**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**

****

**Statefulset概念**

StatefulSet是为了解决有状态服务的问题（对应Deployments和ReplicaSets是为无状态服务）而设计，其应用场景包括

1.稳定的持久化存储，即Pod重新调度后还是能访问到相同的持久化数据，基于PVC来实现

2.稳定的网络标志，即Pod重新调度后其PodName和HostName不变，基于Headless Service（即没有Cluster IP的Service）来实现

3.有序部署，有序扩展，即Pod是有顺序的，在部署或者扩展的时候要依据定义的顺序依次进行（即从0到N-1，在下一个Pod运行之前的所有Pod必须都是Running和Ready状态），基于init containers来实现

4.有序收缩，有序删除（即从N-1到0）

从上面的应用场景可以发现，**StatefulSet由以下几个部分组成：**

1.通过Headless Service生成可解析的DNS记录

2.通过volumeClaimTemplates创建pvc和对应的pv绑定

3.定义StatefulSet来创建pod

和[Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/" \t "_blank)相同的是，StatefulSet管理了基于相同容器定义的一组 Pod。但和 Deployment不同的是，StatefulSet 为它们的每个 Pod 维护了一个固定的ID。这些 Pod 是基于相同的声明来创建的，但是不能相互替换：无论怎么调度，每个 Pod 都有一个永久不变的ID。StatefulSet和其他控制器使用相同的工作模式。你在StatefulSet*对象*中定义你期望的状态，然后StatefulSet的*控制器*就会通过各种更新来达到那种你想要的状态。

StatefulSet中每个Pod的DNS格式为statefulSetName-{0..N-1}.serviceName.namespace.svc.cluster.local，其中

1.serviceName为Headless Service的名字

2.0..N-1为Pod所在的序号，从0开始到N-1

3.statefulSetName为StatefulSet的名字

4.namespace为服务所在的namespace，Headless Servic和StatefulSet必须在相同的namespace

5..svc.cluster.local为Cluster Domain

**学习达到的目标**

1.如何创建StatefulSet？

2.通过StatefulSet怎样管理它的Pod？

3.如何删除 StatefulSet？

4.如何对StatefulSet进行扩容/缩容？

5.如何更新一个 StatefulSet的Pod

**部署一个web应用**

作为开始，使用如下示例创建一个StatefulSet。它和[StatefulSets](https://kubernetes.io/docs/concepts/abstractions/controllers/statefulsets/)概念中的示例相似。它创建了一个[Headless Service](https://kubernetes.io/docs/user-guide/services/" \l "headless-services)的nginx用来发布 StatefulSet web中的Pod的IP地址。

cat statefulset.yaml

apiVersion: v1

kind: Service

metadata:

name: nginx

labels:

app: nginx

spec:

ports:

- port: 80

name: web

clusterIP: None

selector:

app: nginx

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: web

spec:

selector:

matchLabels:

app: nginx

serviceName: "nginx"

replicas: 2

template:

metadata:

labels:

app: nginx

spec:

containers:

- name: nginx

image: nginx

ports:

- containerPort: 80

name: web

volumeMounts:

- name: www

mountPath: /usr/share/nginx/html

volumeClaimTemplates:

- metadata:

name: www

spec:

accessModes: ["ReadWriteOnce"]

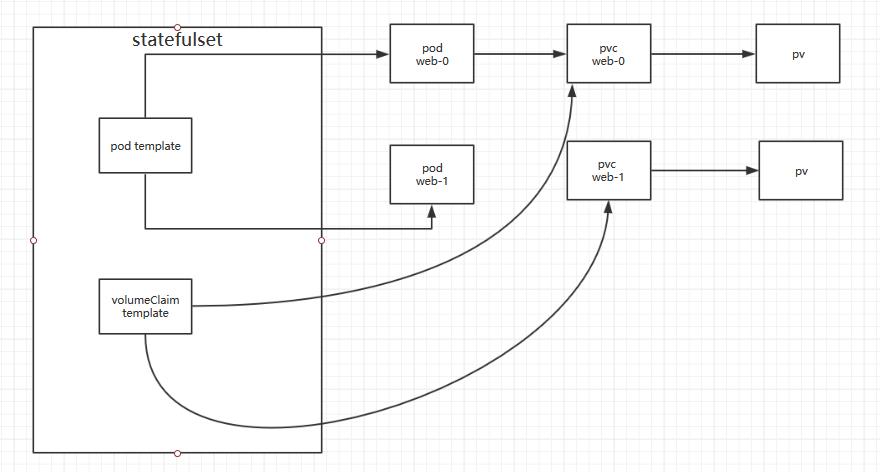
resources:

requests:

storage: 1Gi

**注：为什么要使用volumeClaimTemplate**

对于有状态应用都会用到持久化存储，比如mysql主从，由于主从数据库的数据是不能存放在一个目录下的，每个mysql节点都需要有自己独立的存储空间。而在deployment中创建的存储卷是一个共享的存储卷，多个pod使用同一个存储卷，它们数据是同步的，而statefulset定义中的每一个pod都不能使用同一个存储卷，这就需要使用volumeClainTemplate，当在使用statefulset创建pod时，会自动生成一个PVC，从而请求绑定一个PV，每一个pod都有自己专用的存储卷。Pod、PVC和PV对应的关系图如下：



