k8s基础篇-资源调度

Deployment

什么是Deployment

用于部署无状态的服务，这个最常用的控制器。一般用于管理维护企业内部无状态的微服务，比如configserver、zuul、springboot。他可以管理多个副本的Pod实现无缝迁移、自动扩容缩容、自动灾难恢复、一键回滚等功能。

用于部署无状态的服务!!!

创建一个Deployment

手动创建

```
[root@k8s-master01 ~]# kubectl create deployment nginx --image=nginx:1.15.2
deployment.apps/nginx created
```

使用文件创建

```
# 查看手动创建的nginx的yaml文件,然后把f开头的删了,且删了最后的status标签的内容，得到下面的yaml文件
[root@k8s-master01 ~]# kubectl  get deployment nginx -o yaml > nginx-deploy.yaml
```

```
cat > nginx-deploy.yaml << EFO
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2020-12-22T00:07:49Z"
  generation: 1
  labels:
    app: nginx
  name: nginx
  namespace: default
  resourceVersion: "73782"
  uid: 6186f4c7-50bc-45d0-9ed4-916b311802eb
spec:
```

```
  progressDeadlineSeconds: 600
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:1.15.2
        imagePullPolicy: IfNotPresent
        name: nginx
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
EFO
```

> replicas: 副本数
>
> revisionHistoryLimit: 10 历史记录保留的个数

```
# 使用以下命令去新建一个deployment
[root@k8s-master01 ~]# kubectl replace -f nginx-deploy.yaml
deployment.apps/nginx replaced

# 在线更改yaml，管理deployment    ---把副本数改为2
[root@k8s-master01 ~]# kubectl edit deploy  nginx

# 查看是否生成2个副本
[root@k8s-master01 ~]# kubectl get po
NAME                     READY   STATUS              RESTARTS   AGE
nginx-66bbc9fdc5-c6l6t   1/1     Running             0          57s
nginx-66bbc9fdc5-hsv4d   1/1     Running           0          34m
```

状态解析

```
[root@k8s-master01 ~]# kubectl get deploy -owide
NAME    READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES         SELECTOR
nginx   2/2     2            1           35m   nginx        nginx:1.15.2   app=nginx

 NAME： Deployment名称
 READY：Pod的状态，已经Ready的个数
 UP-TO-DATE：已经达到期望状态的被更新的副本数
 AVAILABLE：已经可以用的副本数
 AGE：显示应用程序运行的时间
 CONTAINERS：容器名称
 IMAGES：容器的镜像
 SELECTOR：管理的Pod的标签
```

Deployment的更新

更改deployment的镜像并记录

```
[root@k8s-master01 ~]# kubectl set image deploy nginx nginx=nginx:1.15.3 --record
deployment.apps/nginx image updated
```

查看更新过程

```
[root@k8s-master01 ~]# kubectl rollout status deploy nginx
Waiting for deployment "nginx" rollout to finish: 1 out of 3 new replicas have been
updated...

# 或者使用describe查看
[root@k8s-master01 ~]# kubectl describe deploy nginx
```

Deployment的回滚

回滚到上一个版本

```
# 例如错误的更新到了一个xxx版本
[root@k8s-master01 ~]# kubectl set image deploy nginx nginx=nginx:xxx --record
deployment.apps/nginx image updated

# 查看kubectl更新的历史命令
[root@k8s-master01 ~]# kubectl rollout history deploy nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
```

```
1         <none>
2         kubectl set image deploy nginx nginx=nginx:1.15.3 --record=true
3         kubectl set image deploy nginx nginx=nginx:xxx --record=true

# 回滚到上一个版本
[root@k8s-master01 ~]# kubectl rollout undo deploy nginx
deployment.apps/nginx rolled back
```

