19.什么是存活探针

> 从前面的学习我们可以了解到，Pod代表了Kubernetes中的基本部署单位。我们知道了如何手动对Pod进行创建、监管。但是在实战中，我们希望部署的应用能够自动地保持正常运行，而且在不需要任何手动干预的情况下保持健康状态。为了达到这个目的，我们几乎不会直接创建Pod，而是创建其他类型的资源，如ReplicationController或者Deployment，由它们来创建和管理实际的Pod。

使用Kubernetes的主要好处之一就是，给它提供一份容器清单，它就能够保证这些容器在集群中某处运行。我们创建一个Pod资源，然后让Kubernetes为其挑选一个工作节点并在该节点上运行pod的容器。但是，如果其中某个容器挂掉了怎么办？如果一个pod中所有容器挂掉了怎么办?
一旦Pod被调度到了某个节点上，该节点上的Kubelet程序就会运行Pod中的容器，从此只要该Pod存在就会保持容器运行。如果容器的主进程崩了，Kubelet就会重启容器。如果我们的应用程序有一个bug导致其每隔一段时间就崩一次，Kubernetes就会自动重启应用程序，所以即使app本身没有做任何处理，在Kubernetes中运行app使其具有了自愈的能力。
但是有时候即使进程没有崩掉，app也会停止工作。例如Java应用在内存泄漏的时候会抛出OutOfMemboryErrors异常，但是JVM进程仍然保持运行。如果有一个方式让app能够通知Kubernetes，表明其不再正常运行，让Kubernetes对其进行重启，那就太好了。
一个崩掉的容器会自动重启，因此，你也许会认为可以在app中捕获这些类错误，然后当这些错误出现的时候就退出进程。当然你可以这样做，但是这并不能完全解决你所有问题。
例如，当你的app由于陷入了死循环或者死锁而停止响应时，你该怎么办？为了确保应用程序在出现这类情况时可以重新启动，我们必须从外部检查app的健康状态，而不是依赖app内部去做这种事情。

1.什么是存活探针

Kubernetes通过存活探针（liveness probe）能够检查一个容器是否仍然处于存活状态。我们可以在Pod的spec区段中为其指定一个存活探针。Kubernetes会定期地执行探针，如果探测失败会自动重启容器。

Kubernetes有三种可以探测容器的机制：

- HTTP GET探针，对容器IP地址、端口和指定的路径执行HTTP GET请求。如果探针收到响应消息并且响应码不表示错误，也就是说，响应码是2xx或者3xx，则认为探测成功。如果服务器返回了一个错误响应码或者根本就没有响应，则探测失败，容器就会被重启。
- TCP Soket探针，尝试与容器指定端口建立TCP连接。连接一旦成功建立，则探测成功，否则探测失败，容器被重新启动。
- Exec探针，在容器中执行任意的命令，然后检查命令的退出状态码。如果状态码是0，则探测成功。其他任何状态码都被认为探测失败。

2.创建HTTP GET存活探针

现在让我们来看一下如何为之前创建的Node.js程序添加一个存活探针。由于是web应用程序，因此添加一个探测web服务器是否服务请求的探针比较合理。但是由于该程序太简单了，所以需要人为地让该程序失败。

我们对这个程序稍作修改，在第5个请求之后，为每个请求返回一个500 Internal Server Error状态码。该程序会正确地处理前五个客户端请求，后续的请求都会返回错误码。由于添加了存活探针 ，在返回错误后，应用应该会被重新启动，使其能够再次正确地处理客户端请求。

修改我们在第7节中创建的app.js程序：

```
const http = require('http');
const os = require('os');
console.log("开始运行...");

var requestCount = 0;

var handler = function(request, response) {
 console.log("收到来自 " + request.connection.remoteAddress + "的消息");
 requestCount++;
 if (requestCount > 5) {
    response.writeHead(500);
    response.end("我出问题了，请重启我吧!");
    return;
  }
 response.writeHead(200);
 response.end("已发送消息至：  " + os.hostname() + "\n");
```

