Kubernetes Pod控制器

控制器

Kubernetes 中内建了很多 controller（控制器），用来确保pod资源符合预期的状态，控制 Pod 的状态和行为。

控制器类型

- ReplicaSet(rs)
- Deployment(deploy)
- DaemonSet(ds)
- StatefulSet(sts)
- Job/CronJob(cj)
- Horizontal Pod Autoscaling(hpa)

```
# 可以使用kubectl explain命令查看k8s API资源对象描述信息
[root@k8s-master ~]# kubectl explain rs
KIND:     ReplicaSet
VERSION:  apps/v1

DESCRIPTION:
     ReplicaSet ensures that a specified number of pod replicas are running at
     any given time.

FIELDS:
     ...

# 查看API资源列表
[root@k8s-master ~]# kubectl api-resources
NAME                              SHORTNAMES   APIGROUP
NAMESPACED   KIND
bindings                                                                      true
Binding
componentstatuses                 cs
false        ComponentStatus
configmaps                        cm                                          true
ConfigMap
endpoints                         ep                                          true
Endpoints
events                            ev                                          true
Event
limitranges                       limits                                      true
LimitRange
namespaces                        ns
false        Namespace
nodes                             no
false        Node
persistentvolumeclaims            pvc                                         true
```

```
PersistentVolumeClaim
persistentvolumes                 pv
false        PersistentVolume
pods                              po                                    true
Pod
replicationcontrollers            rc                                    true
ReplicationController
resourcequotas                    quota                                 true
ResourceQuota
serviceaccounts                   sa                                    true
ServiceAccount
services                          svc                                   true
Service
customresourcedefinitions         crd,crds     apiextensions.k8s.io
false        CustomResourceDefinition
daemonsets                        ds           apps                     true
DaemonSet
deployments                       deploy       apps                     true
Deployment
replicasets                       rs           apps                     true
ReplicaSet
statefulsets                      sts          apps                     true
StatefulSet
horizontalpodautoscalers          hpa          autoscaling              true
HorizontalPodAutoscaler
cronjobs                          cj           batch                    true
CronJob
jobs                                           batch                    true
Job
certificatesigningrequests        csr          certificates.k8s.io
false
networkpolicies                                crd.projectcalico.org    true
NetworkPolicy
networksets                                    crd.projectcalico.org    true
NetworkSet
endpointslices                                 discovery.k8s.io         true
EndpointSlice
events                            ev           events.k8s.io            true
Event
ingresses                         ing          extensions               true
Ingress
...
```

ReplicaSet 和 ReplicationController

ReplicationController(RC)用来确保容器应用的副本数始终保持在用户定义的副本数，即如果有容器异常退出，会自动创建新的 Pod 来替代；而如果异常多出来的容器也会自动回收。

在新版本的 Kubernetes 中建议使用 ReplicaSet 来取代 ReplicationController，ReplicaSet规则跟 ReplicationController 没有本质的不同，只是名字不一样，并且 ReplicaSet 支持集合式的 selector。

rc实现pod动态缩放

- 当前RC和pod情况是

```
[root@k8s-master ~]# kubectl get pods,rc
NAME                          READY   STATUS    RESTARTS   AGE
pod/mysql-hdg66               1/1     Running   3          24h
pod/myweb-ctzhn               1/1     Running   3          24h
pod/myweb-dm94j               1/1     Running   3          24h

NAME                            DESIRED   CURRENT   READY   AGE
replicationcontroller/mysql   1         1         1       24h
replicationcontroller/myweb   2         2         2       24h
```

- 增加pod-mysql的副本(RC)数

```
[root@k8s-master ~]# kubectl scale rc mysql --replicas=3
replicationcontroller/mysql scaled
[root@k8s-master ~]# kubectl get pods,rc
NAME                          READY   STATUS    RESTARTS   AGE
pod/mysql-bqdvv               1/1     Running   0          5s
pod/mysql-hdg66               1/1     Running   3          24h
pod/mysql-hhb6t               1/1     Running   0          5s
pod/myweb-ctzhn               1/1     Running   3          24h
pod/myweb-dm94j               1/1     Running   3          24h

NAME                            DESIRED   CURRENT   READY   AGE
replicationcontroller/mysql   3         3         3       24h
replicationcontroller/myweb   2         2         2       24h
```