回滚到指定版本

```
# 多次更新错误版本
[root@k8s-master01 ~]# kubectl set image deploy nginx nginx=nginx:aa --record
deployment.apps/nginx image updated
[root@k8s-master01 ~]# kubectl set image deploy nginx nginx=nginx:bb --record
deployment.apps/nginx image updated
[root@k8s-master01 ~]# kubectl set image deploy nginx nginx=nginx:cc --record
deployment.apps/nginx image updated

# 查看kubectl更新的历史命令
[root@k8s-master01 ~]# kubectl rollout history deploy nginx
deployment.apps/nginx
REVISION  CHANGE-CAUSE
1         <none>
3         kubectl set image deploy nginx nginx=nginx:xxx --record=true
4         kubectl set image deploy nginx nginx=nginx:1.15.3 --record=true
5         kubectl set image deploy nginx nginx=nginx:aa --record=true
6         kubectl set image deploy nginx nginx=nginx:bb --record=true
7         kubectl set image deploy nginx nginx=nginx:cc --record=true

# 查看指定版本的详细信息 ---看revision对应的数字即可
[root@k8s-master01 ~]# kubectl rollout history deploy nginx --revision=4
deployment.apps/nginx with revision #4
Pod Template:
  Labels:       app=nginx
        pod-template-hash=5dfc8689c6
  Annotations:  kubernetes.io/change-cause: kubectl set image deploy nginx
nginx=nginx:1.15.3 --record=true
  Containers:
   nginx:
    Image:      nginx:1.15.3
    Port:       <none>
    Host Port:  <none>
    Environment:        <none>
    Mounts:     <none>
  Volumes:      <none>

# 回滚到指定版本
[root@k8s-master01 ~]# kubectl rollout undo deploy nginx --to-revision=4
deployment.apps/nginx rolled back
```

Deployment的扩容与缩容

Deployment的扩容

```
# Deployment的扩容与缩容，不会生成新的rs
[root@k8s-master01 ~]# kubectl  scale  --replicas=4  deploy nginx
deployment.apps/nginx scaled

# --replicas  # 指定副本数
# nginx     # pod的名字

# 查看rs
[root@k8s-master01 ~]# kubectl get rs
```

Deployment的缩容

```
# Deployment的扩容与缩容，不会生成新的rs
[root@k8s-master01 ~]# kubectl  scale  --replicas=1  deploy nginx
deployment.apps/nginx scaled

# --replicas  # 指定副本数
# nginx     # pod的名字

# 查看rs
[root@k8s-master01 ~]# kubectl get rs
```

Deployment的暂停和恢复

- **deployment可以在线edit更改（可以一次性更改多个）**
- **也可以用kubectl set image更改（也可以一次性更改多个，但是需要使用到Deployment的暂停和恢复功能）**

Deployment 暂停功能

```
# 暂停
[root@k8s-master01 ~]# kubectl rollout pause deployment nginx
deployment.apps/nginx paused

# 第一次更新
[root@k8s-master01 ~]# kubectl set image deploy nginx nginx=nginx:1.15.4 --record
deployment.apps/nginx image updated

# 第二次更新、添加内存、CPU
```

```
[root@k8s-master01 ~]# kubectl set resources deploy nginx -c nginx --
limits=cpu=200m,memory=128Mi --requests=cpu=10m,memory=16Mi
deployment.apps/nginx resource requirements updated

# 查看被更改以后的nginx镜像的deployment
[root@k8s-master01 ~]# kubectl get deploy nginx -oyaml
```

Deployment 恢复功能

```
# 更新完想更新的内容后，然后恢复镜像
[root@k8s-master01 ~]# kubectl rollout resume deploy nginx
deployment.apps/nginx resumed

# 查看rs，看到有新的
[root@k8s-master01 ~]# kubectl get rs
NAME                 DESIRED   CURRENT   READY   AGE
nginx-5b6bc78b67     1         1         0       41s
```

