
endpoints
horizontalpodautoscalers
ingress
jobs
limitranges
namespaces
nodes
pods
persistentvolumes
persistentvolumeclaims
resourcequotas
replicasets
replicationcontrollers
serviceaccounts
services
```

131

Metadata

It is a set of data to uniquely identify a Kubernetes object. Following are the key metadata that can be added to an object.

```
labels
name
namespace
annotations
```

Let’s have a look at each metadata type

1. **Labels**: Key-value pairs primarily used to group and categorize deployment object. It is intended for an object to object grouping and mapping using selectors. For example, kubernetes service uses the pod labels in its selectors to send traffic to the right pods. We will see more about labels and selectors in the service creation section.
2. **Name**: It represents the name of the deployment to be created.
3. **Namespace**: Name of the namespace where you want to create the deployment.
4. **Annotations**: key-value pairs like labels, however, used for different use cases. You can add any information to annotations. For example, you can have an annotation like `"monitoring" : "true` and external sources will be able to find all the objects with this annotation to scrape its metrics. Objects without this annotation will be omitted.

There are other system generated metadata such us UUID, timestamp, resource version etc. that gets added to each deployment.

Example metadata

```
metadata:
  name: resource-name
  namespace: deployment-demo
  labels:
    app: web
    platform: java
    release: 18.0
  annotations:
    monitoring: true
    prod: true
```

Spec

Under spec, we declare the desired state and characteristics of the object we want to have. For example, in deployment spec, we would specify the number of replicas, image name etc. Kubernetes will make sure all the declaration under the spec is brought to the desired state.

READ **Kubernetes Certification Coupon: 16% Off + $100 Off On Kubernetes Course bundle**

Spec has three important subfields.

1. **Replicas**: It will make sure the numbers of pods running all the time for the deployment. Example,

```
spec:
  replicas: 3
```

2. **Selector**: It defines the labels that match the pods for the deployments to manage. Example,

```
selector:
    matchLabels:
      app: nginx
```

3. **Template**: It has its own metadata and spec. Spec will have all the container information a pod should have. Container image info, port information, ENV variables, command arguments etc. Example,

```
template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
```

131

```
      - image: nginx
        name: nginx
```

Kubernetes Example Deployment

Since we have looked at the basics let start with an example deployment. We will do the following in this section.

1. Create a namespace
2. Create a Nginx Deployment
3. Create a Nginx Service
4. Expose and access the Nginx Service

Note: *Few of the operations we perform in this example can be performed with just kubectl and without a YAML Declaration. However, we are using the YAML specifications for all operations to understand it better.*

131

Exercise Folder

To begin the exercise, create a folder names deployment-demo and cd into that folder. Create all the exercise files in this folder.

```
mkdir deployment-demo && cd deployment-demo
```

Create A Namespace

Let's create a YAML named **namespace.yaml** file for creating the namespace.

```
apiVersion: v1
kind: Namespace
metadata:
  name: deployment-demo
  labels:
    apps: web-based
  annotations:
    type: demo
```

Use kubectl command to create the namespace.

```
kubectl create -f namespace.yaml
```

Equivalent kubectl command

```
kubectl create namespace deployment-demo
```

Assign Resource Quota To Namespace

Now let's assign some resource quota limits to our newly created namespace. This will make sure the pods deployed in this namespace will not consume more system resources than mentioned in the resource quota.

Create a file named **resourceQuota.yaml**. Here is the resource quota YAML contents.

131

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-quota
  namespace: deployment-demo
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
```

Create the resource quota using the YAML.

```
kubectl create -f resourceQuota.yaml
```

Now, let's describe the namespace to check if the resource quota has been applied to the deployment-demo namespace.

```
kubectl describe ns deployment-demo
```

The output should look like the following.

```
Name:          deployment-demo
Labels:        apps=web-based
Annotations:   type=demo
Status:        Active

Resource Quotas
 Name:            mem-cpu-quota
 Resource         Used  Hard
 --------         ---   ---
 limits.cpu       0     2
 limits.memory    0     2Gi
 requests.cpu     0     1
 requests.memory  0     1Gi
```

Create A Deployment

We will use the public Nginx image for this deployment.

Create a file named **deployment.yaml** and copy the following YAML to the file.

