**加我微信可进群学习交流：**

**微信号：**

luckylucky421302

也可通过扫描下面二维码添加



**课程更新的知识点会通过微信公众号免费分享给大家，可以关注我的公众号**

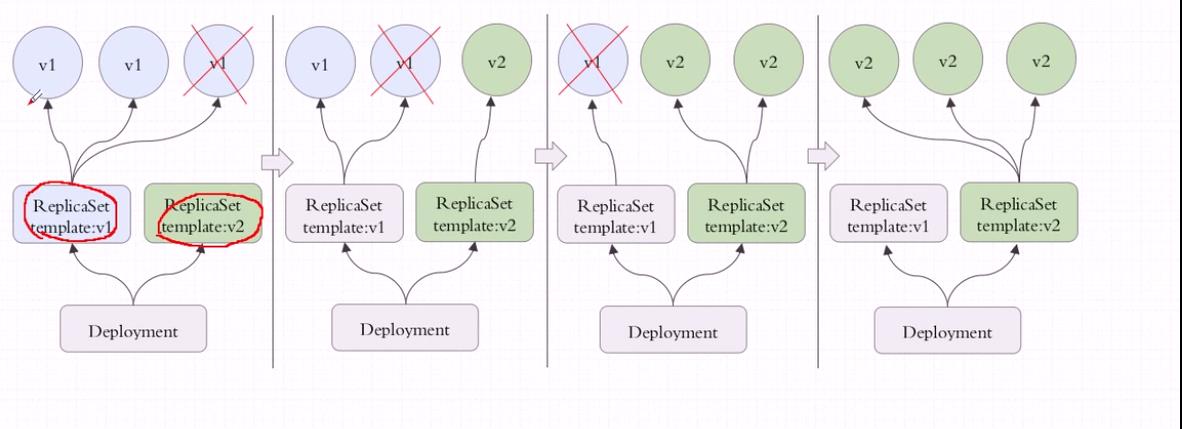
****

pod是有生命周期的，我们可以创建一个自主式pod，也可以创建一个控制器管理的pod，用来管理pod的整个生命周期

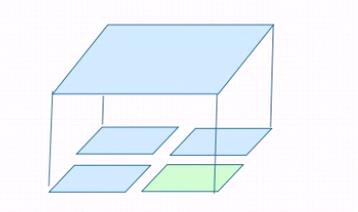
**1.Deployment概念**

Deployment控制器为 Pod 和 ReplicaSet 提供了一个声明式更新的方法，在Deployment对象中描述一个期望的状态，Deployment控制器就会按照一定的控制速率把实际状态改成期望状态，通过定义一个Deployment控制器会创建一个新的ReplicaSets控制器，通过replicaset创建pod，删除Deployment控制器，也会删除Deployment控制器下对应的ReplicaSet控制器和pod资源

Deployment可以用来管理上面说到的那种蓝绿发布情况的，建立在rs之上的，一个Deployment可以管理多个rs，有多个rs存在，但实际运行的只有一个，当你更新到一个新版本的时候，只是创建了一个新的rs，把旧的rs替换掉了



rs的v1控制三个pod，删除一个，在rs的v2上重新建立一个，依次类推，直到全部都是由rs2控制，如果rs v2有问题，还可以回滚，Deployment是建构在rs之上的，多个rs组成一个Deployment，但是只有一个rs处于活跃状态



上面是Deployment,下面是rs，绿色的表示处于活跃状态的rs，一般只保留历史10个版本;

Deployment可以使用声明式定义，直接在命令行通过纯命令的方式完成对应资源版本的内容的修改，也就是通过打补丁的方式进行修改；Deployment能提供滚动式自定义自控制的更新；对Deployment来讲，我们在实现更新时还可以实现控制更新节奏和更新逻辑，什么叫做更新节奏和更新逻辑呢？