**通过statefulset部署pod，并且观察pod创建的过程**

你需要使用两个终端窗口。在第一个终端中，使用[kubectl get](https://kubernetes.io/docs/user-guide/kubectl/v1.15/" \l "get)来查看 StatefulSet的Pods的创建情况。

kubectl get pods -w -l app=nginx

在另一个终端中，使用 kubectl apply -f statefulset.yaml来创建定义在statefulset.yaml中的 Headless Service 和StatefulSet。

kubectl apply -f statefulset.yaml

上面的命令创建了两个 Pod，每个都运行了一个[NGINX](https://www.nginx.com/) web 服务器。获取nginx Service 和web StatefulSet来验证是否成功的创建了它们。

**查看service**

kubectl get service nginx

NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE

nginx None <none> 80/TCP 12s

**查看statefulset**

kubectl get statefulset web

显示如下：

NAME READY AGE

web 2/2 20m

**顺序创建 Pod**

对于一个拥有 N 个副本的 StatefulSet，Pod 被部署时是按照 {0..N-1}的序号顺序创建的。在第一个终端中使用 kubectl get 检查输出。这个输出最终将看起来像下面的样子。

kubectl get pods -w -l app=nginx

**NAME READY STATUS RESTARTS AGE**

**web-0 0/1 Pending 0 0s**

**web-0 0/1 Pending 0 0s**

**web-0 0/1 ContainerCreating 0 0s**

**web-0 1/1 Running 0 19s**

**web-1 0/1 Pending 0 0s**

**web-1 0/1 Pending 0 0s**

**web-1 0/1 ContainerCreating 0 0s**

**web-1 1/1 Running 0 18s**

请注意在web-0 Pod处于[Running和Ready](https://kubernetes.io/docs/user-guide/pod-states) 状态后web-1 Pod才会被启动。

**StatefulSet 中的 Pod**

StatefulSet 中的 Pod 拥有一个唯一的顺序索引和稳定的网络身份标识

**检查 Pod 的顺序索引**

获取 StatefulSet 的 Pod。

kubectl get pods -l app=nginx

NAME READY STATUS RESTARTS AGE

web-0 1/1 Running 0 1m

web-1 1/1 Running 0 1m

如同[StatefulSets](https://kubernetes.io/docs/concepts/abstractions/controllers/statefulsets/) 概念中所提到的， StatefulSet中的Pod拥有一个具有黏性的、独一无二的身份标志。这个标志基于StatefulSet控制器分配给每个 Pod的唯一顺序索引。 Pod的名称的形式为<statefulset name>-<ordinal index>。web StatefulSet 拥有两个副本，所以它创建了两个 Pod：web-0 和 web-1

**使用稳定的网络身份标识**

每个Pod都拥有一个基于其顺序索引的稳定的主机名（statefulset创建的pod的主机名由statefulset的名称和有序索引组成）。使用[kubectl exec](https://kubernetes.io/docs/user-guide/kubectl/v1.15/" \l "exec) 在每个 Pod 中执行hostname

**for i in 0 1; do kubectl exec web-$i -- sh -c 'hostname'; done**

**显示如下：**

**web-0**

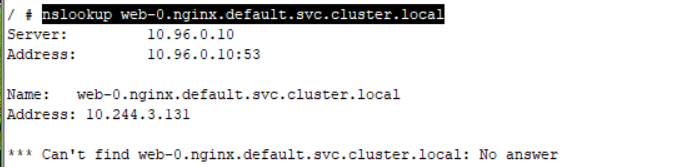
**web-1**

使用[kubectl run](https://kubernetes.io/docs/user-guide/kubectl/v1.15/" \l "run)运行一个提供nslookup命令的容器，该命令来自于dnsutils包。通过对Pod的主机名执行nslookup，你可以检查他们在集群内部的DNS地址。

