前面我们的示例中都是单个 Pod 的介绍，但是实际在生产应用中并不推荐直接使用 Pod 对象来部署应用，而是使用控制器来管理我们的应用，常见的控制有：ReplicationController 、ReplicaSet 和 Deployment 等。这章我们介绍一下 ReplicationController 和 ReplicaSet。

1. ReplicationController

ReplicationController 确保集群内任何时刻都有指定的 Pod 副本处于运行状态，这样我们就能保证应用的高可用。

1.1 创建

下面的示例是使用 ReplicationController 运行 nginx web 服务器的三副本。

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

上面的 yaml 文件中有几个重要的地方：

- kind: 指定为 ReplicationController；
- metadata: 指定该 ReplicationController 的一些元信息，比如 name 等；
- spec：该 ReplicationController 的核心定义，包括几个部分：
  - replicas：定义要运行的 Pod 的副本数；
  - selector：控制器用来筛选 Pod，需要和下面的 Pod template 定义中的 metadata 信息一致；

- template：该 ReplicationController 管理的 Pod 的定义模板，下面定义和 Pod 的定义一致。

1.2 使用

通过 `kubectl apply` 创建 ReplicationController 对象，下面命令中的 imooc 是我自己创建的用来跑 demo 的 namespace，大家实践的使用可以替换成自己的 namespace 名字。

```
kubectl apply -f nginx-rc.yaml -n imooc
```

然后我们可以通过 `kubectl describe rc` 来查看该 ReplicationController，这里的 rc 就是 ReplicationController 的简写。

```
$ kubectl describe rc nginx -n imooc
Name:         nginx
Namespace:    imooc
Selector:     app=nginx
Labels:       app=nginx
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"ReplicationController","metadata":
{"annotations":{},"name":"nginx","namespace":"imooc"},"spec":{"replicas":3,"s...
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:      nginx
    Port:       80/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
  Normal  SuccessfulCreate  3m43s  replication-controller  Created pod: nginx-
q5pdl
  Normal  SuccessfulCreate  3m43s  replication-controller  Created pod: nginx-
2qkhd
  Normal  SuccessfulCreate  3m43s  replication-controller  Created pod: nginx-
xggxr
```

上面的输出中有几点值得关注的是：

- Replicas：表示当前集群中有几个 Pod 副本在运行，`3 current / 3 desired` 表示期望运行的 Pod 是 3 个，当前运行的 Pod 也是 3 个，也就是正常；
- Pods Status：表示集群中运行的 Pod 的状态。`3 Running / 0 Waiting / 0 Succeeded / 0 Failed` 表示目前有 3 个 Pod 处于 Running 状态；
- Events：events 在 Kubernetes 是非常重要的信息，基本每个 Kubernetes 对象都有 events 字段。我们看到这里的 events 创建了三个以 nginx 作为前缀命名的 Pod。

我们可以通过 `kubectl get pods` 来查看一下这个 ReplicationController 创建出来的 pod。

```
$ kubectl get pods -n imooc | grep nginx
nginx-2qkhd           1/1     Running   0          1
nginx-q5pdl           1/1     Running   0          4h30m
nginx-xggxr           1/1     Running   0          4h30m
```

下面我们展示一下 ReplicationController 自动拉起的功能。首先将上面的三个 Pod 任选一个删除，比如第一个。

```
$ kubectl delete po nginx-2qkhd -n imooc
pod "nginx-2qkhd" deleted
```

然后我们再通过 `kubectl get pods` 来查看一下运行状态的 Pod。

```
➜  rc kubectl get pods -n imooc | grep nginx
nginx-lctmn           1/1     Running   0          54s
nginx-q5pdl           1/1     Running   0          7h29m
nginx-xggxr           1/1     Running   0          7h29m
```

我们再查看一下 ReplicationController 的 events 情况。会发现多了一条如下的记录：创建新的 Pod。这就是 ReplicationController 对失败 Pod 自动拉起的直观展示。

```
Events:
  Type    Reason            Age    From                    Message
  ----    ------            ----   ----                    -------
  Normal  SuccessfulCreate  2m12s  replication-controller  Created pod: nginx-
lctmn
```

1.3 删除 ReplicationController

Kubernetes 中所有的对象删除操作都可以通过 `kubectl delete` 来操作，对于 ReplicationController 也是一样。如下显示，我们删除完之后再通过 `kubectl get` 已经查不到了。

```
$ kubectl delete rc nginx -n imooc
replicationcontroller "nginx" deleted
$ kubectl get rc  -n imooc
No resources found in imooc namespace.
```

这里有个问题：删除完 ReplicationController 之后，之前通过 ReplicationController 启动的 Pod 会被删除吗？答案是会的。如果只想删除 ReplicationController，而不想删除 Pod，可以在 `kubectl delete` 的参数上增加选项 `--cascade=false`。

1.4 工作原理

ReplicationController 会监听 Kubernetes 集群内运行的 Pod 的个数，如果多于指定的副本数，则删除多余的 Pod；如果少于指定的副本数，则启动缺少的 Pod。这里所有的删除和启动操作都是通过 ReplicationController 来自动完成的。与手动创建的 Pod 不同的是，由 ReplicationController 创建的 Pod 在失败、删除或者终止时会被自动替换，比如在系统升级之后，节点上的 Pod 会被重建。

在使用中，即使应用程序只需要一个 Pod，也建议使用 ReplicationController 来创建 Pod。可以将 ReplicationController 类比成 supervisor 这种进程管理器，不同在于 ReplicationController 不是管理单个进程，而是监控管理集群内部跨多个节点的多个 Pod。