};
var www = http.createServer(handler);
www.listen(8080);

```
const http = require('http');
const os = require('os');
console.log("开始运行...");
var requestCount = 0;
var handler = function(request, response) {
 console.log("收到来自 " + request.connection.remoteAddress + "的消息");
 requestCount++;
 if (requestCount > 5) {
    response.writeHead(500);
    response.end("我出问题了，请重启我吧!");
    return;
  }
 response.writeHead(200);
 response.end("已发送消息至： " + os.hostname() + "\n");
};
var www = http.createServer(handler);
www.listen(8080);
```

然后创建一个新的镜像registry.cn-shanghai.aliyuncs.com/david-ns01/test2:1.0，并将其发布到阿里云镜像仓库（参考第7节和第8节）。
现在我们来创建一个新的包含HTTP GET探针的Pod。如下为该Pod的YAML文件内容：
vim test2-liveness.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: test2-liveness
spec:
  containers:
  - image: registry.cn-shanghai.aliyuncs.com/david-ns01/test2:1.0
    name: test2
    livenessProbe:
      httpGet:
        path: /
        port: 8080
```

apiVersion: v1
kind: Pod
metadata:
  name: test2-liveness
spec:
  containers:
  - image: registry.cn-shanghai.aliyuncs.com/david-ns01/test2:1.0
    name: test2
    livenessProbe:
      httpGet:
        path: /
        port: 8080

该文件中指定了一个httpGet存活探针，它会告诉Kubernetes在路径/和端口8080上定期地执行HTTP GET请求，以确定该容器是否健康。容器一旦启动，这些请求就会立即开始。

经过5次这样的请求（或者实际的客户端请求）后，我们的应用程序就会开始返回HTTP 500状态码，因此Kubernetes会认为探测失败并重新启动容器。

3.使用存活探针

要想知道存活探针干了什么，现在就尝试创建这个Pod：

kubectl create –f test2–liveness.yaml

```
[david@dhr-demo ~]$ kubectl create -f test2-liveness.yaml
pod/test2-liveness created
[david@dhr-demo ~]$
```

过一会儿之后，容器就会被重启。可以执行如下命令查看pod的状态：

kubectl get po test2–liveness

```
[david@dhr-demo ~]$ kubectl get po test2-liveness
NAME             READY   STATUS    RESTARTS   AGE
test2-liveness   1/1     Running   4          8m3s
[david@dhr-demo ~]$
```

RESTARTS列显示了pod容器重启的次数，该次数随着时间的推移不断递增:

```
[david@dhr-demo ~]$ kubectl get po test2-liveness
NAME             READY   STATUS             RESTARTS   AGE
test2-liveness   0/1     CrashLoopBackOff   6          15m
[david@dhr-demo ~]$ kubectl get po test2-liveness
NAME             READY   STATUS             RESTARTS   AGE
test2-liveness   0/1     CrashLoopBackOff   6          15m
```

```
[david@dhr-demo ~]$ kubectl get po test2-liveness
NAME             READY   STATUS    RESTARTS   AGE
test2-liveness   1/1     Running   7          16m
[david@dhr-demo ~]$
```

我们知道通过kubectl logs可以获取容器应用日志。如果容器重启了，kubectl logs命令会显示当前容器的日志。

如果想知道前一个容器为什么终止了，可以指定––previous选项：

kubectl logs mypod ––previous

通过执行kubectl describe命令，我们可以看到容器为什么会被重启：

kubectl describe po test2–liveness

```
[david@dhr-demo ~]$ kubectl describe po test2-liveness
Name:         test2-liveness
Namespace:    default
Priority:     0
Node:         minikube/172.17.0.3
Start Time:   Sat, 21 Nov 2020 17:49:24 +0800
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           172.18.0.3
IPs:
  IP:  172.18.0.3
Containers:
  test2:
    Container ID:   docker://f3c7f903cac0da8b2e8dc614910c473edaafb6aabb444a9bb163a15279c8024c
    Image:          registry.cn-shanghai.aliyuncs.com/david-ns01/test2:1.0
    Image ID:       docker-pullable://registry.cn-shanghai.aliyuncs.com/david-ns01/test2@sha256:d6cca8aaea8270d9acf502
    Port:           <none>
    Host Port:      <none>
    State:          Running                                          当前容器的状态
      Started:      Sat, 21 Nov 2020 18:05:03 +0800
    Last State:     Terminated
      Reason:       Error                                            上一个容器的状态等信息
      Exit Code:    137
      Started:      Sat, 21 Nov 2020 18:00:19 +0800
      Finished:     Sat, 21 Nov 2020 18:02:09 +0800
    Ready:          True
    Restart Count:  7                                                容器被重启的次数
    Liveness:       http-get http://:8080/ delay=0s timeout=1s period=10s #success=1 #failure=3
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-fnpcr (ro)
Conditions:
  Type              Status
```

可以看到容器当前正在运行，但是先前因为错误被终止了。Exit Code137有一个特殊的含义，它表示该进程由外部信号终止。137是两个数字之和：128+x，x就是发送给进程使其终止的信号编号。这个例子中，x=9，这是SIGKILL的信号编号，表示进程是被强制终止的。