- 减少pod-mysql的副本(RC)数

```
[root@k8s-master ~]# kubectl scale rc mysql --replicas=1
replicationcontroller/mysql scaled
# 正在停止多余的副本
[root@k8s-master ~]# kubectl get pods,rc
NAME                          READY   STATUS        RESTARTS   AGE
pod/mysql-bqdvv               1/1     Terminating   0          62s
pod/mysql-hdg66               1/1     Running       3          24h
pod/mysql-hhb6t               1/1     Terminating   0          62s
pod/myweb-ctzhn               1/1     Running       3          24h
pod/myweb-dm94j               1/1     Running       3          24h

NAME                            DESIRED   CURRENT   READY   AGE
replicationcontroller/mysql   1         1         1       24h
replicationcontroller/myweb   2         2         2       24h
# 缩放完成后
[root@k8s-master ~]# kubectl get pods,rc
NAME                          READY   STATUS    RESTARTS   AGE
pod/mysql-hdg66               1/1     Running   3          24h
```

```
pod/myweb-ctzhn               1/1     Running   3          24h
pod/myweb-dm94j               1/1     Running   3          24h

NAME                           DESIRED   CURRENT   READY   AGE
replicationcontroller/mysql    1         1         1       24h
replicationcontroller/myweb    2         2         2       24h
```

Deployment

Deployment 是一种更高级别的 API 对象，为 Pods 和 ReplicaSets 提供声明式的更新能力。它以类似于 `kubectl rolling-update` 的方式更新其底层 ReplicaSet 及其 Pod。 如果需要这种滚动更新功能，推荐使用 Deployment。

Deployments 的典型用例：

- 创建 Deployment 以将 ReplicaSet 上线。ReplicaSet 在后台创建 Pods。检查 ReplicaSet 的上线状态，查看其是否成功。
- 通过更新 Deployment 的 PodTemplateSpec，声明 Pod 的新状态。新的 ReplicaSet 会被创建，Deployment 以受控速率将 Pod 从旧 ReplicaSet 迁移到新 ReplicaSet。 每个新的 ReplicaSet 都会更新 Deployment 的修订版本。
- 如果 Deployment 的当前状态不稳定，回滚到较早的 Deployment 版本。每次回滚都会更新 Deployment 的修订版本。
- 扩大 Deployment 规模以承担更多负载。
- 暂停 Deployment 以应用对 PodTemplateSpec 所作的多项修改，然后恢复其执行以启动新的上线版本。
- 使用 Deployment 状态 来判定上线过程是否出现停滞。
- 清理较旧的不再需要的 ReplicaSet。

创建 Deployment

下面是 Deployment 示例。其中创建了一个 ReplicaSet，负责启动三个 nginx Pods：

```
# 创建yaml文件
[root@k8s-master manifests]# vi nginx-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
```

```
      containers:
      - name: nginx
        image: nginx:1.16.1
        ports:
        - containerPort: 80
[root@k8s-master manifests]# kubectl apply -f nginx-deploy.yaml
deployment.apps/nginx-deploy created
[root@k8s-master manifests]# kubectl get po,deploy,rs
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deploy-559d658b74-27jn4   1/1     Running   0          100s
pod/nginx-deploy-559d658b74-hzdr2   1/1     Running   0          100s
pod/nginx-deploy-559d658b74-v7rhq   1/1     Running   0          100s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deploy   3/3     3            3           101s

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deploy-559d658b74   3         3         3       100s
# 查看标签
[root@k8s-master manifests]# kubectl get po --show-labels
NAME                           READY   STATUS    RESTARTS   AGE     LABELS
nginx-deploy-559d658b74-27jn4   1/1     Running   0          2m20s
app=nginx,pod-template-hash=559d658b74
nginx-deploy-559d658b74-hzdr2   1/1     Running   0          2m20s
app=nginx,pod-template-hash=559d658b74
nginx-deploy-559d658b74-v7rhq   1/1     Running   0          2m20s
app=nginx,pod-template-hash=559d658b74
# 扩缩容
[root@k8s-master manifests]# kubectl scale deployment nginx-deploy --replicas=2
deployment.apps/nginx-deploy scaled
[root@k8s-master manifests]# kubectl get po,deploy,rs
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deploy-559d658b74-27jn4   1/1     Running   0          10m
pod/nginx-deploy-559d658b74-hzdr2   1/1     Running   0          10m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deploy   2/2     2            2           10m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deploy-559d658b74   2         2         2       10m
# 查看pod详情
[root@k8s-master manifests]# kubectl get pods -o wide
NAME                           READY   STATUS    RESTARTS   AGE   IP
NODE        NOMINATED NODE   READINESS GATES
nginx-deploy-559d658b74-27jn4   1/1     Running   0          11m   172.16.36.105
k8s-node1   <none>           <none>
nginx-deploy-559d658b74-hzdr2   1/1     Running   0          11m   172.16.36.106
k8s-node1   <none>           <none>
# 访问nginx
[root@k8s-master manifests]# curl 172.16.36.105
...
<title>Welcome to nginx!</title>
...
```

