Deployment管理与使用

前言

- 本章节主要介绍Deployment相关概念，包括什么是控制器，如何创建Deployment，什么是kubectl，如何进行deployment的扩容和升级等。

目标

- 学完本课程后，您将能够：
  - 描述kubernetes中各类控制器
  - 创建和使用deployment
  - 使用kubectl命令行工具

目录

Kubernetes管理对象 (1)

- Pod
  - Kubernetes基本管理单元，每个Pod是一个或多个容器的一组集合。
  - 一个Pod作为一个整体运行在一个节点（node）上。
  - Pod内的容器共享存储和网络
- ReplicationController（简称RC）
  - Kubernetes需要管理大量的Pod，而显而易见的是通常情况下一个应用不会以单独的一个Pod完成。比较常见的情况是使用大量的Pod组成一个简单应用。管理这些大量的Pod的一个方式就是RC。
  - RC可以指定Pod的副本数量，并且在其中有Pod故障时可以自动拉起新的Pod，大大简化了管理难度。

- 目前建议使用ReplicaSet和Deployment代替RC。

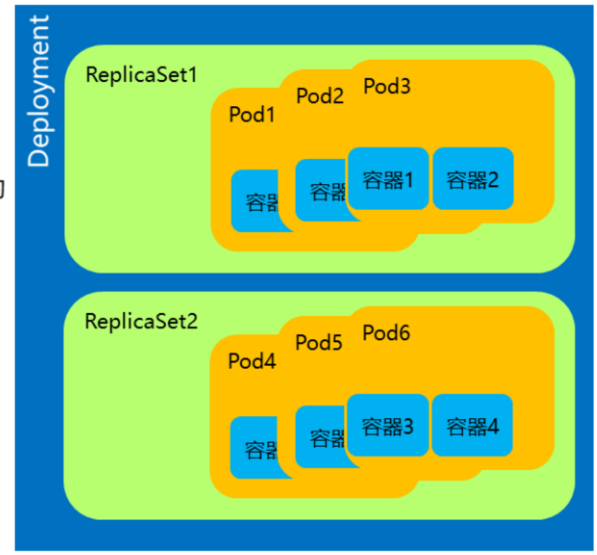
Kubernetes管理对象 (2)

- ReplicaSet（简称RS）
  - ReplicaSet是新一代的RC，主要功能和RC一样，维持Pod的数量稳定，指定Pod的运行位置等，使用方法也相似，主要区别是更新了api，支持更多功能。
  - ReplicaSet不建议直接使用，而是用更上层的概念Deployment调用ReplicaSet。
- Deployment
  - 目前最常用的控制器就是Deployment，创建Deployment时也会自动创建ReplicaSet。
  - Deployment可以管理一个或多个RS，并且通过RS来管理Pod。

- 更多功能指选择器功能。

Kubernetes管理对象 (3)

- 从小到大的管理逻辑
  - 容器<Pod<ReplicaSet<Deployment
- 通常情况下
  - Pod中包含一个容器，或关系特别紧密的几个容器。
  - 一个ReplicaSet中包含一个或多个相同的Pod。
  - Deployment中包含一个或几个不同的RS。



HUAWEI

- 一个完整的应用由多个Deployment构成。

目录

运行一个Deployment

- 创建一个简单的deployment

```
kubectl create deployment mydep  --image=nginx
```

- 完成后我们使用如下语句查看deployment的创建情况

```
kubectl get deployment
```

- 回显：

```
NAME    READY   UP-TO-DATE   AVAILABLE   AGE
mydep   1/1     1            1           2m3s
```

命令行接口 - Kubectl

- 在Kubernetes中的操作很多都是用kubectl来完成，通过其命令可以管理Deployment，Replicaset，ReplicationController，Pod等，进行操作、扩容、删除等全生命周期操作。同时可以对管理对象进行查看或者监控资源使用情况。
- Kubectl的语法

```
kubectl [command] [TYPE] [NAME] [flags]
```

  - Command：指定你希望进行的操作，如create，get，describe，delete等。
  - TYPE：指定操作对象的类型，如deployment，RS，Pod等
  - NAME：指定对象的名字
  - flags：可选的标志位

- 一个完整的kubectl语法不一定要具备所有元素

- 常用flags包括-n=namespace, -o=wide等

使用yaml文件创建Deployment (1)

- 在前面的样例中，我们使用一行命令创建了Deployment。这是一种简单的形式，大量个性化参数没有定义，后续对该Deployment的升级管理也有诸多问题。在实际使用中，我们更常见的用法是通过一个yaml文件来创建各类资源。
- 创建一个yaml文件

```
vi nginx-deployment.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```


HUAWEI

- 建议实际使用中全部使用yaml文件创建对象

使用yaml文件创建Deployment (2)

- 从yaml文件创建deployment

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

- 查看创建结果

