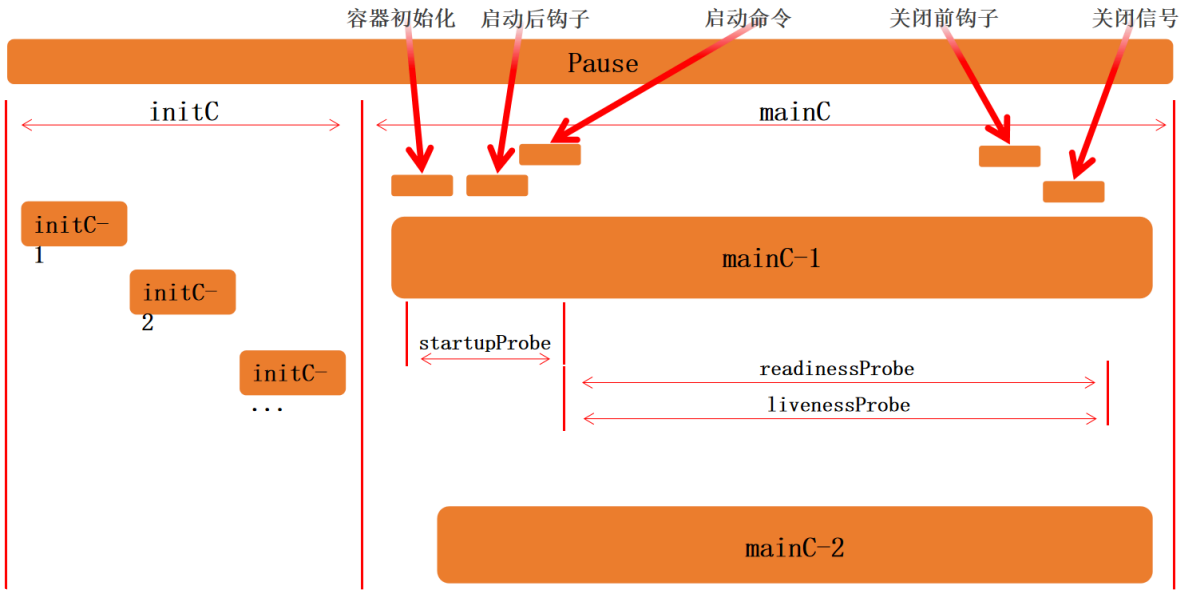
Chapter4.2 Pod's lifecycle

Pod 从创建到死亡的所有流程：



启动前钩子、启动后钩子、启动探测 startupProbe、就绪探测 readinessProbe、存活探测 livenessProvebe 都是由当前 Pod 所在节点 node 所在的 kubelet 去执行

一、initContainers

init 容器与普通的容器非常像，除了如下两点：

- init 容器总是运行到成功完成为止，如果 Pod 的 Init 容器失败，Kubernetes 会不断地重启该 Pod，直到 Init 容器成功为止。然而，如果 Pod 对应的 restartPolicy 为 Never，它不会重新启动
- 每个 init 容器都必须在下一个 init 容器启动之前成功完成（串行，防止某个 init 容器依赖于前面的 init 容器成功运行）

1. init 容器的作用：

Initc 与应用容器具备不同的镜像，可以把一些危险的工具放置在 initc 中，进行使用

initC 多个之间是线性启动的，所以可以做一些延迟性的操作

initC 无法定义 readinessProbe，其它以外同应用容器定义无异

2. init 容器实验

2.1 实验 1

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: initc-1
  labels:
    app: initc
spec:
  containers:
    - name: myapp-container
      image: wangyanglinux/tools:busybox
      command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
    - name: init-myservice
      image: wangyanglinux/tools:busybox
      command: ['sh', '-c', 'until nslookup myservice; do echo waitin
g for myservice; sleep 2; done;']
    - name: init-mydb
      image: wangyanglinux/tools:busybox
      command: ['sh', '-c', 'until nslookup mydb; do echo waiting for
mydb; sleep 2; done;']
```

运行机制：

- Kubernetes 会按顺序启动 `initContainers`。只有当所有初始化容器完成运行后，主容器 `myapp-container` 才会启动。
- 初始化容器主要用于检查依赖的服务（myservice 和 mydb）是否已启动，确保主容器运行环境稳定。

实验步骤：

YAML