2. ReplicaSet

ReplicaSet 可以认为是下一代 Replication Controller，改进的地方主要在于选择器的支持上。ReplicaSet 支持新的基于集合的选择器，其实不光是 ReplicaSet，像 Deployment，DaemonSet 等都支持，只能说 ReplicationController 是老一代的失败产物。关于选择器，我们这里举个例子。

ReplicationController 支持的选择器类似下面。

而 ReplicaSet 支持的选择器则更加的灵活，如下所示 matchExpressions 提供了一种基于集合的选择器，包括 `In` 和 `NotIn`。

```
selector:
  matchLabels:
    component: redis
  matchExpressions:
    - {key: tier, operator: In, values: [cache]}
    - {key: environment, operator: NotIn, values: [dev]}
```

2.1 创建 ReplicaSet

和 Replication Controller 类似，我们创建 ReplicaSet 也只需要编写一个 ReplicaSet 对象的 yaml 文件即可，下面是一个示例。

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
```

```
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 100m
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
```

我们可以看到 ReplicaSet 的 spec 和 Replication Controller 非常的类似，主要包括几个域：

- kind：指定为 ReplicaSet；
- .spec.replicas：同时运行的 Pod 的副本个数；
- .spec.selector：选择器。ReplicaSet 管理所有标签匹配与标签选择器的 Pod。它不区分自己创建或删除的 Pod 和其他人或进程创建或删除的 pod。

  这允许在不影响运行中的 Pod 的情况下替换副本集。需要注意的是 `.spec.template.metadata.labels` 必须匹配 `.spec.selector`，否则 Kubernetes 则认为是非法的；
- .spec.template：Pod 模板。

2.2 使用

我们仍然可以通过 `kubectl apply` 来创建 ReplicaSet 。

```
$ kubectl apply -f nginx-rs.yaml -n imooc
replicaset.apps/frontend created
```

然后我们通过 `kubectl describe` 来查看我们创建出来的 ReplicaSet，rs 是 ReplicaSet 的缩写。

```
$ kubectl describe rs frontend -n imooc
Name:         frontend
Namespace:    imooc
Selector:     tier=frontend
Labels:       app=guestbook
              tier=frontend
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
```

```
                {"apiVersion":"apps/v1","kind":"ReplicaSet","metadata":
{"annotations":{},"labels":
{"app":"guestbook","tier":"frontend"},"name":"frontend",...
Replicas:     3 current / 3 desired
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  tier=frontend
  Containers:
   nginx:
    Image:      nginx
    Port:       80/TCP
    Host Port:  0/TCP
    Limits:
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  99s   replicaset-controller  Created pod: frontend-
9klqt
  Normal  SuccessfulCreate  99s   replicaset-controller  Created pod: frontend-
tr696
  Normal  SuccessfulCreate  99s   replicaset-controller  Created pod: frontend-
7h6rq
```

我们可以看到通过 `kubectl describe` 显示出来的 ReplicaSet 的详情和前面的 Replication Controller 非常的类似。在 events 的列表里面我们也可以看到创建了三个 Pod 的事件。我们查看一下 Pod 情况。

```
$ kubectl get pods -n imooc | grep frontend
frontend-7h6rq        1/1     Running   0          5m4s
frontend-9klqt        1/1     Running   0          5m4s
frontend-tr696        1/1     Running   0          5m4s
```

我们可以将第一个 Pod 删除，然后查看有没有新的 Pod 被创建出来，同时查看 ReplicaSet 的 event 事件中有没有类似上面 Replication Controller 中新建 Pod 的 event。

```
$ kubectl delete pods frontend-7h6rq -n imooc
pod "frontend-7h6rq" deleted
$ kubectl get pods -n imooc | grep frontend
frontend-9klqt        1/1     Running   0          9m34s
frontend-n9vsb        1/1     Running   0          79s
frontend-tr696        1/1     Running   0          9m34s
```

ReplicaSet 中新增的 event 如下。

```
Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  10m   replicaset-controller  Created pod: frontend-
9klqt
```

2.3 删除

如果我们要同时删除 ReplicaSet 和 Pod，则直接使用 `kubectl delete` 删除即可。

```
$ kubectl delete rs frontend -n imooc
replicaset.extensions "frontend" deleted
```

如果我们只想删除 ReplicaSet，但是想保留 Pod，则将 `kubectl delete` 命令后面添加参数 `--cascade=false` 即可。

2.4 ReplicaSet 高级使用

我们可以通过修改 ReplicaSet 中的 `.spec.replicas` 字段来实现运行的 Pod 个数伸缩限制。我们更新完 yaml 文件之后直接使用 `kubectl apply` 重新应用一下即可。

ReplicaSet 还可以结合 HorizontalPodAutoscaler（中文可以叫作水平 Pod 缩放器，可以简写为 HPA）来使用。HAP 可以基于 CPU 利用率自动伸缩 replication controller、deployment 和 replica set 中的 pod 数量，（除了 CPU 利用率）也可以 基于其他应程序提供的度量指标 custom metrics。下面就是基于 CPU 使用来做弹性伸缩的示例，当 CPU 使用率达到 50 时则进行自动伸缩。

```
controllers/hpa-rs.yaml

apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: frontend-scaler
spec:
  scaleTargetRef:
    kind: ReplicaSet
    name: frontend
  minReplicas: 3
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

3. 总结

本文介绍了 Kubernetes 中两种基本的多副本控制器：Replication Controller 和 ReplicaSet。实际上这两种多副本控制器都不推荐使用了，而是使用 Deployment 来替代，但是 Deployment 实际上却是通过委托 ReplicaSet 来实现的，所以这一章相当于 Deployment 的背景知识，下面一章我们重点介绍 Deployment。

}

一手微信itit11223344