**kubectl exec -it web-1 -- /bin/bash**

**apt-get install dnsutils -y**

nslookup web-0.nginx.default.svc.cluster.local



在一个终端中查看 StatefulSet 的 Pod。

**kubectl get pod -w -l app=nginx**

在另一个终端中使用 **[kubectl delete](https://kubernetes.io/docs/user-guide/kubectl/v1.15/" \l "delete)** 删除 StatefulSet 中所有的 Pod。

**kubectl delete pod -l app=nginx**

**pod "web-0" deleted**

**pod "web-1" deleted**

等待 StatefulSet 重启它们，并且两个 Pod 都变成 Running 和 Ready 状态。

**kubectl get pod -w -l app=nginx**

NAME READY STATUS RESTARTS AGE

web-0 0/1 ContainerCreating 0 0s

NAME READY STATUS RESTARTS AGE

web-0 1/1 Running 0 2s

web-1 0/1 Pending 0 0s

web-1 0/1 Pending 0 0s

web-1 0/1 ContainerCreating 0 0s

web-1 1/1 Running 0 34s

使用 **kubectl exec** 和 **kubectl run** 查看 Pod 的主机名和集群内部的 DNS 表项

**for i in 0 1; do kubectl exec web-$i -- sh -c 'hostname'; done**

web-0

web-1

web-2

kubectl exec -it web-1 -- /bin/bash

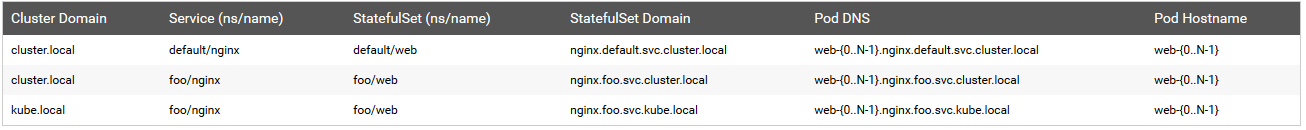
apt-get install dnsutils -y

#安装nslookup命令的的

Nslookup web-0.nginx.default.svc.cluster.local

Pod 的序号、主机名、SRV 条目和记录名称没有改变，但和 Pod 相关联的 IP 地址可能发生了改变。这就是为什么不要在其他应用中使用 StatefulSet 中的 Pod的IP 地址进行连接，这点很重要。如果你需要查找并连接一个 StatefulSet的活动成员，你应该查询Headless Service 的CNAME。CNAME 相关联的 SRV 记录只会包含 StatefulSet 中处于 Running 和 Ready 状态的 Pod。如果你的应用已经实现了用于测试 liveness 和 readiness 的连接逻辑，你可以使用 Pod 的 SRV 记录（web-0.nginx.default.svc.cluster.local, web-1.nginx.default.svc.cluster.local）。因为他们是稳定的，并且当你的 Pod 的状态变为 Running 和 Ready 时，你的应用就能够发现它们的地址。

域名对应的表格：



**导入稳定的存储**

获取**web-0** 和 **web-1** 的 PersistentVolumeClaims

**kubectl get pvc -l app=nginx**

NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE

www-web-0 Bound v1 1Gi RWO 50m

www-web-1 Bound v4 2Gi RWO,RWX 50m

www-web-2 Bound v5 5Gi RWO,RWX 50m

StatefulSet 控制器创建了三个 PersistentVolumeClaims，绑定到两三个[PersistentVolumes](https://kubernetes.io/docs/concepts/storage/volumes/)。

NGINX web 服务器默认会加载位于 **/usr/share/nginx/html/index.html** 的 index 文件。StatefulSets **spec** 中的 **volumeMounts** 字段保证了 **/usr/share/nginx/html** 文件夹由一个 PersistentVolume 支持。

将Pod的主机名写入它们的**index.html**文件并验证 NGINX web 服务器使用该主机名提供服务。

for i in 0 1; do kubectl exec web-$i -- sh -c 'echo $(hostname) > /usr/share/nginx/html/index.html'; done

for i in 0 1; do kubectl exec -it web-$i -- curl localhost; done

web-0

web-1

请注意，如果你看见上面的 curl 命令返回了 403 Forbidden 的响应，你需要像这样修复使用 **volumeMounts** (due to a [bug when using hostPath volumes](https://github.com/kubernetes/kubernetes/issues/2630))挂载的目录的权限：