```
[root@k8s-master01 4]# kubectl get svc
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.0.0.1       <none>        443/TCP    5d1h
myapp        ClusterIP   10.0.223.76    <none>        80/TCP     25h
mydb         ClusterIP   10.6.89.152    <none>        80/TCP     23h
myservice    ClusterIP   10.11.142.36   <none>        80/TCP     23h
[root@k8s-master01 4]# kubectl delete svc myservice
service "myservice" deleted
[root@k8s-master01 4]# kubectl delete svc mydb
service "mydb" deleted
[root@k8s-master01 4]# kubectl create -f init1.pod.yaml
pod/initc-1 created
```

查看 pod 状态可以发现 Init:0/2，因为前面我们将 mservice 和 mydb 这两个 service 都删除了

YAML

```
[root@k8s-master01 4]# kubectl get pod
NAME                    READY   STATUS     RESTARTS          AGE
initc-1                 0/1     Init:0/2   0                 20s
```

而当创建了 myservice 后，Init 将变成 1/2 的状态

Bash

```
kubectl create svc clusterip myservice --tcp=80:80
```

```
[root@k8s-master01 4]# kubectl get pod
NAME                    READY   STATUS     RESTARTS          AGE
initc-1                 0/1     Init:1/2   0                 2m20s
```

而当创建了 mydb 后，pod 就会进入初始化状态

Bash

```
kubectl create svc clusterip mydb --tcp=80:80
```

```
[root@k8s-master01 4]# kubectl get pod
NAME                    READY   STATUS            RESTARTS          AGE
initc-1                 0/1     PodInitializing   0                 2m53s
```

再过一会儿就可以成功 READY

```
[root@k8s-master01 4]# kubectl get pod
NAME                    READY   STATUS    RESTARTS          AGE
initc-1                 1/1     Running   0                 3m15s
```

2.2 实验 2

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod-0
  labels:
    app: myapp
spec:
  containers:
    - name: myapp
      image: wangyanglinux/myapp:v1.0
  initContainers:
    - name: randexit
      image: wangyanglinux/tools:randexitv1
      args: ["--exitcode=1"]
```

运行机制：

- **初始化容器：**
  - Kubernetes 按顺序执行 `initContainers`。如果其中一个容器返回非零退出码（例如 `1`），整个 Pod 会停止初始化并不断重试。
  - 在此配置中，`randexit` 可能会返回退出码 `1`，从而导致初始化失败，Pod 无法进入 `Running` 状态。
- **主容器：**
  - 只有当所有 `initContainers` 成功运行后，主容器 `myapp` 才会启动。

实验步骤：

```
Bash

[root@k8s-master01 4]# kubectl create -f randexit.pod.yaml
pod/myapp-pod-0 created
[root@k8s-master01 4]# kubectl get pod
NAME          READY   STATUS                  RESTARTS      AGE
myapp-pod-0   0/1     Init:CrashLoopBackOff   4 (72s ago)   3m13s
```

创建 pod 后，get pod 发现 pod 已经 Init:CrashLoopBackOff，因为 initContainers 的 exitcode 为 1，该 Init 容器没有成功退出

查看该 Init 容器的日志可以发现失败退出的日志：

```
Bash

[root@k8s-master01 4]# kubectl logs myapp-pod-0 -c randexit
休眠 4 秒，返回码为 1!
```