定向调度(nodeSelector)

Kubernetes上kube-scheduler负责pod调度，通过内置算法实现最佳节点的调度，当然也可以指定调度的节点

```
# 给k8s-node1节点打上test标签
[root@k8s-master manifests]# kubectl label nodes k8s-node1 zone=test
node/k8s-node1 labeled
# 查看node的标签
[root@k8s-master ~]# kubectl get nodes k8s-node1 --show-labels
NAME        STATUS   ROLES    AGE   VERSION   LABELS
k8s-node1   Ready    node     13d   v1.19.3
beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64
,kubernetes.io/hostname=k8s-node1,kubernetes.io/os=linux,node-
role.kubernetes.io/node=,zone=test
# 或者从描述里查看
[root@k8s-master manifests]# kubectl describe node k8s-node1
Name:               k8s-node1
Roles:              node
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=k8s-node1
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/node=
                    zone=test
# 给pod加上定向调度设置
[root@k8s-master manifests]# vi nginx-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.16.1
        ports:
        - containerPort: 80
      nodeSelector:
        zone: test
[root@k8s-master manifests]# kubectl apply -f nginx-deploy.yaml
```

```
deployment.apps/nginx-deploy created
# 查看到所有的pod均部署在k8s-node1上
[root@k8s-master manifests]# kubectl get pods -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP
NODE        NOMINATED NODE   READINESS GATES
nginx-deploy-79bf6fcf-5b7v6   1/1     Running   0          11s   172.16.36.126
k8s-node1   <none>           <none>
nginx-deploy-79bf6fcf-d4hz6   1/1     Running   0          11s   172.16.36.68
k8s-node1   <none>           <none>
nginx-deploy-79bf6fcf-thbpt   1/1     Running   0          11s   172.16.36.127
k8s-node1   <none>           <none>
```

当然也可以通过`kubectl get nodes k8s-node1 --show-labels`查到的系统标签进行定向调度

亲和与反亲和调度

定向调度比较是一种强制分配的方式进行pod调度，推荐使用亲和性调度代替定向调度，亲和性调度有下面两种表达：

- requiredDuringSchedulingIgnoredDuringExecution: hard(硬限制)，严格执行，满足规则调度，否则不调度
- preferredDuringSchedulingIgnoredDuringExecution：soft(软限制)，尽力执行，优先满足规则调度，多个规则可用权重来决定先执行哪一个

OPerator参数：

- In：label的值在某个列表中
- NotIn：label的值不在某个列表中
- Gt：label的值大于某个值
- Lt：label的值小于某个值
- Exists：某个label存在
- DoesNotExist：某个label不存在

node亲和调度(nodeAffinity)

Note: 支持的operator操作： In, NotIn, Exists, DoesNotExist, Gt, Lt. 其中，NotIn and DoesNotExist用于实现反亲和性。

Note: weight范围1-100。这个涉及调度器的优选打分过程，每个node的评分都会加上这个weight，最后bind最高的node。

```
# 第一个规则限制只运行在amd64架构的节点上，第二个规则是尽量调度到在k8s-node1节点上
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
```

```
          nodeSelectorTerms:
          - matchExpressions:
            - key: beta.kubernetes.io/arch
              operator: In
              values:
              - amd64
        preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          preference:
            matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
              - k8s-node1
    containers:
    - name: with-node-affinity
      image: k8s.gcr.io/pause:2.0
```

pod亲和和反亲和调度

Pod的亲和性与反亲和性是基于Node节点上已经运行pod的标签(而不是节点上的标签)决定的，从而约束哪些节点适合调度pod。

规则是：如果X已经运行了一个或多个符合规则Y的pod，则此pod应该在X中运行(如果是反亲和的情况下，则不应该在X中运行）。当然pod必须处在同一名称空间，不然亲和性/反亲和性无作用。
X是一个拓扑域，可以使用topologyKey来表示它，topologyKey 的值是node节点标签的键以便系统用来表示这样的拓扑域。当然这里也有个隐藏条件，就是node节点标签的键值相同时，才是在同一拓扑域中；如果只是节点标签名相同，但是值不同，那么也不在同一拓扑域。