```
kubectl get deployment
```

- 回显如下

```
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           114s
```


- create –f 指从某个文件创建对应资源
- 可用apply代替create

Yaml文件格式 (1)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
……
```

- apiVersion：版本号，固定为apps/v1，如果使用1.9.0以前版本的kubernetes，填写apps/v1beta2
- kind：类型，选择创建资源类型，可以填写pod，replicaset等
- metadata：元数据，其中name项指定了名称，label项指定标签。
- spec：deployment规格，其中replicas指定pod副本数量，选择器选择标签匹配为app：nginx

- 选择器的详细信息我们在后面章节会介绍

Yaml文件格式 (2)

```
……
spec:
……
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

- template：对pod模板的定义，其中至少要定义一个label
- spec：描述pod的规格
- containers：定义容器的属性，在范例中，容器名字是nginx，镜像为nginx:1.7.9，容器输入输出的端口是80端口。
- 最后注意格式，缩进一般使用两个空格，千万不要使用tab！

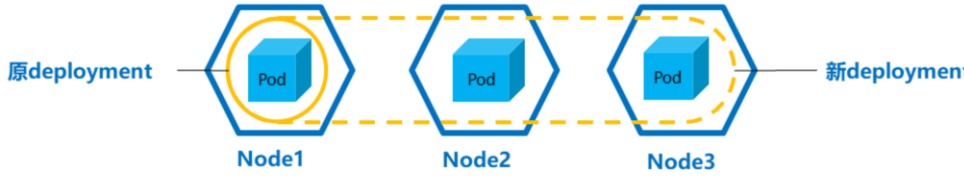
- 有短横杠的地方表示是列表格式，不可忽略短横杠。

目录

HUAWEI

Deployment弹性伸缩 (1)

- 容器对比虚拟机，最大的优势在于可以灵活的弹性伸缩，而这一部分工作由Kubernetes进行调度。



HUAWEI

Deployment弹性伸缩 (2)

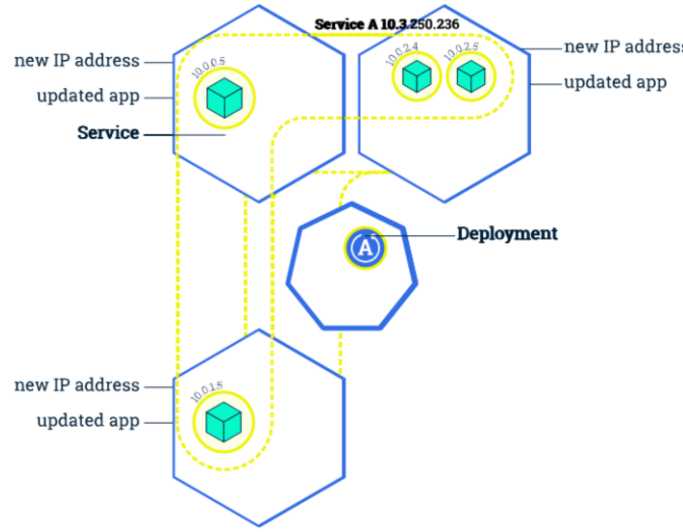
- Deployment弹性伸缩本质是Pod数量增加或减少。
- 弹性伸缩可以支持自动化部署，并在很短时间内实现数量变更。
- 弹性伸缩通过修改yaml文件中的replicas参数实现。
- 修改yaml后使用scale命令应用变更完成扩容或减容。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 5
  selector:
    matchLabels:
      app: nginx
……
```


- 可用apply命令代替scale

滚动更新 (1)

- 当使用的deployment需要升级时（如软件版本更新），可以使用rolling update功能滚动升级deployment中所有pod。



HUAWEI

滚动更新 (2)

- 已有一个nginx-deployment，查看它现在的状态。

```
[root@k8s-master]# kubectl get rs
NAME                               DESIRED   CURRENT   READY   AGE
nginx-deployment-6dd86d77d         3         3         3       11s
[root@k8s-master]# kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
nginx-deployment-6dd86d77d-7vlmb   1/1     Running   0          3s
nginx-deployment-6dd86d77d-92cqm   1/1     Running   0          3s
nginx-deployment-6dd86d77d-16pw8   1/1     Running   0          3s
```

- 修改yaml文件
- 执行rolling-update

```
……
spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
```


- Rolling-update可用apply命令

滚动更新 (3)