我们只能修改 yaml 文件中的 exitcode 为 0，删除当前 pod 后重新创建 pod，再次 get pod 可以发现创建成功

```
Bash

[root@k8s-master01 4]# kubectl get pod
NAME          READY   STATUS    RESTARTS   AGE
myapp-pod-0   1/1     Running   0          9s
```

二、探针

探针是由 kubelet 对容器执行的定期诊断。要执行诊断，kubelet 调用由容器实现的 Handler。有三种

类型的**处理程序**：

- ExecAction: 在容器内**执行指定命令**。如果命令退出时返回码为 0 则认为诊断成功
- TCPSocketAction: 对指定端口上的容器的 IP 地址进行 **TCP 检查**。如果端口打开，则诊断被认为是成功的
- HTTPGetAction: 对指定的端口和路径上的容器的 IP 地址执行 **HTTP Get 请求**。如果响应的状态

  码**大于等于 200 且小于 400**，则诊断被认为是成功的

每次探测都将获得以下三种结果之一：

成功：容器通过了诊断。

失败：容器未通过诊断。

未知：诊断失败，因此不会采取任何行动

1. 探针分类

1.1 startupProbe:

startupProbe 决定是否开始后续检测，只有当 startupProbe 探针探测通过，才会进行 livenessProbe 探针检测和 readinessProbe 探针检测，k8s 在 1.16 版本后增加 startupProbe 探针，主要解决在复杂的程序中 readinessProbe、ivenessProbe 探针无法更好的判断程序是否启动、是否存活。

启动探针保障存活探针在执行的时候不会因为时间设定问题导致无限死亡或者延迟很长的情况

- 成功：开始允许存活探测 就绪探测开始执行
- 失败：静默
- 未知：静默

startupProbe 选项说明：

- initialDelaySeconds: 容器启动后要等待多少秒后就探针开始工作，单位“秒”，默认是 0 秒，最小值是 0
- periodSeconds: 执行探测的时间间隔(单位是秒)，默认为 10s，单位“秒”，最小值是 1
- timeoutSeconds: 探针执行检测请求后，等待响应的超时时间，默认为 1s，单位“秒”，最小值是 1
- successThreshold: 探针检测失败后认为成功的最小连接成功次数，默认值为 1。必须为 1 才能激活和启动。最小值为 1。
- failureThreshold: 探测失败的重试次数，重试一定次数后将认为失败，默认值为 3，最小值为 1。

startupProbe 实验

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: startupprobe-1
  namespace: default
spec:
  containers:
  - name: myapp-container
    image: wangyanglinux/myapp:v1.0
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
      containerPort: 80
    readinessProbe:
      httpGet:
        path: /index2.html
        port: 80
      initialDelaySeconds: 1
      periodSeconds: 3
    startupProbe:
      httpGet:
        path: /index1.html
        port: 80
      failureThreshold: 30
      periodSeconds: 10
```

Ps：应用程序将会有最多 5 分钟 failureThreshold * periodSeconds（30 * 10 = 300s）的时间来完成其启动过程。

探针配置：

1. **`readinessProbe` （就绪探针）**：
   - 用于检查容器是否已准备好接收流量。
   - 配置：
     - `httpGet`：通过 HTTP GET 请求检查 `/index2.html` 页面是否返回 2xx 或 3xx 的状态码。
     - `initialDelaySeconds: 1`：容器启动后等待 1 秒开始检测。
     - `periodSeconds: 3`：每隔 3 秒执行一次检测。
2. **`startupProbe` （启动探针）**：
   - 用于检查容器启动是否完成，确保容器启动前不会因为 `readinessProbe` 或 `livenessProbe` 失败而被重启。

- 配置：
  - `httpGet`：通过 HTTP GET 请求检查 `/index1.html` 页面是否返回 2xx 或 3xx 的状态码。
  - `failureThreshold: 30`：允许最大失败次数为 30 次。
  - `periodSeconds: 10`：每隔 10 秒执行一次检测。

运行机制

- **启动阶段**：
  - `startupProbe` 先运行，检测 `/index1.html` 页面是否正常。
  - 如果检测失败的次数达到 `failureThreshold`（30 次），容器会被判定为启动失败。
- **就绪阶段**：
  - 当 `startupProbe` 检测通过后，`readinessProbe` 开始运行。
  - `readinessProbe` 确保 `/index2.html` 正常后，容器会被标记为 `Ready`，可以接收流量。

实验步骤：

YAML

