37 Kubernetes Deployment 使用

更新时间：2020-10-21 09:41:04

> 合理安排时间，就等于节约时间。——培根

在上一篇文章介绍完 ReplicationController 和 ReplicaSet 之后，我们这一篇文章来介绍一下目前用来替代这两种控制的 Deployment。

1. 使用场景

目前的 Deployment 的典型使用场景和 ReplicaSet 类似，都是 Pod 的一种多副本控制器，但是相比于 ReplicaSet，Deployment 在一些更新和扩缩容操作上面更加友好，所以现在基本不再直接使用 ReplicaSet，而是使用 Deployment 来代替，下面介绍一些 Deployment 的典型使用场景。

2. 创建 Deployment 对象

创建 Deployment 对象，我们需要编写一个 Deployment 的描述文件：

```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:1.14.2
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

如果对上一篇介绍 ReplicaSet 的文章还有印象的话，可以发现 Deployment 的声明和 ReplicaSet 非常的相似。这个 Deployment 描述会创建一个名字叫 nginx-deployment 拥有三个 Pod 副本的 Deployment。

这里的标签选择器这里使用的是简单的 kv 选择器 matchLabels，实际上 Deployment 的标签选择器也是支持 matchExpressions 形式的。下面是是 matchExpressions 的举例。

```yaml
- matchExpressions:
  - key: kubernetes.io/e2e-az-name
    operator: In
    values:
    - e2e-az1
    - e2e-az2
```

下面我们通过 `kubectl apply` 创建这个 Deployment，我们这里是应用到 imooc 这个 namespace。

```
$ kubectl apply -f nginx-dm.yaml -n imooc
```

然后可以通过 `kubectl get deployment` 检查 Deployment 的创建情况。

```
$ kubectl get deployment -n imooc
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           2m31s
```

通过 `kubectl describe deployment` 查看 Deployment 的情况。

```
$ kubectl describe deployment nginx-deployment -n imooc
Name:                   nginx-deployment
Namespace:              imooc
CreationTimestamp:      Sun, 12 Apr 2020 11:43:04 +0800
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 1
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"nginx"},"name":"nginx-deployment","namespace
":"i...
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:      nginx:1.14.2
    Port:       80/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-64969b6699 (3/3 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  4m8s  deployment-controller  Scaled up replica set nginx-deployment-64969b6699 to 3
```

通过上面的输出我们可以看到如下几个重要信息。

- 基本信息：包括 Name，Namespace，CreationTimestamp，Labels 等信息；
- selector：标签选择器；；
- Replicas：Pod 副本的运行情况；
- StrategyType：表示升级的时候如何使用新的 Pod 替换旧的 Pod。这里的值是 RollingUpdate，也就是滚动更新；除了 RollingUpdate，还支持 Recreate，表示在新建 Pod 之前将老的 Pod 都删除。Deployment 默认使用 RollingUpdate StrategeType；
- RollingUpdateStrategy：对于使用 RollingUpdate StrategyType 的情况，我们可以指定 maxUnavailable 和 maxSurge 来控制滚动更新操作：
  - maxUnavailable：表示在更新过程中最大不可用的 Pod 数，可以为绝对值也可以是百分比，默认值为 25%；
  - maxSurge：表示能够额外创建的副本数。当 maxSurge 为 0 时，maxUnavailble 不能为 0，因为这两个同时为 0 的话就死锁了。maxSurge 取值也可以是百分比或者绝对值，默认值是 25%；
- Pod Template：定义了该 Deployment 管理的 Pod；

- oldReplicaSets/NewReplicaSet：这个是什么情况呢？从 old 和 new 我们可以推测一下有可能 Deployment 每次更新都是更新 ReplicaSet，然后 ReplicaSet 是通过 Deployment 来进行管理。实际上确实是这样的，ReplicaSet 作为 Pod 的多副本控制器，很少会直接使用，而是通过 Deployment 来间接管理 ReplicaSet。Deployment 相比 ReplicaSet 在 Pod 的更新，扩缩容上支持的更好；
- Events：我们可以看到该 Deployment 创建了一个 ReplicaSet nginx-deployment-64969b6699，下面我们通过命令 `kubectl get rs` 来看一下：

```
$ kubectl get rs -n imooc
NAME                          DESIRED   CURRENT   READY   AGE
nginx-deployment-64969b6699   3         3         3       144m
```

这个 ReplicaSet 的名字正是上面 Deployment 的描述 event 所显示的。我们通过 `kubectl describe rs` 看一下该 ReplicaSet 的描述。