比如说ReplicaSet控制5个pod副本，pod的期望值是5个，但是升级的时候需要额外多几个pod，那么我们控制器可以控制在5个pod副本之外还能再增加几个pod副本；比方说能多一个，但是不能少，那么升级的时候就是先增加一个，再删除一个，增加一个删除一个，始终保持pod副本数是5个，但是有个别交叉之间是6个；还有一种情况，最多允许多一个，最少允许少一个，也就是最多6个，最少4个，第一次加一个，删除两个，第二次加两个，删除两个，依次类推，可以自己控制更新方式，这种是滚动更新的，需要加readinessProbe和livenessProbe探测，确保pod中容器里的应用都正常启动了才删除之前的pod；启动的第一步，刚更新第一批就暂停了也可以；假如目标是5个，允许一个也不能少，允许最多可以10个，那一次加5个即可；这就是我们可以自己控制节奏来控制更新的方法

**典型的使用场景：**

1)创建无状态的应用

2)滚动更新和回滚

**2.使用Deployment运行一个无状态应用**

（1）创建一个nginx deployment

（2）使用kubectl列举关于deployment信息

（3）更新deployment

**3.创建一个nginx deployment**

通过创建一个Kubernetes Deployment对象来运行一个应用, 可以在一个YAML文件中描述Deployment. 例如, 下面这个YAML文件描述了一个运行nginx:1.9.1 Docker镜像的Deployment:

cat deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 3 # tells deployment to run 2 pods matching the template

template: # create pods using pod definition in this template

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.9.1

ports:

- containerPort: 80

kubectl apply -f deployment.yaml

**在该示例中：**

1.将创建名为 nginx-deployment 的Deployment ，由 .metadata.name 字段定义。

2.Deployment创建有三个副本的Pods，由 replicas 字段指定。

3.selector 字段定义Deployment如何查找要管理的Pods。在这种情况下，只需选择在 Pod模板（app: nginx）中定义的标签。

**template 字段包含以下子字段：**

使用labels字段将pod 的标签设置成app: nginx

 .template.spec 定义pod要运行的容器，template.spec.name指定名字是nginx，template.spec.image指定运行的镜像是nginx:1.9.1 。

**kubectl get deployment**

显示如下

**NAME READY UP-TO-DATE AVAILABLE AGE**

**nginx-deployment 3/3 3 3 46m**

1.NAME 列出名称空间中deployment的名称。

2.READY显示deployment有多少副本数。它遵循ready/desired的模式。

3.UP-TO-DATE 显示已更新到所需状态的副本数。

4.AVAILABLE 显示你的可以使用多少个应用程序副本。

5.AGE 显示应用程序已运行的时间。

请注意，根据.spec.replicas字段，所需的副本数是3 。

**kubectl get rs**

显示如下

**NAME DESIRED CURRENT READY AGE**

**nginx-deployment-678645bf77 3 3 3 98m**

1.NAME 列出名称空间中ReplicaSet的名称。

2DESIRED显示应用程序的所需副本数，这些副本数是在创建时定义的。这是所需的状态。

3.CURRENT 显示当前正在运行多少个副本。

4.READY 显示你的用户可以使用多少个应用程序副本。

5.AGE 显示应用程序已运行的时间。

请注意，ReplicaSet的名称始终设置为[DEPLOYMENT-NAME]-[RANDOM-STRING]。RANDOM-STRING是随机生成的

**1）展示刚才创建的deployment详细信息**

kubectl describe deployment nginx-deployment

**2）列出deployment创建的拥有标签是app=nginx的pods**

kubectl get pods -l app=nginx

**3）展示某一个pod的信息**

kubectl describe pods <POD-NAME>

**4）更新deployment**

你可以通过更新一个新的YAML文件来更新deployment. 下面的YAML文件指定该deployment镜像更新为nginx 1.8

cat deployment-1.yaml

修改成为如下所示：

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 2

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.8 # Update the version of nginx from 1.7.9 to 1.8

ports:

- containerPort: 80

kubectl apply -f deployment-1.yaml

**5）查看该deployment创建的pods，可发现创建新的pod的同时删除了旧的pods:**

kubectl get pods -l app=nginx

**6）通过增加副本数来扩容pod应用**

你可以通过应用新的YAML文件来增加Deployment中pods的数量. 该YAML文件将replicas设置为4, 指定该Deployment应有4个pods:

cat deployment-1.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 4

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.8 # Update the version of nginx from 1.7.9 to 1.8

ports:

- containerPort: 80

kubectl apply -f deployment-1.yaml

**7）验证Deployment是否有4个pods:**

kubectl get pods -l app=nginx

显示如下：

nginx-deployment-5f8c6846ff-2lhm9 1/1 Running 0 9s

nginx-deployment-5f8c6846ff-95qdx 1/1 Running 0 9s

nginx-deployment-5f8c6846ff-bqht4 1/1 Running 0 2m16s

nginx-deployment-5f8c6846ff-h4ndt 1/1 Running 0 2m18s

**8）查看历史版本**

kubectl rollout history deployment nginx-deployment

显示如下：

deployment.apps/nginx-deployment

REVISION CHANGE-CAUSE

1 <none>

2 <none>

**9）按照指定版本回滚**

kubectl rollout undo deployment/nginx-deployment --to-revision=1

**10）通过Deployment管理replicaset**

创建deployment的时候会创建一个replicaset，replicaset会在后台创建pod

kubectl get rs

#可看到创建的replicaset有哪些

**11）删除deployment**

kubectl delete deployment nginx-deployment

**4.通过yaml编写deployment的规范**

和kubernetes中的其他资源对象一样，创建deployment的yaml文件需要apiversion、kind、metadata字段。

apiVersion: apps/v1

kind: Deployment

metadata:

name: nginx-deployment

spec:

selector:

matchLabels:

app: nginx

replicas: 3 # tells deployment to run 2 pods matching the template

template: # create pods using pod definition in this template

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx:1.9.1

ports:

- containerPort: 80

**5.写一个deployment的spec**

**Pod template**

.spec.template和.spec.selector是.spec下的必须字段，.spec.template是[Pod模板](https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/" \l "pod-templates)。它具有与[Pod](https://kubernetes.io/docs/concepts/workloads/pods/pod/)完全相同的架构，只是它是嵌套的并且没有apiVersion和kind。除了Pod的必填字段外，部署中的Pod模板还必须指定适当的标签和适当的重新启动策略。对于标签，请确保不要与其他控制器重叠。[.spec.template.spec.restartPolicy](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#restart-policy)默认是Always。

**replicas**

.spec.replicas是一个可选字段，用于指定所需Pod的数量。默认为1

**selector**

.spec.selector是指定本次Deployment Pods[标签选择器](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/)的必要字段。

.spec.selector必须匹配 .spec.template.metadata.labels，否则请求会被 API 拒绝。在 API apps/v1版本中，.spec.selector 和 .metadata.labels 不会被默认设置为 .spec.template.metadata.labels，如果没有设置的话。所以需要明确进行设置。同时在 apps/v1版本中， Deployment 创建后 .spec.selector 是可变的。当 Pods 的标签和选择器匹配时，此类 Pods 的模板和 .spec.template 不同，或者此类 Pods 的总数超过 .spec.replicas， Deployment 会终结这些 Pods。如果 Pods 总数达不到期望值，会用 .spec.template 创建新的 Pods。

**注意：**

不应直接通过创建另一个Deployment或创建另一个控制器（例如ReplicaSet或ReplicationController）来创建其标签与该选择器匹配的其他Pod。如果这样做，则第一个Deployment会认为它创建了其他Pod。如果你有多个具有重叠选择器的控制器，则这些控制器将相互竞争，并且无法正常运行