```
[root@k8s-master01 6]# kubectl create -f startProbe.pod.yaml
pod/startupprobe-1 created
[root@k8s-master01 6]# kubectl get pod
NAME             READY   STATUS    RESTARTS   AGE
startupprobe-1   0/1     Running   0          119s
```

创建 pod 后可以发现 pod 一直处于 READY 0/1 状态，因为 **`startupProbe`** 没有检测到 `/index1.html`

YAML

```
[root@k8s-master01 6]# kubectl exec -it startupprobe-1 -- /bin/bash
startupprobe-1:/# cd /usr/local/nginx/html/
startupprobe-1:/usr/local/nginx/html# ls
50x.html       hostname.html  index.html
startupprobe-1:/usr/local/nginx/html# touch index1.html
```

进入容器，创建 index1.html 文件

再次 get pod 发现仍然处于处于 READY 0/1 状态，因为 `readinessProbe` 没有检测到 `/index2.html`

```yaml
[root@k8s-master01 6]# kubectl exec -it startupprobe-1 -- /bin/bash
startupprobe-1:/# cd /usr/local/nginx/html/
startupprobe-1:/usr/local/nginx/html# touch index2.html
```

进入该容器，创建 index1.html 文件

之后再 get pod 可以发现 pod 已成功运行

```yaml
[root@k8s-master01 6]# kubectl get pod
NAME             READY   STATUS    RESTARTS        AGE
startupprobe-1   1/1     Running   1 (3m50s ago)   8m51s
```

1.2 livenessProbe:

存活探针 livenessProbe 探测容器还是否存活，k8s 通过添加存活探针，解决虽然活着但是已经死了的问题。

如果 pod 内部不指定存活探测，可能会发生容器运行但是无法提供服务的情况

- 成功：静默
- 失败：根据重启的策略进行重启的动作
- 未知：静默

livenessProbe 选项说明：

- initialDelaySeconds: 容器启动后要等待多少秒后就探针开始工作，单位“秒”，默认是 0 秒，最小值是 0
- periodSeconds: 执行探测的时间间隔(单位是秒)，默认为 10s，单位“秒”，最小值是 1
- timeoutSeconds: 探针执行检测请求后，等待响应的超时时间，默认为 1s，单位“秒”，最小值是 1
- successThreshold: 探针检测失败后认为成功的最小连接成功次数，默认值为 1。必须为 1 才能激活和启动。最小值为 1。
- failureThreshold: 探测失败的重试次数，重试一定次数后将认为失败，默认值为 3 ，最小值为 1。

livenessProbe 实验

1.2.1 基于 HTTP Get 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-httpget-pod
  namespace: default
spec:
  containers:
  - name: liveness-httpget-container
    image: wangyanglinux/myapp:v1.0
    imagePullPolicy: IfNotPresent
    ports:
    - name: http
      containerPort: 80
    livenessProbe:
      httpGet:
        port: 80
        path: /index.html
      initialDelaySeconds: 1
      periodSeconds: 3
      timeoutSeconds: 3
```

具体配置说明

- **`containers`**:
  - `name: liveness-httpget-container`：容器的名称。
  - `image: wangyanglinux/myapp:v1.0`：容器使用的镜像。
  - `imagePullPolicy: IfNotPresent`：
    - 指定镜像拉取策略：优先使用本地缓存的镜像，只有在本地没有镜像时才从镜像仓库拉取。
  - **`ports`**:
    - 配置容器的网络端口：
      - `containerPort: 80`：容器监听的 HTTP 端口。
  - **存活探针 (`livenessProbe`)**:
    - 用于检测容器是否处于健康存活状态。
    - 配置：
      - **`httpGet`**:
        - 探针通过向容器的 HTTP 端口发送 GET 请求来检测健康状态。

