POD详解二

restartPolicy

K8S在设计之初就是要做一套完善的平台。 所以针对其中的服务肯定是要支持高可用，灾难恢复，负载均衡等功能。
而K8S也确实提供了诸如Deployment，Service等资源对象来协助POD完成这些设计。
但是也需要POD本身拥有一些机制。restartPolicy就是其中之一。这个功能就是 Kubernetes 里的Pod 恢复机制。它是 Pod 的 Spec 部分的一个标准字段，默认值是Always，意思是不论任何时候POD出现了异常，都一定会重新创建。当然用户还可以手动指定这个重启的策略。 它有3个值。

- Always： 不论任何时候，只要k8s检测到POD不在运行状态，都会尝试重新创建它。
- OnFailure：只要在POD中的容器出现异常时才重启容器， 如果容器是处于complete状态，也就是正常退出的状态，是不会重启的
- Nerver：从来不重启容器。

注意：如果POD中存在多个容器(不包括pause容器)，那么只有所有容器都都进入异常状态后，POD的状态才会是Failure，否则仍然是running状态。

我们需要根据自己的业务形态来合理的设置这个值，
如果我们的容器是一个在线的服务，那么可能设置为默认的Always是比较合理的，因为我们期望它能够在异常的时候自动重启来保持服务状态， 当然如果我们是在测试环境中，可能希望这个值是Nerver比较好， 因为有些时候我们希望能保留现场。
而如果是离线的task，比如运行一个测试任务，那么设置为OnFailure是比较合理的，也就是只有在出现异常的时候才会重跑任务，而程序正常退出处于complete状态的时候，是不会触发重启策略的。

探针

拥有了restartPolicy这种在出现异常时的恢复机制， 我们就为高可用打下了第一个基础。 那么接下来我们就要考虑另外一件事了，
就是我们如何发现POD出现了异常呢？
如果是一些比较严重的异常，比如OOM或者容器内1号进程异常退出。面对这种异常我们是不需要增加任何机制k8s就能检测到并根据restart Policy进行重启。
但是我们面对的大多数情况都是k8s无法自动检测出的异常。比如容器内进程仍然存在，但是已经因为一些异常无法对外正常提供服务了。
所以为了解决这种健康检查问题，k8s为我们提供了两种探针：livenessProbe和readnessProbe

- readnessProbe：用来探后容器启动后服务是否达到ready状态，k8s一旦探测到容器中的服务在启动后达到了可以对外服务的状态，就会将POD设置为Running状态，从此这个探针就会失效。
  而如果在探针有效期内容器中的服务一直没有处于ready状态，就会把POD标记为异常状态并触发POD的restartPolicy
- livenessProbe：在容器启动后，周期性的持续对容器中的服务进行探活，如果发现容器内的服务处于异常状态，就会将POD标记为异常状态并触发POD的restartPolicy

下面我们看一个例子：

```
livenessProbe:
failureThreshold: 3
periodSeconds: 5
httpGet:
path: /grid/console
port: 4444
initialDelaySeconds: 30
timeoutSeconds: 5
readinessProbe:
failureThreshold: 3
periodSeconds: 5
httpGet:
path: /grid/console
port: 4444
initialDelaySeconds: 10
timeoutSeconds: 5
```

上面是我使用k8s部署selenium grid时针对hub pod做的健康检查。我们来看看这两个探针是怎么玩的。 探针的方式有数种：

- ExecAction：在container中执行指定的命令。当其执行成功时，将其退出码设置为0；
- TCPSocketAction：执行一个TCP检查使用container的IP地址和指定的端口作为socket。如果端口处于打开状态视为成功；
- HTTPGetAction：执行一个HTTP默认请求使用container的IP地址和指定的端口以及请求的路径作为url，用户可以通过host参数设置请求的地址，通过scheme参数设置协议类型(HTTP、HTTPS)如果其响应代码在200~400之间，设为成功。

而我使用的是httpGet的方式，用来探测grid hub 服务的健康状态。 我们看这两个探针的使用方式几乎是一样的， 连字段都一模一样。
不同的是他们的目标。 livenessProbe是在容器运行时周期性的对服务进行探活，是一直不停的。
而readinessProbe只是用来探测容器中的服务是否启动成功， 一旦成功后就会失效。
他们分别再之后要说的关于Service的负载均衡策略中分别扮演了不同的角色。 这个之后我们再说。接下来我们看看在探针中常用的字段。

- initialDelaySeconds：探针的初次探活的延迟时间，
  也就是说在容器启动后多长的延迟时间后，探针才开始运作。很多服务的启动时间很长，所以需要设置这个参数来预留足够的时间，避免服务因为启动时间过长， 但是探针已经开始运作而导致了探针的失败。
- failureThreshold：探测时连续失败几次视为探测异常。

这个算是一个重试机制，有些时候因为网络卡顿或者一些暂时性的原因，导致探测失败。
那么我们可以设置这个重试机制来增加容错性

- periodSeconds：每一次探活的间隔时间
- timeoutSeconds：探活时的超时时间