Deployment注意事项

```
kubectl get deploy nginx -oyaml
# 查看配置，下面是针对的描述。
```

```
.spec.revisionHistoryLimit：设置保留RS旧的revision的个数，设置为0的话，不保留历史数据

.spec.minReadySeconds：可选参数，指定新创建的Pod在没有任何容器崩溃的情况下视为Ready最小的秒数，默认为0，即一旦被创建就视为可用。

滚动更新的策略：
  .spec.strategy.type：更新deployment的方式，默认是RollingUpdate
    RollingUpdate：滚动更新，可以指定maxSurge和maxUnavailable
      maxUnavailable：指定在回滚或更新时最大不可用的Pod的数量，可选字段，默认25%，可以设置成数字或百分比，如果该值为0，那么maxSurge就不能0
      maxSurge：可以超过期望值的最大Pod数，可选字段，默认为25%，可以设置成数字或百分比，如果该值为0，那么maxUnavailable不能为0
    Recreate：重建，先删除旧的Pod，在创建新的Pod
```

StatefulSet

什么是StatefulSet

StatefulSet（有状态集，缩写为sts）常用于部署有状态的且需要有序启动的应用程序，比如在进行SpringCloud项目容器化时，Eureka的部署是比较适合用StatefulSet部署方式的，可以给每个Eureka实例创建一个唯一且固定的标识符，并且每个Eureka实例无需配置多余的Service，其余Spring Boot应用可以直接通过Eureka的Headless Service即可进行注册

Eureka的statefulset的资源名称是eureka，eureka-0 eureka-1 eureka-2
Service：headless service，没有ClusterIP eureka-svc
Eureka-0.eureka-svc.NAMESPACE_NAME eureka-1.eureka-svc

StatefulSet的基本概念

StatefulSet主要用于管理有状态应用程序的工作负载API对象。比如在生产环境中，可以部署ElasticSearch集群、MongoDB集群或者需要持久化的RabbitMQ集群、Redis集群、Kafka集群和ZooKeeper集群等。

和Deployment类似，一个StatefulSet也同样管理着基于相同容器规范的Pod。不同的是，StatefulSet为每个Pod维护了一个粘性标识。这些Pod是根据相同的规范创建的，但是不可互换，每个Pod都有一个持久的标识符，在重新调度时也会保留，一般格式为StatefulSetName-Number。比如定义一个名字是Redis-Sentinel的StatefulSet，指定创建三个Pod，那么创建出来的Pod名字就为Redis-Sentinel-0、Redis-Sentinel-1、Redis-Sentinel-2。而StatefulSet创建的Pod一般使用Headless Service（无头服务）进行通信，和普通的Service的区别在于Headless Service没有ClusterIP，它使用的是Endpoint进行互相通信，Headless一般的格式为：

statefulSetName-{0..N-1}.serviceName.namespace.svc.cluster.local。

说明：

- serviceName为Headless Service的名字，创建StatefulSet时，必须指定Headless Service名称；
- 0..N-1为Pod所在的序号，从0开始到N-1；
- statefulSetName为StatefulSet的名字；
- namespace为服务所在的命名空间；
- cluster.local为Cluster Domain（集群域）。

假如公司某个项目需要在Kubernetes中部署一个主从模式的Redis，此时使用StatefulSet部署就极为合适，因为StatefulSet启动时，只有当前一个容器完全启动时，后一个容器才会被调度，并且每个容器的标识符是固定的，那么就可以通过标识符来断定当前Pod的角色。

比如用一个名为redis-ms的StatefulSet部署主从架构的Redis，第一个容器启动时，它的标识符为redis-ms-0，并且Pod内主机名也为redis-ms-0，此时就可以根据主机名来判断，当主机名为redis-ms-0的容器作为Redis的主节点，其余从节点，那么Slave连接Master主机配置就可以使用不会更改的Master的Headless Service，此时Redis从节点（Slave）配置文件如下：

```
port 6379
slaveof redis-ms-0.redis-ms.public-service.svc.cluster.local 6379
tcp-backlog 511
timeout 0
tcp-keepalive 0
...
```

其中redis-ms-0.redis-ms.public-service.svc.cluster.local是Redis Master的Headless Service，在同一命名空间下只需要写redis-ms-0.redis-ms即可，后面的public-service.svc.cluster.local可以省略。