    - 请求目标：`http://<容器_IP>:80/index.html`。
- **`initialDelaySeconds: 1`**：
    - 容器启动后等待 1 秒开始执行探针检查。
- **`periodSeconds: 3`**：
    - 每隔 3 秒执行一次探针检查。
- **`timeoutSeconds: 3`**：
    - 每次探针检查的超时时间为 3 秒。如果在 3 秒内没有响应，则探针检测失败。

运行机制

- 容器启动后，Kubernetes 等待 1 秒后开始执行 HTTP 存活探针。
- 探针向容器的 HTTP 端口 80 发起 GET 请求访问 `/index.html`：
    - 如果响应成功（HTTP 200 状态码），探针检测通过，容器被认为是健康的。
    - 如果响应超时（超过 3 秒）或返回非 200 状态码，则探针检测失败，Kubernetes 将判定容器失效并自动重启容器。

实验步骤：

YAML

```
[root@k8s-master01 6]# kubectl create -f livenessProbeGet.pod.yaml
pod/liveness-httpget-pod created
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS    RESTARTS      AGE
liveness-httpget-pod    1/1     Running   0             5s
```

创建 pod 后即成功运行，进入容器中删除`/index.html`

YAML

```
[root@k8s-master01 6]# kubectl exec -it liveness-httpget-pod -- /bin/ba
sh
liveness-httpget-pod:/# cd /usr/local/nginx/html/
liveness-httpget-pod:/usr/local/nginx/html# ls
50x.html       hostname.html  index.html
liveness-httpget-pod:/usr/local/nginx/html# rm index.html
liveness-httpget-pod:/usr/local/nginx/html# command terminated with ex
it code 137
```

删除后过一会儿，存活探针检测到当前容器已经死亡，就会将当前登录的容器踢下线

1.2.2 基于 exec 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-exec-pod
  namespace: default
spec:
  containers:
  - name: liveness-exec-container
    image: wangyanglinux/tools:busybox
    imagePullPolicy: IfNotPresent
    command: ["/bin/sh", "-c", "touch /tmp/live; sleep 60; rm -rf /tmp/live; sleep 3600"]
    livenessProbe:
      exec:
        command: ["test", "-e", "/tmp/live"]
      initialDelaySeconds: 1
      periodSeconds: 3
```

具体配置说明

- **`containers`**:
  - `name: liveness-exec-container`：容器的名称。
  - `image: wangyanglinux/tools:busybox`：容器使用的镜像。
  - `imagePullPolicy: IfNotPresent`：
    - 指定镜像拉取策略：优先使用本地缓存的镜像，只有在本地没有镜像时才从镜像仓库拉取。
  - **`command`**：
    - 指定容器启动时运行的命令：
      1. `touch /tmp/live`：创建一个临时文件 `/tmp/live`，作为存活状态标志。
      2. `sleep 60`：休眠 60 秒。
      3. `rm -rf /tmp/live`：删除 `/tmp/live` 文件。
      4. `sleep 3600`：休眠 3600 秒，保持容器运行。
- **存活探针 (`livenessProbe`)**:
  - 用于检测容器是否处于健康存活状态。
  - 配置：

- `exec`：
  - 通过在容器内部执行命令来检测健康状态。
  - 命令 `test -e /tmp/live` 用于检查文件 `/tmp/live` 是否存在。如果文件存在，探针检测通过；否则检测失败。
- `initialDelaySeconds: 1`：
  - 容器启动后等待 1 秒开始执行探针检查。
- `periodSeconds: 3`：
  - 每隔 3 秒执行一次探针检查。

运行机制

- 容器启动后，Kubernetes 等待 1 秒后开始执行 `livenessProbe` 探测。
- 探针检查 `/tmp/live` 文件是否存在：
  - 在前 60 秒内，文件 `/tmp/live` 存在，因此探针通过，容器被认为是健康的。
  - 60 秒后，文件被删除，探针检测失败。Kubernetes 将判定容器已失效并自动重启容器。

实验步骤：

YAML