Pod的亲和性/反亲和性调度是根据拓扑域来界定调度的，而不是根据node节点。

Pod: 支持的operator操作：In, NotIn, Exists, DoesNotExist, Gt, Lt.

```
# 创建参照deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-flag
  labels:
    seccurity: s1
    app: nginx
spec:
  containers:
    - name: nginx
      image: nginx
# 亲和反亲和配置实例
apiVersion: apps/v1
kind: Deployment
metadata:
  name: affinity-all
  labels:
```

```
    app: affinity-all
spec:
  containers:
  - name: affinity-all
    image: k8s.gcr.io/pause:2.0
  affinity:
    # pod亲和性
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
          matchExpressions:
          - key: security
            operator: In
            values:
            - s1
        # 拓扑域
        topologyKey: disk-type
    # pod反亲和性
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          # 由于是Pod亲和性/反亲和性；因此这里匹配规则写的是Pod的标签信息
          matchExpressions:
          - key: app
            operator: In
            values:
            - nginx
        # 拓扑域
        topologyKey: kubernetes.io/hostname
```

上面创建的deployment应满足下面规则：

- 与security=s1的pod为同一种disk-type(同一种磁盘的拓扑域)
- 不与app=nginx的pod调度在同一node节点上

污点和容忍(Taints和Tolerations)

Taint需要和Toleration配合使用，让pod避开某些节点，除非pod创建时声明容忍策略，否则不会在有污点的节点上运行。

```
# 为k8s-node1设置不能调度的污点
[root@k8s-master manifests]# kubectl taint nodes k8s-node1 test=node1:NoSchedule
# 如果创建pod时设置容忍策略，则该pod能够（不是必须）被分配到该节点，具体能不能分配到该节点上由分配算法决定
# 常见的容忍配置
tolerations:
- key: "key"
  operator: "Equal"
  value: "value"
  effect: "NoSchedule"
```

```
---
tolerations:
- key: "key"
  operator: "Exists"
  effect: "NoSchedule"
---
tolerations:
- key: "key"
  operator: "Equal"
  value: "value"
  effect: "NoExecute"
  tolerationSeconds: 3600

# 在yaml文件中的位置
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test
  labels:
    app: test
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: test
    spec:
      containers:
      - name: test
        image: nginx
      tolerations:
      - key: "test"
        operator: "Exists"
        effect: "NoSchedule"
```

升级更新 Deployment

```
[root@k8s-master manifests]# kubectl set image deployment/nginx-deploy
nginx=nginx:1.18.0 --record
deployment.apps/nginx-deploy image updated
[root@k8s-master manifests]# kubectl get po,deploy,rs
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deploy-67dfd6c8f9-gxj29   1/1     Running   0          43s
pod/nginx-deploy-67dfd6c8f9-xm68b   1/1     Running   0          45s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deploy   2/2     2            2           17m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deploy-559d658b74   0         0         0       17m
replicaset.apps/nginx-deploy-67dfd6c8f9   2         2         2       45s
```

```
[root@k8s-master manifests]# kubectl get pods -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP
NODE        NOMINATED NODE   READINESS GATES
nginx-deploy-67dfd6c8f9-gxj29   1/1     Running   0          3m35s   172.16.36.109
k8s-node1   <none>           <none>
nginx-deploy-67dfd6c8f9-xm68b   1/1     Running   0          3m37s   172.16.36.108
k8s-node1   <none>           <none>
```

回滚 Deployment

```
[root@k8s-master manifests]# kubectl rollout undo deployment/nginx-deploy
deployment.apps/nginx-deploy rolled back
[root@k8s-master manifests]# kubectl get po,deploy,rs
NAME                                READY   STATUS        RESTARTS   AGE
pod/nginx-deploy-559d658b74-9dgg8   1/1     Running       0          8s
pod/nginx-deploy-559d658b74-pgrqv   1/1     Running       0          10s
pod/nginx-deploy-67dfd6c8f9-gxj29   0/1     Terminating   0          4m42s
pod/nginx-deploy-67dfd6c8f9-xm68b   0/1     Terminating   0          4m44s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deploy   2/2     2            2           21m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deploy-559d658b74   2         2         2       21m
replicaset.apps/nginx-deploy-67dfd6c8f9   0         0         0       4m44s
[root@k8s-master manifests]# kubectl get pods -o wide
NAME                            READY   STATUS    RESTARTS   AGE   IP
NODE        NOMINATED NODE   READINESS GATES
nginx-deploy-559d658b74-9dgg8   1/1     Running   0          59s   172.16.36.111
k8s-node1   <none>           <none>
nginx-deploy-559d658b74-pgrqv   1/1     Running   0          61s   172.16.36.110
k8s-node1   <none>           <none>
```