Note*: This deployment YAML has minimal required information we discussed above. You can have more specification in the deployment YAML based on the requirement.*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
  namespace: deployment-demo
  annotations:
    monitoring: "true"
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
        ports:
        - containerPort: 80
        resources:
          limits:
            memory: "2Gi"
            cpu: "1000m"
          requests:
            memory: "1Gi"
            cpu: "500m"
```

131

Under containers, we have defined its resource limits, requests and container port (one exposed in Dockerfile).

Create the deployment using kubectl

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

Check the deployment

```
kubectl get deployments -n deployment-demo
```

Even though we have added minimal information, after deployment, Kubernetes will add more information to the deployment such as resourceVersion, uid, status etc.

You can check it by describing the deployment in YAML format using the kubectl command.

```
kubectl get deployment nginx -n deployment-demo  --output yaml
```

131

Create A Service And Expose The Deployment

Now that we have a running deployment, we will create a Kubernetes service of type NodePort (30500) pointing to the nginx deployment. Using NodePort you will be able to access the Nginx service on all the kubernetes node on port 30500.

Create a file named **service.yaml** and copy the following contents.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: nginx
  namespace: deployment-demo
spec:
  ports:
  - nodePort: 30500
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: NodePort
```

Service is the best example for explaining **labels and selectors**. In this service, we have a selector with “app” = “nginx” label. Using this, the service will be able to match the pods in our nginx deployment as the deployment and the pods have the same label. So automatically all the requests coming to the nginx service will be sent to the nginx deployment.

Let’s create the service using kubectl command.

```
kubectl create -f service.yaml
```

You can view the service created using kubectl command.

```
kubectl get services  -n deployment-demo
```

131

Now, you will be able to access the nginx service on any one of the kubernetes node IP on port 30500

For example,

```
http://35.134.110.153:30500/
```

Free DevOps Resources

Get DevOps news, tutorials and resources in your inbox. A perfect way If you want to get started with devops. Like you, we dont like spam.

Email SUBSCRIBE

CATEGORIES COMMON

Twitter Facebook Google+ Pinterest Tumblr

131

devopscube

Established in 2014, a community for developers and system admins. Our goal is to continue to build a growing DevOps community offering the best in-depth articles, interviews, event listings, whitepapers, infographics and much more on DevOps.

Related Articles

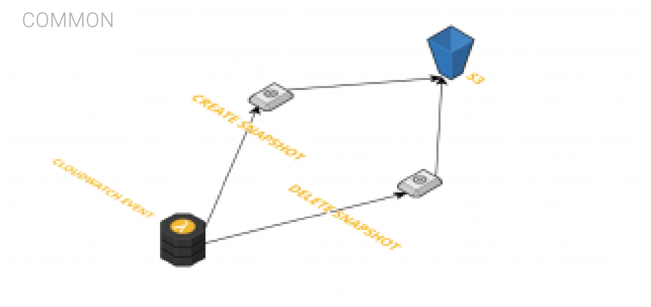
COMMON

How To Get Six Months Pluralsight Free Subscription

COMMON

Kubernetes Certification Tips From A Kubernetes Certified Administrator

COMMON



How To Automate EBS Snapshot Creation And Deletion Using Lambda

131

What do you think?

3 Responses

Upvote · Funny · Love · Surprised · Angry · Sad

4 Comments **devopscube** 1 **Login**

Recommend · Tweet · Share · **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Jon Laberge • 4 months ago

As a beginner, for alpha and beta, do you have the names mixed up? Under alpha it shows beta keyword, under beta, it shows alpha keyword. Is this correct or indeed it is mixed up? Slightly confusing.

Reply • Share ›

Devopscube Mod → Jon Laberge • 4 months ago

@j**@Jon Laberge** it was a mistake from our side. Thanks for letting us know. It corrected now.

Reply • Share ›

Jon Laberge ➦ Devopscube • 3 months ago

Great!

^ | ∨ • Reply • Share ›

Devopscube Mod ➦ Jon Laberge • 3 months ago

Thanks Jon!

^ | ∨ • Reply • Share ›

ALSO ON **DEVOPSCUBE**

Setup Jenkins master and Build Slaves as Docker Container

9 comments • 6 months ago

Avatar monn — Hi,Thanks for the article!On my Jenkins, I don't see the 'docker container' and 'docker template definition' on the …

Kubernetes Certification Coupon: 16% Off + $100 Off On Kubernetes Course …

2 comments • 6 months ago

Avatar devopscube — Hi, We checked with Linux Foundation. It's a known error. Please create the account first then use the …

How To Run/Access Jenkins on Port 80 in Linux

2 comments • 6 months ago

Avatar devopscube — Correct! Docker makes the setup very easy. However, many organizations have not started using