**for i in 0 1 2; do kubectl exec web-$i -- chmod 755 /usr/share/nginx/html; done**

在你重新尝试上面的 curl 命令之前。

**部署和扩缩容**

1.对于包含N个副本的StatefulSet，当部署Pod时，它们是依次创建的，顺序为 0..N-1。

2.当删除 Pod 时，它们是逆序终止的，顺序为 N-1..0。

3.在将缩放操作应用到Pod之前，它前面的所有Pod 必须是 Running 和 Ready 状态。

4.在 Pod 终止之前，所有的继任者必须完全关闭。

StatefulSet 不应将 pod.Spec.TerminationGracePeriodSeconds 设置为 0。这种做法是不安全的，要强烈阻止。更多的解释请参考 [强制删除 StatefulSet Pod](https://kubernetes.io/docs/tasks/run-application/force-delete-stateful-set-pod/)。

在上面的 nginx 示例被创建后，会按照 web-0、web-1的顺序部署Pod。在 web-0 进入 [Running 和 Ready](https://kubernetes.io/docs/user-guide/pod-states/) 状态前不会部署 web-1。在 web-1 进入 Running 和 Ready 状态前不会部署 web-2。如果 web-1 已经处于 Running 和 Ready 状态，而 web-2 尚未部署，在此期间发生了 web-0 运行失败，那么 web-2 将不会被部署，要等到 web-0 部署完成并进入 Running 和 Ready 状态后，才会部署 web-2。如果用户想将示例中的 StatefulSet 收缩为 replicas=1，首先被终止的是 web-2。在 web-2 没有被完全停止和删除前，web-1 不会被终止。当 web-2 已被终止和删除、web-1 尚未被终止，如果在此期间发生 web-0 运行失败，那么就不会终止 web-1，必须等到 web-0 进入 Running 和 Ready 状态后才会终止 web-1。

**Statefulset的资源清单yaml文件书写规范**

**1.Pod selector**

你必须设置StatefulSet的.spec.selector字段，使之匹配其在.spec.template.metadata.labels中设置的标签。在Kubernetes 1.8版本之前，被忽略.spec.selector字段会获得默认设置值。在 1.8和以后的版本中，未指定匹配的Pod 选择器将在创建 StatefulSet 期间导致验证错误。

**2.Pod Identity（pod标识）**

StatefulSet Pod 具有唯一的标识，该标识包括顺序标识、稳定的网络标识和稳定的存储。该标识和Pod是绑定的，不管它被调度在哪个节点上。

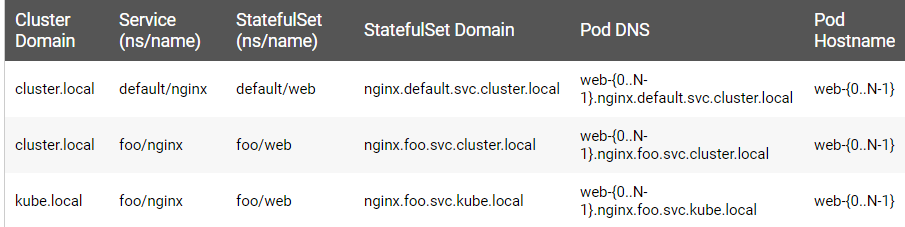
**3.有序索引**

对于具有N个副本的StatefulSet，StatefulSet 中的每个 Pod 将被分配一个整数序号，从 0 到 N-1，该序号在StatefulSet 上是唯一的。

**4.稳定的网络ID**

### 1）StatefulSet 中的每个Pod根据StatefulSet的名称和Pod的序号派生出它的主机名。组合主机名的格式为$(StatefulSet 名称)-$(序号)。上例将会创建名称分别为 web-0、web-1的Pod。 StatefulSet 可以使用[headless 服务](https://kubernetes.io/docs/concepts/services-networking/service/" \l "headless-services)控制它的Pod的网络域。管理域的这个服务的格式为： $(服务名称).$(命名空间).svc.cluster.local，其中cluster.local是集群域。 一旦每个 Pod 创建成功，就会得到一个匹配的 DNS 子域，格式为：$(pod 名称).$(所属服务的 DNS 域名)，其中所属服务由 StatefulSet的serviceName域来设定。

### 2）下面给出一些选择集群域、服务名、StatefulSet 名、及其怎样影响 StatefulSet 的 Pod 上的 DNS 名称的示例：



**注意：** 集群域会被设置为 cluster.local，除非有[其他配置](https://kubernetes.io/docs/concepts/services-networking/dns-pod-service/" \l "how-it-works)。

**5.稳定的存储**

### 1）Kubernetes 为每个 VolumeClaimTemplate 创建一个 [PersistentVolume](https://kubernetes.io/docs/concepts/storage/persistent-volumes/)。在上面的 nginx 示例中，每个 Pod 将会得到基于 StorageClass my-storage-class 提供的 1 Gib 的 PersistentVolume。如果没有声明 StorageClass，就会使用默认的 StorageClass。当一个 Pod 被调度（重新调度）到节点上时，它的 volumeMounts 会挂载与其 PersistentVolumeClaims 相关联的 PersistentVolume。请注意，当 Pod 或者 StatefulSet 被删除时，与 PersistentVolumeClaims 相关联的 PersistentVolume 并不会被删除。要删除它必须通过手动方式来完成。