更新过程记录

```
[root@k8s-master manifests]# kubectl describe po nginx-deploy-559d658b74-9dgg8
Labels:       app=nginx
              pod-template-hash=559d658b74
Annotations:  cni.projectcalico.org/podIP: 172.16.36.111/32
              cni.projectcalico.org/podIPs: 172.16.36.111/32
Status:       Running
IP:           172.16.36.111
IPs:
  IP:           172.16.36.111
Controlled By:  ReplicaSet/nginx-deploy-559d658b74
Containers:
  nginx:
    Container ID:
docker://ddb4e8852ec11faa7e784f43d554c679772d6f47fd3473e3ea138cbc3bbc60c0
```

```
    Image:          nginx:1.16.1
    Image ID:       docker-
pullable://nginx@sha256:d20aa6d1cae56fd17cd458f4807e0de462caf2336f0b70b5eeb69fcaaf
30dd9c
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Mon, 23 Nov 2020 02:46:15 -0500
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-64lwm (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-64lwm:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-64lwm
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  111s  default-scheduler  Successfully assigned default/nginx-
deploy-559d658b74-9dgg8 to k8s-node1
  Normal  Pulled     110s  kubelet            Container image "nginx:1.16.1"
already present on machine
  Normal  Created    110s  kubelet            Created container nginx
  Normal  Started    109s  kubelet            Started container nginx
[root@k8s-master manifests]# kubectl set image deployment/nginx-deploy
nginx=nginx:1.18.0 --record
deployment.apps/nginx-deploy image updated
[root@k8s-master manifests]# kubectl get po,deploy,rs
NAME                                READY   STATUS    RESTARTS   AGE
pod/nginx-deploy-67dfd6c8f9-mkmmn   1/1     Running   0          70s
pod/nginx-deploy-67dfd6c8f9-tj2j8   1/1     Running   0          72s

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nginx-deploy   2/2     2            2           26m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/nginx-deploy-559d658b74   0         0         0       26m
replicaset.apps/nginx-deploy-67dfd6c8f9   2         2         2       9m17s
[root@k8s-master manifests]# kubectl describe po nginx-deploy-67dfd6c8f9-mkmmn
Name:         nginx-deploy-67dfd6c8f9-mkmmn
Namespace:    default
```

```
Priority:     0
Node:         k8s-node1/192.168.43.20
Start Time:   Mon, 23 Nov 2020 02:49:44 -0500
Labels:       app=nginx
              pod-template-hash=67dfd6c8f9
Annotations:  cni.projectcalico.org/podIP: 172.16.36.113/32
              cni.projectcalico.org/podIPs: 172.16.36.113/32
Status:       Running
IP:           172.16.36.113
IPs:
  IP:           172.16.36.113
Controlled By:  ReplicaSet/nginx-deploy-67dfd6c8f9
Containers:
  nginx:
    Container ID:
docker://d260e6d5b361d0344c450201e25bc5c227b0b1d4bf1dae79254fdadea914d18e
    Image:          nginx:1.18.0
    Image ID:       docker-
pullable://nginx@sha256:2104430ec73de095df553d0c7c2593813e01716a48d66f85a3dc439e05
0919b3
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Mon, 23 Nov 2020 02:49:46 -0500
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-64lwm (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-64lwm:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-64lwm