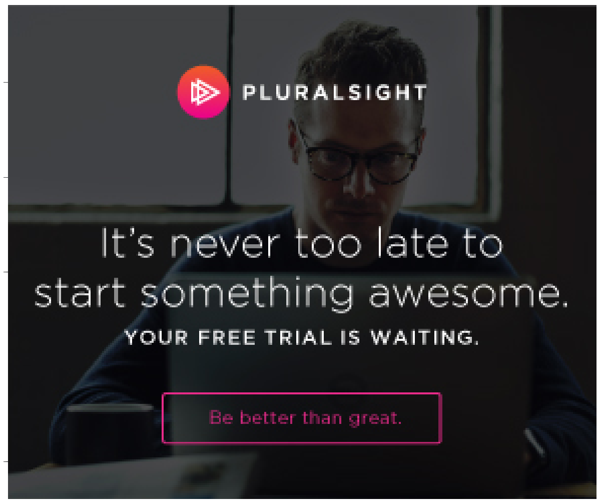
Service Discovery and Other Cluster Management Techniques Using Consul

1 comment • 6 months ago

Avatar riki — Still don't know how service communicate each other.let say I have :1. user service at 127.0.0.1:8000 and 2

131

Search … SEARCH


PLURALSIGHT
It's never too late to start something awesome.
YOUR FREE TRIAL IS WAITING.
Be better than great.

131

RECENT POSTS

How To Get Six Months Pluralsight Free Subscription

February 26, 2019

How To Setup Jenkins Slaves Using Password And Ssh Keys

February 23, 2019

Packer Tutorial For Beginners – Automate AMI Creation

January 30, 2019

List Of Best Docker Container Clustering And Orchestration Tools/Services

January 27, 2019

List Of DevOps Blogs And Resources For Learning

January 25, 2019

How To Get Six Months Pluralsight Free Subscription

How to Attach and Mount an EBS volume to EC2 Linux Instance

How to Setup Prometheus Monitoring On Kubernetes Cluster

How to Setup Docker containers as Build Slaves for Jenkins

How To Setup and Configure a Proxy Server - Squid Proxy

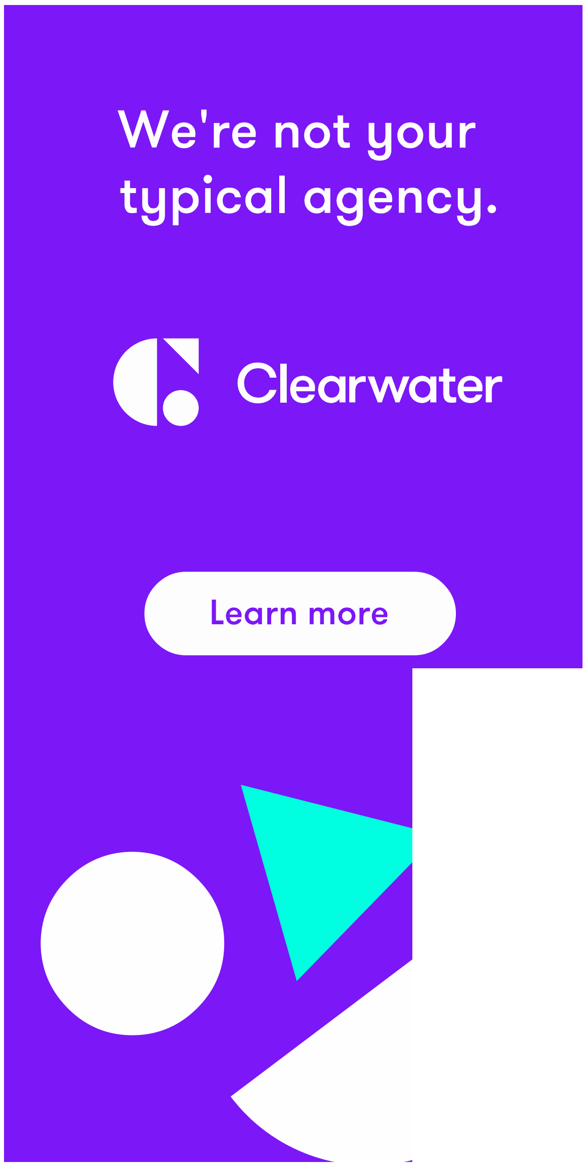
10 Devops Tools for Infrastructure Automation and Monitoring

How To Backup Jenkins Data and Configurations

How To Monitor Linux Servers Using Prometheus Node Exporter

How to Install and configure Sonarqube on Linux (RHEL/Centos/ec2)

131


We're not your typical agency.
Clearwater
Learn more

131

treehouse™
Learn web design, coding, mobile app development & more.
Start Learning for Free

Advertise Contribute Resources DEVOPS

131