StatefulSet注意事项

一般StatefulSet用于有以下一个或者多个需求的应用程序：
- 需要稳定的独一无二的网络标识符
- 需要持久化数据
- 需要有序的、优雅的部署和扩展
- 需要有序的自动滚动更新

如果应用程序不需要任何稳定的标识符或者有序的部署、删除或者扩展，应该使用无状态的控制器部署应用程序，比如Deployment或者ReplicaSet

StatefulSet是Kubernetes 1.9版本之前的beta资源，在1.5版本之前的任何Kubernetes版本都没有

Pod所用的存储必须由PersistentVolume Provisioner（持久化卷配置器）根据请求配置StorageClass，或者由管理员预先配置，当然也可以不配置存储

为了确保数据安全，删除和缩放StatefulSet不会删除与StatefulSet关联的卷，可以手动选择性地删除PVC和PV

StatefulSet目前使用Headless Service（无头服务）负责Pod的网络身份和通信，需要提前创建此服务

删除一个StatefulSet时，不保证对Pod的终止，要在StatefulSet中实现Pod的有序和正常终止，可以在删除之前将StatefulSet的副本缩减为0

定义一个StatefulSet资源文件

定义一个简单的StatefulSet的示例

```
cat > nginx-sts.yaml << EFO
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:
  serviceName: "nginx"
  replicas: 2
  selector:
```

```
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.15.2
        ports:
        - containerPort: 80
          name: web
EFO
# 此示例没有添加存储配置，后面的章节会单独讲解存储相关的知识点
```

创建一个StatefulSet

```
[root@k8s-master01 ~]# kubectl create -f nginx-sts.yaml
service/nginx created
statefulset.apps/web created

# 查看svc信息
[root@k8s-master01 ~]# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   42h
nginx        ClusterIP   None         <none>        80/TCP    24s

# 查看pod信息
[root@k8s-master01 ~]# kubectl get po
NAME                        READY   STATUS              RESTARTS   AGE
nginx-66bbc9fdc5-vc2gh      1/1     Running             0          12h
nginx-v2-644cd9ccc7-b99nf   1/1     Running             0          11h
web-0                       1/1     Running             0          113s
web-1                       0/1     ContainerCreating   0          111s

# 查看sts
[root@k8s-master01 ~]# kubectl get sts
NAME   READY   AGE
web    2/2     2m42s
```

StatefulSet的扩容

```
[root@k8s-master01 ~]# kubectl scale --replicas=3 sts web
statefulset.apps/web scaled

# 查看pod、发现名字是固定增长的
[root@k8s-master01 ~]# kubectl get pod
NAME                          READY   STATUS              RESTARTS   AGE
nginx-66bbc9fdc5-vc2gh        1/1     Running             0          12h
nginx-v2-644cd9ccc7-b99nf     1/1     Running             0          12h
web-0                         1/1     Running             0          4m48s
web-1                         1/1     Running             0          4m46s
web-2                         0/1     ContainerCreating   0          28s
```

StatefulSet的缩容

```
[root@k8s-master01 ~]# kubectl scale --replicas=2 sts web
statefulset.apps/web scaled

# 先删最后一个
[root@k8s-master01 ~]# kubectl get pod
NAME                          READY   STATUS        RESTARTS   AGE
busybox                       1/1     Running       0          3m41s
nginx-66bbc9fdc5-vc2gh        1/1     Running       0          12h
nginx-v2-644cd9ccc7-b99nf     1/1     Running       0          12h
web-0                         1/1     Running       0          12m
web-1                         1/1     Running       0          12m
web-2                         0/1     Terminating   0          7m45s
```

StatefulSet更新策略

On Delete策略

OnDelete更新策略实现了传统（1.7版本之前）的行为，它也是默认的更新策略。当我们选择这个更新策略并修改StatefulSet的.spec.template字段时，StatefulSet控制器不会自动更新Pod，我们必须手动删除Pod才能使控制器创建新的Pod。