```
[root@k8s-master01 6]# kubectl create -f livenessProbeExec.pod.yaml
pod/liveness-exec-pod created
[root@k8s-master01 6]# kubectl get pod
NAME                   READY   STATUS    RESTARTS        AGE
liveness-exec-pod      1/1     Running   0               4s
```

创建容器后等待 60s，存活探针发现 `/tmp/live` 已经没有，就会标记容 pod 已经死亡：

YAML

```
[root@k8s-master01 6]# kubectl get pod
NAME                   READY   STATUS    RESTARTS        AGE
liveness-exec-pod      1/1     Running   1 (2s ago)      101s
```

可以发现上面的 RESTARTS 已经变成 1，即当探针失败达到一定次数时（根据 `failureThreshold` 的配置），容器会被 Kubernetes 重启。Pod 只会在其中的所有容器都被重启时才会被认为需要重启，**由于这个 pod 中只有一个容器，因此 pod 也会被重启，Kubernetes 会保留 Pod 中的状态信息，但每个容器都会根据探针结果进行单独重启。**

执行 `kubectl describe pod liveness-exec-pod` ，在输出信息中可以找到容器相关信息，可以发现其 Restart Count 为 3，即容器已经重启了 3 次

YAML

```yaml
kubectl describe  pod liveness-exec-pod
Name:             liveness-exec-pod
Namespace:        default
Priority:         0
Service Account:  default
Node:             k8s-node01/192.168.66.12
Containers:
  liveness-exec-container:
    Container ID:  docker://1490def296e8785a9678acef29ce44daad5831f00ac655e4bc74a3c352d0f481
    Image:         wangyanglinux/tools:busybox
    Image ID:      docker-pullable://wangyanglinux/tools@sha256:a024bc31a3a6d57ad06e0a66efa453c8cbdf818ef8d720ff6d4a36027dd1f0ae
    Port:          <none>
    Host Port:     <none>
    Command:
      /bin/sh
      -c
      touch /tmp/live; sleep 60; rm -rf /tmp/live; sleep 3600
    State:          Running
      Started:      Sun, 15 Dec 2024 17:50:48 +0800
    Last State:     Terminated
      Reason:       Error
      Exit Code:    137
      Started:      Sun, 15 Dec 2024 17:49:09 +0800
      Finished:     Sun, 15 Dec 2024 17:50:48 +0800
    Ready:          True
    Restart Count:  3
    Liveness:       exec [test -e /tmp/live] delay=1s timeout=1s period=3s #success=1 #failure=3
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-7mh69 (ro)
```

同时，根据 `Node: k8s-node01/192.168.66.12` ，可以知道当前 pod 被部署在 node01 上，在 node01 上执行如下命令，可以发现该容器 name 后面带有 `_15` ，即该容器已经反复重启了 15 次

容器 2 (`f629eee8e623`) 是 Kubernetes 为每个 Pod 自动创建的 `pause` 容器。 `pause` 容器不执行任何实际任务，而是保持 Pod 内网络命名空间的存在。每个 Pod 必须至少有一个 `pause` 容器。

YAML