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age   From               Message
  ----    ------     ----  ----               -------
  Normal  Scheduled  95s   default-scheduler  Successfully assigned default/nginx-
deploy-67dfd6c8f9-mkmmn to k8s-node1
  Normal  Pulled     93s   kubelet            Container image "nginx:1.18.0"
already present on machine
  Normal  Created    93s   kubelet            Created container nginx
  Normal  Started    93s   kubelet            Started container nginx
# 查看回滚状态
```

```
[root@k8s-master manifests]# kubectl rollout status deployments nginx-deploy
deployment "nginx-deploy" successfully rolled out
# 历史记录
[root@k8s-master manifests]# kubectl rollout history deployments nginx-deploy
deployment.apps/nginx-deploy
REVISION  CHANGE-CAUSE
3         <none>
4         kubectl set image deployment/nginx-deploy nginx=nginx:1.18.0 --
record=true
# 查看deployment详情
[root@k8s-master manifests]# kubectl describe deploy nginx-deploy
Name:                   nginx-deploy
Namespace:              default
CreationTimestamp:      Mon, 23 Nov 2020 02:24:33 -0500
Labels:                 app=nginx
Annotations:            deployment.kubernetes.io/revision: 4
                        kubernetes.io/change-cause: kubectl set image
deployment/nginx-deploy nginx=nginx:1.18.0 --record=true
Selector:               app=nginx
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0
unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:        nginx:1.18.0
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deploy-67dfd6c8f9 (2/2 replicas created)
Events:
  Type    Reason             Age                From                   Message
  ----    ------             ----               ----                   -------
  Normal  ScalingReplicaSet  40m                deployment-controller  Scaled up
replica set nginx-deploy-559d658b74 to 3
  Normal  ScalingReplicaSet  30m                deployment-controller  Scaled down
replica set nginx-deploy-559d658b74 to 2
  Normal  ScalingReplicaSet  18m                deployment-controller  Scaled up
replica set nginx-deploy-559d658b74 to 1
  Normal  ScalingReplicaSet  18m                deployment-controller  Scaled up
replica set nginx-deploy-559d658b74 to 2
  Normal  ScalingReplicaSet  18m                deployment-controller  Scaled down
replica set nginx-deploy-67dfd6c8f9 to 1
```

```
  Normal  ScalingReplicaSet  18m                deployment-controller  Scaled down
replica set nginx-deploy-67dfd6c8f9 to 0
  Normal  ScalingReplicaSet  15m (x2 over 23m)  deployment-controller  Scaled up
replica set nginx-deploy-67dfd6c8f9 to 1
  Normal  ScalingReplicaSet  15m (x2 over 23m)  deployment-controller  Scaled up
replica set nginx-deploy-67dfd6c8f9 to 2
  Normal  ScalingReplicaSet  15m (x2 over 23m)  deployment-controller  Scaled down
replica set nginx-deploy-559d658b74 to 1
  Normal  ScalingReplicaSet  15m (x2 over 23m)  deployment-controller  Scaled down
replica set nginx-deploy-559d658b74 to 0
# 可以使用 --revision参数指定某个历史版本
kubectl rollout undo deployment/nginx-deploy --to-revision=2
# 暂停 deployment 的更新
kubectl rollout pause deployment/nginx-deploy
```