```
              node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type     Reason     Age                  From               Message
  ----     ------     ----                 ----               -------
  Normal   Scheduled  32m                  default-scheduler  Successfully assigned default/test2-liveness to minikube
  Normal   Pulling    32m                  kubelet, minikube  Pulling image "registry.cn-shanghai.aliyuncs.com/david-ns01/te
  Normal   Pulled     32m                  kubelet, minikube  Successfully pulled image "registry.cn-shanghai.aliyuncs.com/d
  Normal   Killing    27m (x3 over 31m)    kubelet, minikube  Container test2 failed liveness probe, will be restarted
  Normal   Created    27m (x4 over 32m)    kubelet, minikube  Created container test2
  Normal   Started    27m (x4 over 32m)    kubelet, minikube  Started container test2
  Normal   Pulled     27m (x3 over 30m)    kubelet, minikube  Container image "registry.cn-shanghai.aliyuncs.com/david-ns01/
  Warning  Unhealthy  7m16s (x28 over 31m)  kubelet, minikube  Liveness probe failed: HTTP probe failed with statuscode: 500
  Warning  BackOff    2m38s (x60 over 19m)  kubelet, minikube  Back-off restarting failed container
[david@dhr-demo ~]$ kubectl get po test2-liveness
```

kubectl describe po 输出内容的底部包含Events区段，它显示了容器被终止的原因是Kubernetes探测到容器不健康，因此终止并重新创建容器。

需要注意的是，当容器被终止后，Kubernetes会创建一个全新的容器，而不是重启原来的容器。

4.配置存活探针的附加属性

可能你已经注意到了，kubectl describe命令还显示了关于存活探针的附加信息：

```
[david@dhr-demo ~]$ kubectl describe po test2-liveness
Name:         test2-liveness
Namespace:    default
Priority:     0
Node:         minikube/172.17.0.3
Start Time:   Sat, 21 Nov 2020 17:49:24 +0800
Labels:       <none>
Annotations:  <none>
Status:       Running
IP:           172.18.0.3
IPs:
  IP:  172.18.0.3
Containers:
  test2:
    Container ID:   docker://4180446f9e2db73399ee2a00842e09c9695457af509301acc8894a381527a0e6
    Image:          registry.cn-shanghai.aliyuncs.com/david-ns01/test2:1.0
    Image ID:       docker-pullable://registry.cn-shanghai.aliyuncs.com/david-ns01/test2@sha256:d6cca8aaea8270d9ac
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sat, 21 Nov 2020 18:20:44 +0800
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      Sat, 21 Nov 2020 18:13:49 +0800
      Finished:     Sat, 21 Nov 2020 18:15:39 +0800
    Ready:          True
    Restart Count:  10
    Liveness:       http-get http://:8080/ delay=0s timeout=1s period=10s #success=1 #failure=3
    Environment:    <none>
    Mounts:
```

除了我们显示地指定存活探针选项，还可以看到一些附加的属性，比如delay、timeout、period等等。

delay=0s表示容器启动后就立即开始探测。

timeout=1s表示容器必须在1s内做出响应，否则这次探测被认为失败。

period=10s表示每隔10s对容器进行一次探测。

#failure=3表示在连续三次探测失败后重新启动容器。

#success=1表示探针在探测失败后，被视为成功的最小连续成功数为1。

在指定探针时可以对这些附加参数进行定制化。例如，在存活探针配置中添加initialDelaySeconds属性设置初始延迟值：