RollingUpdate策略

RollingUpdate（滚动更新）更新策略会更新一个StatefulSet中所有的Pod，采用与序号索引相反的顺序进行滚动更新

比如Patch一个名称为web的StatefulSet来执行RollingUpdate更新：

```
[root@k8s-master01]# kubectl patch statefulset web -p '{"spec":{"updateStrategy":
{"type":"RollingUpdate"}}}'
statefulset.apps/web patched

查看更改后的StatefulSet:
[root@k8s-master01 2.2.7]# kubectl get sts web -o yaml | grep -A 1 "updateStrategy"
  updateStrategy:
    type: RollingUpdate

然后改变容器的镜像进行滚动更新:
[root@k8s-master01 2.2.7]# kubectl patch statefulset web --type='json' -p='[{"op":
"replace", "path": "/spec/template/spec/containers/0/image",
"value":"dotbalo/canary:v1"}]'
statefulset.apps/web patched

  如上所述，StatefulSet里的Pod采用和序号相反的顺序更新。在更新下一个Pod前，StatefulSet控制器会终止每一个Pod并等待它们变成Running和Ready状态。在当前顺序变成Running和Ready状态之前，StatefulSet控制器不会更新下一个Pod，但它仍然会重建任何在更新过程中发生故障的Pod，使用它们当前的版本。已经接收到请求的Pod将会被恢复为更新的版本，没有收到请求的Pod则会被恢复为之前的版本

  在更新过程中可以使用 kubectl rollout status sts/<name> 来查看滚动更新的状态:
[root@k8s-master01 ~]# kubectl rollout status sts/web
```

分段更新（partition）

```
# 比如我们定义一个分区"partition":3，可以使用patch直接对StatefulSet进行设置
[root@k8s-master01 ~]# kubectl patch statefulset web -p '{"spec":{"updateStrategy":
{"type":"RollingUpdate","rollingUpdate":{"partition":3}}}}'
statefulset.apps/web patched

  # 然后再次patch改变容器的镜像:
[root@k8s-master01 ~]# kubectl patch statefulset web --type='json' -p='[{"op":
"replace", "path": "/spec/template/spec/containers/0/image", "value":"k8s.gcr.io/nginx-
slim:0.7"}]'
statefulset.apps/web patched
  # 删除Pod触发更新
[root@k8s-master01 ~]# kubectl delete po web-1
pod "web-1" deleted
```

删除StatefulSet

删除StatefulSet有两种方式，即级联删除和非级联删除。使用非级联方式删除StatefulSet时，StatefulSet的Pod不会被删除；使用级联删除时，StatefulSet和它的Pod都会被删除