### 2）Pod 名称标签

### 3）当StatefulSet[控制器](https://kubernetes.io/docs/admin/kube-controller-manager/" \t "_blank)创建 Pod 时，它会添加一个标签statefulset.kubernetes.io/pod-name，该标签设置为Pod名称。这个标签允许您给 StatefulSet 中的特定 Pod 绑定一个 Service。

**下面的内容自行学习查看即可**

**Pod 管理策略**

在 Kubernetes 1.7 及以后的版本中，StatefulSet 允许你不要求其排序保证，同时通过它的 .spec.podManagementPolicy 域保持其唯一性和身份保证。 在 Kubernetes 1.7 及以后的版本中，StatefulSet 允许您放宽其排序保证，同时通过它的 .spec.podManagementPolicy 域保持其唯一性和身份保证。

**OrderedReady Pod 管理**

OrderedReady Pod 管理是 StatefulSet 的默认设置。它实现了[上面](https://kubernetes.io/zh/docs/concepts/workloads/controllers/statefulset/" \l "deployment-and-scaling-guarantees)描述的功能。

**Parallel Pod 管理**

Parallel Pod 管理让 StatefulSet 控制器并行的启动或终止所有的 Pod，启动或者终止其他 Pod 前，无需等待 Pod 进入 Running 和 ready 或者完全停止状态。

**更新策略**

在Kubernetes 1.7及以后的版本中，StatefulSet 的.spec.updateStrategy 字段让你可以配置和禁用掉自动滚动更新Pod的容器、标签、资源请求或限制、以及注解。

**关于删除策略**

OnDelete 更新策略实现了 1.6 及以前版本的历史遗留行为。当 StatefulSet 的 .spec.updateStrategy.type 设置为 OnDelete 时，它的控制器将不会自动更新 StatefulSet 中的 Pod。用户必须手动删除Pod以便让控制器创建新的 Pod，以此来对StatefulSet的 .spec.template 的变动作出反应。

**滚动更新**

RollingUpdate 更新策略对 StatefulSet 中的 Pod 执行自动的滚动更新。在没有声明 .spec.updateStrategy 时，RollingUpdate 是默认配置。 当 StatefulSet 的 .spec.updateStrategy.type 被设置为 RollingUpdate 时，StatefulSet 控制器会删除和重建 StatefulSet 中的每个 Pod。 它将按照与 Pod 终止相同的顺序（从最大序号到最小序号）进行，每次更新一个 Pod。它会等到被更新的 Pod 进入 Running 和 Ready 状态，然后再更新其前身。

**分区**

通过声明 .spec.updateStrategy.rollingUpdate.partition 的方式，RollingUpdate 更新策略可以实现分区。如果声明了一个分区，当 StatefulSet 的 .spec.template 被更新时，所有序号大于等于该分区序号的 Pod 都会被更新。所有序号小于该分区序号的 Pod 都不会被更新，并且，即使他们被删除也会依据之前的版本进行重建。如果 StatefulSet 的 .spec.updateStrategy.rollingUpdate.partition 大于它的 .spec.replicas，对它的 .spec.template 的更新将不会传递到它的 Pod。 在大多数情况下，您不需要使用分区，但如果您希望进行阶段更新、执行金丝雀或执行分阶段展开，则这些分区会非常有用。

#### 强制回滚

在默认 [Pod 管理策略](https://kubernetes.io/zh/docs/concepts/workloads/controllers/statefulset/#pod-management-policies)(OrderedReady) 时使用 [滚动更新](https://kubernetes.io/zh/docs/concepts/workloads/controllers/statefulset/#rolling-updates) ，可能进入需要人工干预才能修复的损坏状态。

如果更新后 Pod 模板配置进入无法运行或就绪的状态（例如，由于错误的二进制文件或应用程序级配置错误），StatefulSet 将停止回滚并等待。

在这种状态下，仅将 Pod 模板还原为正确的配置是不够的。由于[已知问题](https://github.com/kubernetes/kubernetes/issues/67250)，StatefulSet 将继续等待损坏状态的 Pod 准备就绪（永远不会发生），然后再尝试将其恢复为正常工作配置。

恢复模板后，还必须删除 StatefulSet 尝试使用错误的配置来运行的 Pod。这样，StatefulSet 才会开始使用被还原的模板来重新创建 Pod。