**strategy**

.spec.strategy策略指定用新Pods替换旧Pods的策略。.spec.strategy.type可以是“Recreate”或“RollingUpdate”。“RollingUpdate”是默认值。

**Recreate Deployment**

当.spec.strategy.type=Recreate,所有现有的 Pods 在创建新 Pods 之前被杀死。

**滚动更新 Deployment**

Deployment 会在 .spec.strategy.type==RollingUpdate时，采取 [滚动更新](https://kubernetes.io/docs/tasks/run-application/rolling-update-replication-controller/)的方式更新Pods。可以指定maxUnavailable和maxSurge来控制滚动更新操作。

**Max Unavailable**

spec.strategy.rollingUpdate.maxUnavailable是一个可选字段，用于指定在更新过程中不可用的pod的最大数量，该值可以是绝对数或者所需pod的百分比（例如，10%），绝对数按照四舍五入的百分比计算，如果.spec.strategy.rollingUpdate.maxSurge为0，则该值不能为0。默认值为25％。例如，当此值设置为30％时，在滚动更新开始时，立即将旧的ReplicaSet缩小为所需Pod的70％。一旦准备好新的Pod，就可以进一步缩小旧的ReplicaSet的大小，然后按比例放大新的ReplicaSet，以确保更新期间始终可用的Pod总数至少为所需Pod的70％。

**Max Surge**

.spec.strategy.rollingUpdate.maxSurge是一个可选字段，用于指定可以在所需数量的Pod上创建的最大Pod数量。该值可以是绝对数（例如5）或所需Pod的百分比（例如10％）。如果MaxUnavailable为0，则该值不能为0。绝对值是通过四舍五入从百分比中得出的。默认值为25％。

例如，当此值设置为30％时，可以在滚动更新开始时立即按比例放大新的ReplicaSet，以使旧Pod和新Pod的总数不超过所需Pod的130％。一旦旧Pod被杀死，新的ReplicaSet便可以进一步扩展，以确保更新期间随时运行的Pod总数最多为所需Pod的130％。

**Progress Deadline Seconds**

.spec.progressDeadlineSeconds是一个可选字段，它指定在系统报告部署[失败](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/" \l "failed-deployment)之前你要等待部署进行的秒数,以Type=Progressing， Status=False以及Reason=ProgressDeadlineExceeded资源的状态为例。deployment控制器将继续重试部署。默认值为600。将来，一旦实现自动回滚，Deployment控制器将在观察到这种情况后立即回滚Deployment。如果指定，则此字段必须大于.spec.minReadySeconds。

**Min Ready Seconds**

.spec.minReadySeconds是一个可选字段，用于指定新创建的Pod在不使其任何容器崩溃的情况下应准备就绪的最小秒数，以便将其视为可用。默认值为0（准备就绪后，Pod将被视为可用）。要了解有关何时将Pod准备就绪的更多信息，可参考

<https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes>

**Revision History Limit**

deployment的修订历史记录存储在它控制的ReplicaSets中。

.spec.revisionHistoryLimit是一个可选字段，用于指定要保留的可用于回滚的旧的replicaset的数量。这些旧的ReplicaSets在etcd中消耗资源，将kubectl get rs的输出结果聚集起来。每个Deployment版本的配置都存储在ReplicaSets中。因此，一旦删除了旧的ReplicaSet，你将无法回滚到该版本的Deployment。默认情况下，将保留10个旧的ReplicaSet，但是其理想值取决于新部署的频率和稳定性。

更具体地说，将此字段设置为零意味着将清除所有具有0个副本的旧ReplicaSet。在这种情况下，由于已清除其修订历史记录，新的部署将不能被撤销。

**Paused**

.spec.paused是一个可选的布尔字段，用于暂停和恢复部署。暂停的Deployment和未暂停的Deployment之间的唯一区别是，只要暂停，对暂停的Deployment的PodTemplateSpec所做的任何更改都不会触发新的rollouts。当创建时，默认情况下不会暂停deployment。