```
[root@k8s-node01 ~]# docker ps | grep liveness-exec-pod
371e2d0fda76   adf0836b2bab                                                "/bin/sh -c 'touch /."   9 seconds ago    Up 8 seconds              k8s_liveness-exec-container_liveness-exec-pod_default_d6c01f8e-0b44-4ce9-898c-7920de1db7a2_15
f629eee8e623   registry.aliyuncs.com/google_containers/pause:3.8   "/pause"                 47 minutes ago   Up 47 minutes             k8s_POD_liveness-exec-pod_default_d6c01f8e-0b44-4ce9-898c-7920de1db7a2_0
```

1.2.3 基于 TCP Check 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: liveness-tcp-pod
spec:
  containers:
  - name: liveness-tcp-container
    image: wangyanglinux/myapp:v1.0
    livenessProbe:
      tcpSocket:
        port: 80
      initialDelaySeconds: 5
      timeoutSeconds: 1
```

具体配置说明

- **`containers`**:
  - `name: liveness-tcp-container`：容器的名称。
  - `image: wangyanglinux/myapp:v1.0`：容器使用的镜像。
- **存活探针 (`livenessProbe`)**:
  - 用于检测容器是否处于健康存活状态。
  - 配置：

- `tcpSocket`：
  - 通过向容器的 TCP 端口（这里是端口 80）发起连接请求来检查容器是否存活。
  - 如果 TCP 连接成功，则表示容器存活；如果连接失败，则容器会被认为不健康。
- `initialDelaySeconds: 5`：
  - 容器启动后等待 5 秒开始执行存活探针。
- `timeoutSeconds: 1`：
  - 每次探针检测的超时时间为 1 秒。如果在 1 秒内无法建立 TCP 连接，探针检测失败。

运行机制

- 容器启动后，Kubernetes 等待 5 秒后开始执行 TCP 存活探针。
- 探针尝试连接容器的端口 80。
  - 如果容器的端口 80 可访问并且建立了 TCP 连接，探针检测通过，容器被认为是健康的。
  - 如果端口 80 不可访问，探针检测失败，Kubernetes 将重启容器。

1.3 readinessProbe:

readinessProbe 谈探针探测容器是否准备好提供服务，k8s 通过添加就绪探针，解决尤其是在**扩容时保证提供给用户的服务都是可用的**。

如果 pod 内部的 C 不添加就绪探测，默认就绪。如果添加了就绪探测， 只有就绪通过以后，才标记修改为就绪状态。当前 pod 内的所有的 **C 都就绪**，才标记当前 **Pod 就绪**

- 成功：将当前的 C 标记为就绪
- 失败：静默
- 未知：静默

readinessProbe 选项说明：

- initialDelaySeconds: 容器启动后要等待多少秒后就探针开始工作，单位“秒”，默认是 0 秒，最小值是 0
- periodSeconds: 执行探测的时间间隔(单位是秒)，默认为 10s，单位“秒”，最小值是 1
- timeoutSeconds: 探针执行检测请求后，等待响应的超时时间，默认为 1s，单位“秒”，最小值是 1
- successThreshold: 探针检测失败后认为成功的最小连接成功次数，默认值为 1。必须为 1 才能激活和