级联删除

使用kubectl delete sts xxx删除StatefulSet时，只需提供--cascade=false参数，就会采用非级联删除，此时删除StatefulSet不会删除它的Pod

```
[root@k8s-master01 ~]# kubectl get po
NAME                        READY   STATUS    RESTARTS   AGE
busybox                     1/1     Running   0          56m
nginx-66bbc9fdc5-vc2gh      1/1     Running   0          13h
nginx-v2-644cd9ccc7-b99nf   1/1     Running   0          13h
web-0                       1/1     Running   0          64m
web-1                       1/1     Running   0          2m54s

# 删除sts
[root@k8s-master01 ~]# kubectl delete statefulset web --cascade=false
warning: --cascade=false is deprecated (boolean value) and can be replaced with --
cascade=orphan.
statefulset.apps "web" deleted

# 可以看到已经删除了sts，但是pod还在
[root@k8s-master01 ~]# kubectl get sts
No resources found in default namespace.
[root@k8s-master01 ~]# kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
busybox                     1/1     Running   0          57m
nginx-66bbc9fdc5-vc2gh      1/1     Running   0          13h
nginx-v2-644cd9ccc7-b99nf   1/1     Running   0          13h
web-0                       1/1     Running   0          65m
web-1                       1/1     Running   0          4m

# 由于此时删除了StatefulSet，因此单独删除Pod时，不会被重建
[root@k8s-master01 ~]# kubectl get pod
NAME                        READY   STATUS    RESTARTS   AGE
busybox                     1/1     Running   0          59m
nginx-66bbc9fdc5-vc2gh      1/1     Running   0          13h
nginx-v2-644cd9ccc7-b99nf   1/1     Running   0          13h
web-0                       1/1     Running   0          68m
web-1                       1/1     Running   0          6m21s
[root@k8s-master01 ~]# kubectl delete pod web-1
pod "web-1" deleted
[root@k8s-master01 ~]# kubectl get po
NAME                        READY   STATUS    RESTARTS   AGE
busybox                     1/1     Running   0          60m
nginx-66bbc9fdc5-vc2gh      1/1     Running   0          13h
nginx-v2-644cd9ccc7-b99nf   1/1     Running   0          13h
web-0                       1/1     Running   0          68m

# 当再次创建此StatefulSet时，web-0会被重新创建，web-1由于已经存在而不会被再次创建，因为最初此
StatefulSet的replicas是2，所以web-2会被删除，如下（忽略AlreadyExists错误）
```

```
[root@k8s-master01 ~]# kubectl create -f nginx-sts.yaml
statefulset.apps/web created
Error from server (AlreadyExists): error when creating "nginx-sts.yaml": services
"nginx" already exists

[root@k8s-master01 ~]# kubectl get po
NAME                          READY   STATUS    RESTARTS   AGE
busybox                       1/1     Running   1          64m
nginx-66bbc9fdc5-vc2gh        1/1     Running   0          13h
nginx-v2-644cd9ccc7-b99nf     1/1     Running   0          13h
web-0                         1/1     Running   0          72m
web-1                         1/1     Running   0          26s
web-2                         1/1     Running   0          24s
```

非级联删除

省略--cascade=false参数即为级联删除

```
[root@k8s-master01 ~]# kubectl delete statefulset web
statefulset.apps "web" deleted
```

也可以使用-f参数直接删除StatefulSet和Service（此文件将sts和svc写在了一起）

```
[root@k8s-master01 ~]# kubectl delete -f nginx-sts.yaml
service "nginx" deleted
Error from server (NotFound): error when deleting "nginx-sts.yaml": statefulsets.apps
"web" not found
```

DaemonSet

什么是DaemonSet

DaemonSet（守护进程集）和守护进程类似，它在符合匹配条件的节点上均部署一个Pod

```
 DaemonSet确保全部（或者某些）节点上运行一个Pod副本。当有新节点加入集群时，也会为它们新增一个Pod。当节点从集群中移除时，这些Pod也会被回收，删除DaemonSet将会删除它创建的所有Pod
  使用DaemonSet的一些典型用法：
   运行集群存储daemon（守护进程），例如在每个节点上运行Glusterd、Ceph等
   在每个节点运行日志收集daemon，例如Fluentd、Logstash
   在每个节点运行监控daemon，比如Prometheus Node Exporter、Collectd、Datadog代理、New Relic代理或 Ganglia gmond
```

编写DaemonSet规范

创建一个DaemonSet的内容大致如下，比如创建一个fluentd的DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-es-v2.0.4
  namespace: logging
  labels:
    k8s-app: fluentd-es
    version: v2.0.4
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-es
      version: v2.0.4
  template:
    metadata:
      labels:
        k8s-app: fluentd-es
        kubernetes.io/cluster-service: "true"
        version: v2.0.4
      # This annotation ensures that fluentd does not get evicted if the node
      # supports critical pod annotation based priority scheme.
      # Note that this does not guarantee admission on the nodes (#40573).
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
        seccomp.security.alpha.kubernetes.io/pod: 'docker/default'
    spec:
      serviceAccountName: fluentd-es
      containers:
      - name: fluentd-es
        image: k8s.gcr.io/fluentd-elasticsearch:v2.0.4
        env:
        - name: FLUENTD_ARGS
          value: --no-supervisor -q
        resources:
          limits:
            memory: 500Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
```