DeamonSet

DaemonSet 确保全部（或者一些）Node上运行一个 Pod 的副本。当有 Node 加入集群时，也会为它们新增一个 Pod，当有 Node 从集群移除时，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 deamon，例如在每个 Node 上运行 glusterd、ceph
- 在每个 Node 上运行日志收集 deamon，例如 fluentd、logstash
- 在每个 Node 上运行监控 daemon，例如 Prometheus Node Exporter

```
[root@k8s-master manifests]# kubectl explain ds
KIND:     DaemonSet
VERSION:  apps/v1

DESCRIPTION:
     DaemonSet represents the configuration of a daemon set.

FIELDS:
...
# 创建yaml
[root@k8s-master manifests]# vi daemonset-example.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: daemonset-example
  labels:
    app: daemonset
spec:
  selector:
    matchLabels:
      name: daemonset-example
  template:
    metadata:
      labels:
        name: daemonset-example
```

```
    spec:
      containers:
      - name: daemonset-example
        image: wangyanglinux/myapp:v1
[root@k8s-master manifests]# kubectl get po
NAME                      READY   STATUS    RESTARTS   AGE
daemonset-example-f57kg   1/1     Running   0          25s
[root@k8s-master manifests]# kubectl get ds
NAME                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset-example   1         1         1       1            1           <none>
27s
[root@k8s-master manifests]# kubectl get po -o wide
NAME                      READY   STATUS    RESTARTS   AGE   IP              NODE
NOMINATED NODE   READINESS GATES
daemonset-example-f57kg   1/1     Running   0          74s   172.16.36.118   k8s-
node1   <none>           <none>
[root@k8s-master manifests]# curl 172.16.36.118
Hello MyApp | Version: v1 | <a href="hostname.html">Pod Name</a>
[root@k8s-master manifests]# kubectl describe po daemonset-example-f57kg
Name:         daemonset-example-f57kg
Namespace:    default
Priority:     0
Node:         k8s-node1/192.168.43.20
Start Time:   Mon, 23 Nov 2020 04:38:37 -0500
Labels:       controller-revision-hash=5867b74f5c
              name=daemonset-example
              pod-template-generation=1
Annotations:  cni.projectcalico.org/podIP: 172.16.36.118/32
              cni.projectcalico.org/podIPs: 172.16.36.118/32
Status:       Running
IP:           172.16.36.118
IPs:
  IP:           172.16.36.118
Controlled By:  DaemonSet/daemonset-example
Containers:
  daemonset-example:
    Container ID:
docker://8f5dbe49a96a7a00a310a6ab0e442db39a55ffd67361cb6432078cc1d13dead3
    Image:          wangyanglinux/myapp:v1
    Image ID:       docker-
pullable://wangyanglinux/myapp@sha256:9c3dc30b5219788b2b8a4b065f548b922a34479577be
fb54b03330999d30d513
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 23 Nov 2020 04:39:00 -0500
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-64lwm (ro)
Conditions:
  Type              Status
```

```
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-64lwm:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-64lwm
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/disk-pressure:NoSchedule op=Exists
                 node.kubernetes.io/memory-pressure:NoSchedule op=Exists
                 node.kubernetes.io/not-ready:NoExecute op=Exists
                 node.kubernetes.io/pid-pressure:NoSchedule op=Exists
                 node.kubernetes.io/unreachable:NoExecute op=Exists
                 node.kubernetes.io/unschedulable:NoSchedule op=Exists
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  2m47s  default-scheduler  Successfully assigned
default/daemonset-example-f57kg to k8s-node1
  Normal  Pulling    2m46s  kubelet            Pulling image
"wangyanglinux/myapp:v1"
  Normal  Pulled     2m24s  kubelet            Successfully pulled image
"wangyanglinux/myapp:v1" in 21.955882149s
  Normal  Created    2m24s  kubelet            Created container daemonset-example
  Normal  Started    2m24s  kubelet            Started container daemonset-example
```

Job

Job 负责批处理任务，即仅执行一次的任务，它保证批处理任务的一个或多个 Pod 成功结束。

```
[root@k8s-master manifests]# kubectl explain job
KIND:     Job
VERSION:  batch/v1

DESCRIPTION:
     Job represents the configuration of a single job.

FIELDS:
...
# 创建yaml
[root@k8s-master manifests]# vi job-example.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
```

```
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(1000)"]
      restartPolicy: Neve
[root@k8s-master manifests]# kubectl get po
NAME                      READY   STATUS      RESTARTS   AGE
daemonset-example-f57kg   1/1     Running     0          13m
pi-2nlj5                  0/1     Completed   0          4m
[root@k8s-master manifests]# kubectl get job
NAME   COMPLETIONS   DURATION   AGE
pi     1/1           3m19s      4m15s
[root@k8s-master manifests]# kubectl get po pi-2nlj5 -o wide
NAME       READY   STATUS      RESTARTS   AGE     IP              NODE
NOMINATED NODE   READINESS GATES
pi-2nlj5   0/1     Completed   0          4m54s   172.16.36.119   k8s-node1
<none>           <none>
# 查看日志可以看到job执行的结果
# 计算出了圆周率后1000位
```

```
[root@k8s-master manifests]# kubectl logs pi-2nlj5
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272489122793818301194912983367336244065664308602139494639522473719070217986094370277053921717629317675238467481846766940513200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892354201995611212902196086403441815981362977477130996051870721134999999837297804995105973173281609631859502445945534690830264252230825334468503526193118817101000313783875288658753320838142061717766914730359825349042875546873115956286388235378759375195778185778053217122680661300192787661119590921642019 9
```