```
$ kubectl describe rs nginx-deployment-64969b6699 -n imooc
Name:           nginx-deployment-64969b6699
Namespace:      imooc
Selector:       app=nginx,pod-template-hash=64969b6699
Labels:         app=nginx
                pod-template-hash=64969b6699
Annotations:    deployment.kubernetes.io/desired-replicas: 3
                deployment.kubernetes.io/max-replicas: 4
                deployment.kubernetes.io/revision: 1
Controlled By:  Deployment/nginx-deployment
Replicas:       3 current / 3 desired
Pods Status:    3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx
           pod-template-hash=64969b6699
  Containers:
   nginx:
    Image:      nginx:1.14.2
    Port:       80/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:           <none>
```

上面的输出的第 10 行显示的 `Controlled By` 正是我们所创建的 Deployment。如果是直接创建的 ReplicaSet 是没有这个 `Contorlled By` 字段域的。

我们下面通过 `kubectl get pods` 来查看一下该 Deployment 间接创建出来的 Pod。

```
□ deployment kubectl get pods -n imooc
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-64969b6699-6rjgj   1/1     Running   0          148m
nginx-deployment-64969b6699-gw5fx   1/1     Running   0          148m
nginx-deployment-64969b6699-jqvqh   1/1     Running   0          148m
```

可以看出来 pod 的名字是 ReplicaSet 的名字加一个随机的字符串。我们使用 `kubectl describe pod` 来查看一下。

```
$ kubectl describe pod nginx-deployment-64969b6699-6rjgj -n imooc
Name:         nginx-deployment-64969b6699-6rjgj
Namespace:    imooc
Priority:     0
Node:         cn-beijing.172.16.60.188/172.16.60.188
Start Time:   Sun, 12 Apr 2020 11:43:05 +0800
Labels:       app=nginx
              pod-template-hash=64969b6699
Annotations:  <none>
Status:       Running
IP:           10.1.1.142
IPs:          <none>
Controlled By:  ReplicaSet/nginx-deployment-64969b6699
Containers:
  nginx:
    Container ID:   docker://4f1003385c63d073e59b64b236d210a74b0434a892138df403ee34b75e2ad259
    Image:          nginx:1.14.2
    Image ID:       docker-pullable://nginx@sha256:f7988fb6c02e0ce69257d9bd9cf37ae20a60f1df7563c3a2a6abe24160306b8d
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 12 Apr 2020 11:43:07 +0800
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-84db9 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-84db9:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-84db9
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:          <none>
```

上面的输出的第 13 行也可以看到一个 `Controlled By` 字段域，显示为我们上面看到的 ReplicaSet。

所以这里我们得出一个结论，Deployment 创建的过程会首先创建一个 ReplicaSet，然后由 ReplicaSet 间接创建 Pod。Deployment 负责管理 ReplicaSet，ReplicaSet 负责管理 Pod。

但是上面 Deployment 描述中的 oldReplicaSet 和 NewReplicaSet 的问题还没有得到解决，我们有理由猜测 Deployment 每次更新都会将老的 ReplicaSet 进行删除，并新建 ReplicaSet。我们下面来看看 Deployment 的更新。

3. Deployment 的更新

Deployment 的更新支持多种方式的更新，比如通过命令行，或者修改 yaml 文件。我们这里演示一下修改 yaml 文件的形式，这种形式是一种类似声明式 API 的方式，不管是创建（create）还是更新（update），都只需要修改同一个 yaml 文件，然后调用 `kubectl apply` 即可。

比如我们将上面 yaml 中的 `replicas: 3` 改为 `replicas: 4`，然后再调用 `kubectl apply` 我们就会发现 Pod 变成了 4 个。

```
$ kubectl apply -f nginx-dm.yaml -n imooc
deployment.apps/nginx-deployment configured
$ kubectl get pods -n imooc
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-64969b6699-6rjgj   1/1     Running   0          166m
nginx-deployment-64969b6699-gw5fx   1/1     Running   0          166m
nginx-deployment-64969b6699-jqvqh   1/1     Running   0          166m
nginx-deployment-64969b6699-xnfkz   1/1     Running   0          5m21s
```

这种形式的更新只需要将原 ReplicaSet 管理的 Pod 增加一个即可，并不会涉及到 ReplicaSet 的变动。我们下面看一个涉及到 ReplicaSet 变动的例子，修改 Pod Template 中的镜像的版本，将 nginx 的版本改为 1.9.1 版本

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 4
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.9.1
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

使用 `kubectl apply` 将变更应用一下。