```
apiVersion: v1
kind: Pod
metadata:
  name: test2-liveness
spec:
  containers:
  - image: registry.cn-shanghai.aliyuncs.com/david-ns01/test2:1.0
    name: test2
    livenessProbe:
      httpGet:
        path: /
        port: 8080
     initialDelaySeconds: 15
```

initialDelaySeconds属性指示Kubernetes在执行第一次探测之前等待的时间为15s。
如果不指定初始延迟值，探针就会在容器一启动后就开始探测，这通常会导致探测失败，因为app还未准备好开始接受请求。如果失败的次数超过failure指定的阈值，容器就会被重启，直到它可以开始正常响应请求。

> 需要注意的是，设置的初始延迟值一定要考虑到应用程序的启动时间

很多时候，用户可能会疑惑为什么他们的容器正在重启。如果他们执行kubectl describe就会看到容器被终止了且状态码为137或者143，告知他们该Pod是由外部信号终止的。除此之外，Pod的事件列表里会显示容器由于存活探测失败而被终止。如果在启动的时候出现了这样的情况，那应该就是你没有合适地设置initialDelaySeconds的值。

> 退出码（Exit Code）137表示进程被外部信号终止。同样的，退出码143表示进程是被SIGTERM信号终止的（143=128+15，15代表SIGTERM）

5.创建有效的存活探针

对于运行在生产环境中的Pod，我们应该总是定义一个存活探针。如果没有的话，Kubernetes就没法知道应用程序是否存活。只要进程正在运行，Kubernetes就会认为容器是健康的。

存活探针应该检查什么

我们上面指定的存活探针简单地检查服务器是否响应。这虽然看起来过度简单，但即使是这样的存活探针也可以创造奇迹，因为如果运行在容器中的web服务器停止响应HTTP请求，它将使容器重新启动。与没有存活探针相比，这是一个很大的进步，而且在大多数情况下可能已经足够了。
但是为了更好地进行存活检查，我们最好配置探针在特定的URL路径（如/health）上执行请求，并且让app对内部运行的所有关键组件执行一个内部状态检查，以确保它们中任何一个都没有终止或停止响应。

> 需要确保/health HTTP端点不需要认证，否则探针就会失败，导致容器无限地重启。

请确保只检查应用程序的内部，不要受外部因素的干扰。例如，当一个前端web服务器无法连接到后端数据库时，针对该服务器的存活探针不应该返回失败。如果问题的底层原因是在数据库本身，重启web服务器的容器不会解决问题。此种情况下，由于存活探针会再次失败，你将面对的是容器不断地重启，直到数据库恢复。

轻量级的存活探针

存活探针不应该使用太多的计算资源，而且不应该花太长时间来完成探测。默认情况下，探针执行相对比较频繁，而且只允许在一秒内完成。让探针执行过重的任务会大大地减慢容器运行。后面我们会了解到如何限制一个容器可用的CPU时间。探针的CPU时间会计入容器的CPU时间配额，因此配备一个重量级的存活探针会减少主应用程序进程可用的CPU时间。

> 如果在容器中运行Java程序，一定要确保使用的是HTTP GET存活探针，而不是Exec探针，因为它会启动一个全新的JVM来获取存活信息。这同样也适用于任何基于JVM或者类似的应用，它们的启动过程需要花费大量的计算资源。

不必在探针中实现重试机制

我们已经看到探针的失败阈值是可以配置的，而且通常情况下，必须在探针失败多次后，容器才会重启。但即使你将失败阈值设置成1，Kubernetes也会在多次重试探针后才会把它当做一次失败的尝试。因此在探针中实现你自己的重试机制显然是多此一举。

总结

现在我们知道，Kubernetes是通过在容器崩掉或者容器存活探测失败后对其进行重启的方式来保持容器运行的。这个工作是由Pod所在节点上的Kubelet执行的。运行在master节点上的所有控制面板组件没有参与到这个过程中。

但是如果节点（Node）崩掉了，就需要控制面板来创建随节点一起崩掉的所有Pod的替代品了。不过对于我们直接创建的Pod，它不会执行这样的操作。这些Pod只由Kubelet管理，但是由于Kubelet本身运行在这个节点上，如果节点挂了，它将无法执行任何操作。

如果想确保我们的应用程序在另外一个节点上运行，就需要使用ReplicationController或者类似的机制来管理Pod。我们将在后面的章节中讨论。