CronJob

Cron Job 管理基于时间的 Job，即：

在给定时间点只运行一次 周期性地在给定时间点运行

典型的用法示例：

在给你写的时间点调度 Job 运行 创建周期性运行的 Job，例如：数据库备份、发送邮件

```
[root@k8s-master manifests]# kubectl explain cj
KIND:     CronJob
VERSION:  batch/v1beta1

DESCRIPTION:
     CronJob represents the configuration of a single cron job.

FIELDS:
...
# 创建cronjob yaml文件
[root@k8s-master manifests]# vi cronjob-example.yaml
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
```

```
    schedule: "*/1 * * * *"
    jobTemplate:
      spec:
        template:
          spec:
            containers:
            - name: hello
              image: busybox
              args:
              - /bin/sh
              - -c
              - date; echo Hello CronJob
            restartPolicy:  OnFailure
[root@k8s-master manifests]# kubectl apply -f cronjob-example.yaml
cronjob.batch/hello created
[root@k8s-master manifests]# kubectl get cj
NAME    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
hello   */1 * * * *   False     0        44s             69s
[root@k8s-master manifests]# kubectl get po
NAME                     READY   STATUS      RESTARTS   AGE
hello-1606186260-4pbtt   0/1     Completed   0          100s
hello-1606186320-gxtkn   0/1     Completed   0          39s
# 查看输出日志
[root@k8s-master manifests]# kubectl logs hello-1606186260-4pbtt
Tue Nov 24 02:51:25 UTC 2020
Hello CronJob
[root@k8s-master manifests]# kubectl logs hello-1606186320-gxtkn
Tue Nov 24 02:52:26 UTC 2020
Hello CronJob
[root@k8s-master manifests]# kubectl get job
NAME               COMPLETIONS   DURATION   AGE
hello-1606186260   1/1           18s        2m42s
hello-1606186320   1/1           18s        101s
hello-1606186380   1/1           30s        41s
# 删除cronjob
[root@k8s-master manifests]# kubectl delete cronjob hello
cronjob.batch "hello" deleted
[root@k8s-master manifests]# kubectl get job
No resources found in default namespace.
```

StatefulSet

StatefulSet 作为 Controller 为 Pod 提供唯一的标识，它可以保证部署和 scale 的顺序。 StatefulSet 是为了解决有状态服务的问题（对应 Deployment 和 ReplicaSet 是为无状态服务而设计），其应用场景包括：

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 稳定的网络标识，即 Pod 重新调度后其 Pod Name 和 Host Name 不变，基于 Headless Service （即没有 Cluster IP 的 Service）来实现 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住所定义的顺序依次进行（即从 0 到 N-1，在下一个 Pod 运行之前所有之前的 Pod 必须都是 Running 和 Ready 状态），基于 init containers 来实现 有序收缩，有序删除（即从 N-1 到 0 ）

Horizontal Pod Autoscaling(HPA)

顾名思义，使 Pod 水平自动缩放，提高集群的整体资源利用率。 Horizontal Pod Autoscaling 仅适用于 Deployment 和 ReplicaSet。在 v1 版本中仅支持根据 Pod 的 CPU 利用率扩缩容，在 v1alpha 版本中，支持根据内存和用户自定义的 metric 扩缩容。

LivenessProbe探针健康检查

Liveness 探测让用户可以自定义判断容器是否健康的条件。如果探测失败，Kubernetes 就会重启容器。

```
# 创建liveness.yaml
[root@k8s-master manifests]# vi liveness.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness
spec:
  restartPolicy: OnFailure
  containers:
  - name: liveness
    image: busybox
    args:
    - /bin/sh
    - -c
    - echo ok > /tmp/healthy; sleep 10; rm -rf /tmp/healthy; sleep 60
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 15
      timeoutSecond: 1

# 查看检查失败的日志以及后续操作
[root@k8s-master manifests]# kubectl describe po liveness
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  114s               default-scheduler  Successfully assigned
default/liveness to k8s-node1
  Normal   Pulled     97s                kubelet            Successfully pulled
image "busybox" in 15.98593525s
  Warning  Unhealthy  55s (x3 over 75s)  kubelet            Liveness probe
failed: cat: can't open '/tmp/healthy': No such file or directory
  Normal   Killing    55s                kubelet            Container liveness
failed liveness probe, will be restarted
  Normal   Pulling    26s (x2 over 113s)  kubelet            Pulling image
"busybox"
  Normal   Created    0s (x2 over 97s)    kubelet            Created container
liveness
  Normal   Started    0s (x2 over 96s)    kubelet            Started container
liveness
```

```
  Normal   Pulled     0s                  kubelet            Successfully pulled
image "busybox" in 26.131183161s
```

说明：在pod运行后，将创建的/tmp/health文件10s后删除，LivenessProbe探针健康检查探测时间是15s，检查结果`Container liveness failed liveness probe`，然后容器会重启。