```
$ kubectl apply -f nginx-dm.yaml -n imooc
deployment.apps/nginx-deployment configured
```

如果你的手速够快，可以通过 `kubectl rollout status` 查看这个变更过程，幸运的话会看到类似下面的输出。

```
$ kubectl rollout status deployment nginx-deployment
Waiting for rollout to finish: 1 out of 2 new replicas have been updated...
```

但是很多情况下还没有来得及查看就已经变更完成了。

```
$ kubectl rollout status deployment nginx-deployment
deployment "nginx-deployment" successfully rolled out
```

再回到我们之前说的那个 OldReplicaSet 和 NewReplicaSet 的问题，通过 `kubectl describe` 查看一下 Deployment 的描述信息。

```
$ kubectl describe deployment nginx-deployment -n imooc
Name:                   nginx-deployment
Namespace:              imooc
CreationTimestamp:      Sun, 12 Apr 2020 11:43:04 +0800
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 2
                        kubectl.kubernetes.io/last-applied-configuration:
                          {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{},"labels":{"app":"nginx"},"name":"nginx-deployment","namespace
":"i...
Selector:               app=nginx
Replicas:               4 desired | 4 updated | 4 total | 4 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:      nginx:1.9.1
    Port:       80/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
  Progressing   True    NewReplicaSetAvailable
OldReplicaSets:  nginx-deployment-64969b6699 (4/4 replicas created)
NewReplicaSet:   nginx-deployment-c464767dd (4/4 replicas created)
Events:
  Type    Reason             Age    From                   Message
  ----    ------             ----   ----                   -------
  Normal  ScalingReplicaSet  24m    deployment-controller  Scaled up replica set nginx-deployment-64969b6699 to 4
  Normal  ScalingReplicaSet  6m50s  deployment-controller  Scaled up replica set nginx-deployment-c464767dd to 1
  Normal  ScalingReplicaSet  6m50s  deployment-controller  Scaled down replica set nginx-deployment-64969b6699 to 3
  Normal  ScalingReplicaSet  6m50s  deployment-controller  Scaled up replica set nginx-deployment-c464767dd to 2
  Normal  ScalingReplicaSet  6m37s  deployment-controller  Scaled down replica set nginx-deployment-64969b6699 to 2
  Normal  ScalingReplicaSet  6m37s  deployment-controller  Scaled up replica set nginx-deployment-c464767dd to 3
  Normal  ScalingReplicaSet  6m37s  deployment-controller  Scaled down replica set nginx-deployment-64969b6699 to 1
  Normal  ScalingReplicaSet  6m37s  deployment-controller  Scaled up replica set nginx-deployment-c464767dd to 4
  Normal  ScalingReplicaSet  6m35s  deployment-controller  Scaled down replica set nginx-deployment-64969b6699 to 0
```

通过输出我们可以看到 Deployment 的描诉信息里面的 OldReplicaSets 显示的确实是老的 ReplicaSet 对象，而 NewReplicaSet 显示的为新的 ReplicaSet 对象。（注：一旦切换完成，OldReplicaSets 就会显示为 <none>，所以有时候虽然发生了变更，但是 OldReplicaSet 显示还是空也是没有问题的）

我们从 Deployment 的 Events 里面（41 行开始）来看一下变更的具体过程是如何发生的，需要注意的时，此时 maxUnavailable 和 maxSurge 都是 25%，也就是容许最多有 4 * 25% = 1 个 Pod 处于不可用状态，容器最多额外创建出来 1 个 Pod。

1. 新 ReplicaSet `nginx-deployment-c464767dd` 做扩容新建出 1 个 Pod；
2. 老 ReplicaSet `nginx-deployment-64969b6699` 做缩容，从 4 个 Pod 缩至 3 个 Pod；
3. 新 ReplicaSet `nginx-deployment-c464767dd` 做扩容新建出 2 个 Pod；
4. 老 ReplicaSet `nginx-deployment-64969b6699` 做缩容，从 3 个 Pod 缩至 2 个 Pod；
5. 新 ReplicaSet `nginx-deployment-c464767dd` 做扩容新建出 3 个 Pod；
6. 老 ReplicaSet `nginx-deployment-64969b6699` 做缩容，从 2 个 Pod 缩至 1 个 Pod；
7. 新 ReplicaSet `nginx-deployment-c464767dd` 做扩容新建出 4 个 Pod；
8. 老 ReplicaSet `nginx-deployment-64969b6699` 做缩容，从 1 个 Pod 缩至 0 个 Pod。

所以可以得出结论，**Deployment 的更新实际上就是两个 ReplicaSet 通过 StrategyType 做更新的过程。**

4. Deployment 的回滚

我们在日常开发中有时候在做发布的时候发布了异常的版本，这时候就需要我们做回滚操作将线上版本回滚到上一个版本，在 Kubernetes 中通过 Deployment 管理我们的应用的时候也可以进行回滚。

首先我们通过 `kubectl rollout history` 查看历史版本，但是因为我们在做 `kubectl apply` 操作的时候没有设置 `--record=true`（这个选项默认为 false），所以这里的 CHANGE-CAUSE 显示为空。