```
          mountPath: /var/lib/docker/containers
          readOnly: true
        - name: config-volume
          mountPath: /etc/fluent/config.d
      nodeSelector:
        beta.kubernetes.io/fluentd-ds-ready: "true"
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
      - name: config-volume
        configMap:
          name: fluentd-es-config-v0.1.4
```

创建一个DaemonSet

```
[root@k8s-master01 ~]# cat > nginx-ds.yaml  << EFO
apiVersion: apps/v1
kind: DaemonSet
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:1.15.2
        imagePullPolicy: IfNotPresent
        name: nginx
        resources: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Alwaysyaml
```

```
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
EFO
```

```
# 创建一个ds
[root@k8s-master01 ~]# kubectl create -f nginx-ds.yaml
daemonset.apps/nginx created

# 查看ds信息，个个节点都有一个
[root@k8s-master01 ~]# kubectl get node -owide
NAME           STATUS   ROLES    AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE
               KERNEL-VERSION                CONTAINER-RUNTIME
k8s-master01   Ready    matser   43h   v1.20.0   192.168.1.100   <none>        CentOS
Linux 7 (Core)   4.19.12-1.el7.elrepo.x86_64   docker://19.3.14
k8s-master02   Ready    <none>   43h   v1.20.0   192.168.1.101   <none>        CentOS
Linux 7 (Core)   4.19.12-1.el7.elrepo.x86_64   docker://19.3.14
k8s-master03   Ready    <none>   43h   v1.20.0   192.168.1.102   <none>        CentOS
Linux 7 (Core)   4.19.12-1.el7.elrepo.x86_64   docker://19.3.14
k8s-node01     Ready    <none>   43h   v1.20.0   192.168.1.103   <none>        CentOS
Linux 7 (Core)   4.19.12-1.el7.elrepo.x86_64   docker://19.3.14
k8s-node02     Ready    <none>   43h   v1.20.0   192.168.1.104   <none>        CentOS
Linux 7 (Core)   4.19.12-1.el7.elrepo.x86_64   docker://19.3.14
```

1. 必需字段

和其他所有Kubernetes配置一样，DaemonSet需要apiVersion、kind和metadata字段，同时也需要一个.spec配置段。

2. Pod模板

.spec唯一需要的字段是.spec.template。.spec.template是一个Pod模板，它与Pod具有相同的配置方式，但它不具有apiVersion和kind字段。

除了Pod必需的字段外，在DaemonSet中的Pod模板必须指定合理的标签。

在DaemonSet中的Pod模板必须具有一个RestartPolicy，默认为Always。

3. Pod Selector

.spec.selector字段表示Pod Selector，它与其他资源的.spec.selector的作用相同。

.spec.selector表示一个对象，它由如下两个字段组成：

matchLabels，与ReplicationController的.spec.selector的作用相同，用于匹配符合条件的Pod。

matchExpressions，允许构建更加复杂的Selector，可以通过指定key、value列表以及与key和value列表相关的操作符。

如果上述两个字段都指定时，结果表示的是AND关系（逻辑与的关系）。

.spec.selector必须与.spec.template.metadata.labels相匹配。如果没有指定，默认是等价的，如果它们的配置不匹配，则会被API拒绝。

（4）指定节点部署Pod

如果指定了.spec.template.spec.nodeSelector，DaemonSet Controller将在与Node Selector（节点选择器）匹配的节点上创建Pod，比如部署在磁盘类型为ssd的节点上（需要提前给节点定义标签Label）：

```
    containers:
```

```
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
      nodeSelector:
        disktype: ssd
```

```
1. 命令式更新
kubectl edit ds/<daemonset-name>
kubectl patch ds/<daemonset-name> -p=<strategic-merge-patch>

2. 更新镜像
kubectl set image ds/<daemonset-name><container-name>=<container-new-image>--
record=true

3. 查看更新状态
kubectl rollout status ds/<daemonset-name>

4. 列出所有修订版本
kubectl rollout history daemonset <daemonset-name>

5. 回滚到指定revision
kubectl rollout undo daemonset <daemonset-name> --to-revision=<revision>
DaemonSet的更新和回滚与Deployment类似，此处不再演示。
```