- 升级后，再次查看状态，会发现replicaset和pod的状态都发生了变化
  - 出现了一个新的replicaset，原有replicaset中无pod
  - 三个pod的名字发生了变更

```
[root@k8s-master]# kubectl get rs
NAME                                DESIRED   CURRENT   READY   AGE
nginx-deployment-6dd86d77d          0         0         0       63s
nginx-deployment-784b7cc96d         3         3         3       21s
[root@k8s-master runfile]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-784b7cc96d-4wlnl   1/1     Running   0          9s
nginx-deployment-784b7cc96d-j72jm   1/1     Running   0          12s
nginx-deployment-784b7cc96d-kbx6n   1/1     Running   0          10s
```

为什么会有这些变化?

- 新的pod部署在新的rs中

滚动更新 (4)

- 再次查看该deployment日志，可以发现在滚动更新中系统所做的操作。

```
Events:
  Type    Reason             Age   Message
  ----    ------             ----  -------
  Normal  ScalingReplicaSet  50s   Scaled up replica set nginx-deployment-784b7cc96d to 1
  Normal  ScalingReplicaSet  48s   Scaled down replica set nginx-deployment-6dd86d77d to 2
  Normal  ScalingReplicaSet  48s   Scaled up replica set nginx-deployment-784b7cc96d to 2
  Normal  ScalingReplicaSet  47s   Scaled down replica set nginx-deployment-6dd86d77d to 1
  Normal  ScalingReplicaSet  47s   Scaled up replica set nginx-deployment-784b7cc96d to 3
  Normal  ScalingReplicaSet  45s   Scaled down replica set nginx-deployment-6dd86d77d to 0
```


- Deployment控制器依次在新的replicaset中创建pod，并依次将原有replicaset中的pod数量减少，最终用新的replicaset代替原有，实现了所有pod的滚动更新。

回滚 (1)

- 使用kubernetes滚动更新后，kubernetes会记录下本次更新，并且保存为一个历史版本，如果更新后出现应用异常，可以通过回滚操作回到之前版本。

```
kubectl apply -f nginx-deployment.v1.yaml --record
kubectl apply -f nginx-deployment.v2.yaml --record
kubectl apply -f nginx-deployment.v3.yaml --record
```


- 在初始创建deployment时，使用nginx1.7.9版本，后续使用另外两个yaml文件，修改nginx版本号，进行两次滚动更新操作，更新到1.8.1和1.9.1，使用apply命令时附加参数--record记录下这两次更新。

回滚 (2)

- 使用命令查看历史版本

```
[root@k8s-master]# kubectl rollout history deployment nginx-deployment
deployment.extensions/nginx-deployment
REVISION  CHANGE-CAUSE
1         kubectl apply --filename=nginx-deployment.v1.yaml --record=true
2         kubectl apply --filename=nginx-deployment.v2.yaml --record=true
3         kubectl apply --filename=nginx-deployment.v3.yaml --record=true
```

- 系统记录下了三条信息，分别是：
  - 第一次是创建nginx-deployment
  - 第二次是滚动更新nginx-deployment
  - 第三次是再次滚动更新nginx-deployment

回滚 (3)

- 通--revision=命令可以查看某个历史版本的详细信息。

```
[root@k8s-master]# kubectl rollout history deployment nginx-deployment --
revision=2
deployment.extensions/nginx-deployment with revision #2
Pod Template:
  Labels:       app=nginx
        pod-template-hash=59988f74c7
  Annotations:  kubernetes.io/change-cause: kubectl apply --filename=nginx-
deployment.v2.yaml --record=true
  Containers:
   nginx:
    Image:      nginx:1.8.1
    Port:       80/TCP
    Host Port:  0/TCP
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>
```

- 使用rollout undo命令回滚到指定版本。

```
kubectl rollout undo deployment nginx-deployment --to-revision=2
```


HUAWEI

- 标红的部分要特别注意

实验&实训任务

- 实验任务
  - 请按照实验手册2.3章节完成Deployment相关实验，包括：
    - 运行Deployment
    - 使用kubectl命令行工具
    - 使用yaml文件创建Deployment
    - 弹性伸缩Deployment
    - 滚动升级deployment
- 实训任务
  - 请灵活使用本章节课程及实验手册中学到的知识，按照实验手册2.3.6章节完成Deployment实训任务。

本章总结

- 本章节介绍了deployment及相关知识，包括如下：
  - 什么是kubernetes控制器
  - 如何使用kubectl命令行
  - 如何编写deployment的yaml文件
  - 如何实现deployment的升级和弹性伸缩


谢 谢
www.huawei.com