```
$ kubectl rollout history deployment nginx-deployment -n imooc
deployment.extensions/nginx-deployment
REVISION  CHANGE-CAUSE
1         <none>
2         <none>
3         <none>
```

但是我们可以通过指定 `--revision` 参数来显示每个版本具体的行为，如下所示，我们查看一下 #1 版本的信息。

```
$ kubectl rollout history deployment nginx-deployment -n imooc --revision=1
deployment.extensions/nginx-deployment with revision #1
Pod Template:
  Labels: app=nginx
 pod-template-hash=64969b6699
  Containers:
   nginx:
    Image: nginx:1.14.2
    Port: 80/TCP
    Host Port: 0/TCP
    Limits:
      cpu: 100m
      memory: 200Mi
    Requests:
      cpu: 100m
      memory: 200Mi
    Environment: <none>
    Mounts: <none>
  Volumes: <none>
```

回滚的话我们可以回滚到上一次修改的版本。

```
$ kubectl rollout undo deployment nginx-deployment -n imooc
```

或者回滚到指定的某个版本。

```
$ kubectl rollout undo deployment nginx-deployment -n imooc --to-revisoin=1
```

回滚完之后我们可以通过 `kubectl describe` 查看 Deployment 的明细信息来查看是否回滚成功。

5. 缩放 Deployment

缩放 Deployment 也是常用的一个操作，比如流量高峰期，扩容出更多的 Pod。我们可以通过如下的命令来进行自动的扩缩容。

```
kubectl scale deployment nginx-deployment --replicas=7
```

```
$ kubectl get pods -n imooc
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-57f49c59d-8dzn4    1/1     Running   0          5m5s
nginx-deployment-57f49c59d-9jvrp    1/1     Running   0          5m5s
nginx-deployment-57f49c59d-lddjm    1/1     Running   0          5m3s
nginx-deployment-57f49c59d-m57sr    1/1     Running   0          5m5s
nginx-deployment-57f49c59d-q6mx6    1/1     Running   0          5m4s
$ kubectl scale deployment nginx-deployment --replicas=7
deployment.extensions/nginx-deployment scaled
$ kubectl get pods -n imooc
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-57f49c59d-8dzn4    1/1     Running   0          5m49s
nginx-deployment-57f49c59d-9jvrp    1/1     Running   0          5m49s
nginx-deployment-57f49c59d-l89w2    1/1     Running   0          3s
nginx-deployment-57f49c59d-lddjm    1/1     Running   0          5m47s
nginx-deployment-57f49c59d-m57sr    1/1     Running   0          5m49s
nginx-deployment-57f49c59d-pfpsm    1/1     Running   0          3s
nginx-deployment-57f49c59d-q6mx6    1/1     Running   0          5m48s
```

下面是缩容操作。

```
$ kubectl scale deployment nginx-deployment -n imooc --replicas=3
deployment.extensions/nginx-deployment scaled
$ kubectl get pods -n imooc
NAME                                READY   STATUS        RESTARTS   AGE
nginx-deployment-57f49c59d-8dzn4    1/1     Running       0          6m52s
nginx-deployment-57f49c59d-9jvrp    1/1     Running       0          6m52s
nginx-deployment-57f49c59d-l89w2    0/1     Terminating   0          66s
nginx-deployment-57f49c59d-m57sr    1/1     Running       0          6m52s
nginx-deployment-57f49c59d-pfpsm    0/1     Terminating   0          66s
nginx-deployment-57f49c59d-q6mx6    0/1     Terminating   0          6m51s
```

如果设置了 水平自动缩放 Pod，则可以通过 `kubectl autoscale` 来根据 cpu 使用率来进行自动缩放。

```
kubectl autoscale deployment nginx-deployment --min=10 --max=15 --cpu-percent=80
```

6. 总结

本文介绍了 Deployment 的适用场景和典型使用 case，包括：创建、更新、回滚、缩放，希望大家可以自动动手操作起来。

}

36 Kubernetes
ReplicationController 和
ReplicaSet 介绍