启动。最小值为 1。

- failureThreshold: 探测失败的重试次数，重试一定次数后将认为失败，默认值为 3 ，最小值为 1。

readinessProbe 实验

1.2.1 基于 HTTP Get 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: readiness-httpget-pod
  namespace: default
  labels:
    app: myapp
    env: test
spec:
  containers:
  - name: readiness-httpget-container
    image: wangyanglinux/myapp:v1.0
    imagePullPolicy: IfNotPresent
    readinessProbe:
      httpGet:
        path: /index1.html
        port: 80
      initialDelaySeconds: 1
      periodSeconds: 3
```

具体配置说明

- **`containers`**:
  - `name: readiness-httpget-container`：容器名称。
  - `image: wangyanglinux/myapp:v1.0`：容器使用的镜像及版本。
  - `imagePullPolicy: IfNotPresent`:
    - 指定镜像拉取策略：优先使用本地缓存的镜像，只有在本地没有镜像时才从镜像仓库拉取。
- **就绪探针 (`readinessProbe`)**:
  - 用于检测容器是否已经准备好接收流量。
  - 配置：
    - **`httpGet`**:
      - 通过 HTTP GET 请求检查 `/index1.html` 页面是否返回 2xx 或 3xx 的状态码，作为健康检查的依据。

        - `port: 80`：指定容器内提供 HTTP 服务的端口。
    - **`initialDelaySeconds: 1`**：
        - 容器启动后等待 1 秒开始健康检查。
    - **`periodSeconds: 3`**：
        - 每隔 3 秒执行一次健康检查。

运行机制

- 当 Pod 的容器启动后，Kubernetes 会按照 `readinessProbe` 配置进行定期检查。
- 只有当探针检测返回成功时（HTTP 状态码为 2xx 或 3xx），Pod 的状态才会被标记为 `Ready`。
- 如果检测失败，Pod 不会被标记为 `Ready`，并从 Service 的端点中移除，避免向该 Pod 发送流量。

实验步骤：

YAML

```
[root@k8s-master01 6]# kubectl create -f readinessProbe.pod.yaml
pod/readiness-httpget-pod created
[root@k8s-master01 6]# kubectl get pod
NAME                     READY   STATUS    RESTARTS      AGE
readiness-httpget-pod    0/1     Running   0             7s
```

创建 pod 后可以发现 pod 一直处于 READY 0/1 状态，因为 **`readinessProbe`** 没有检测到 `/index1.html`

YAML

```
[root@k8s-master01 6]# kubectl exec -it readiness-httpget-pod -- /bin/bash
readiness-httpget-pod:/# cd /usr/local/nginx/html/
readiness-httpget-pod:/usr/local/nginx/html# ls
50x.html       hostname.html  index.html
readiness-httpget-pod:/usr/local/nginx/html# touch index1.html
```

进入容器，创建 index1.html 文件

再次 get pod 发现 pod 已成功运行

YAML

```
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS    RESTARTS      AGE
readiness-httpget-pod   1/1     Running   0             2m8s
```

1.2.2 基于 exec 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: readiness-exec-pod
  namespace: default
spec:
  containers:
  - name: readiness-exec-container
    image: wangyanglinux/tools:busybox
    imagePullPolicy: IfNotPresent
    command: ["/bin/sh", "-c", "touch /tmp/live; sleep 60; rm -rf /tmp/live; sleep 3600"]
    readinessProbe:
      exec:
        command: ["test", "-e", "/tmp/live"]
      initialDelaySeconds: 1
      periodSeconds: 3
```

具体配置说明

- **`containers`**：
  - `name: readiness-exec-container`：容器的名称。
  - `image: wangyanglinux/tools:busybox`：容器使用的镜像。
  - `imagePullPolicy: IfNotPresent`：
    - 指定镜像拉取策略：优先使用本地缓存的镜像，只有在本地没有镜像时才从镜像仓库拉取。
  - **`command`**：
    - 指定容器启动时运行的命令：
      1. `touch /tmp/live`：创建一个临时文件 `/tmp/live`，作为标志文件。
      2. `sleep 60`：休眠 60 秒。

3. `rm -rf /tmp/live`：删除 `/tmp/live` 文件。
4. `sleep 3600`：休眠 3600 秒，保持容器运行。

就绪探针 (`readinessProbe`):

- 用于检测容器是否已准备好接收流量。
- 配置：
  - **`exec`**：
    - 通过在容器内执行命令来检测健康状态。
    - 命令 `test -e /tmp/live` 用于检查文件 `/tmp/live` 是否存在。如果文件存在，探针检测通过；否则检测失败。
  - **`initialDelaySeconds: 1`**：
    - 容器启动后等待 1 秒开始执行探针检查。
  - **`periodSeconds: 3`**：
    - 每隔 3 秒执行一次探针检查。

- **运行机制**
- 容器启动后：
  a. 首先创建 `/tmp/live` 文件，表示容器已准备好接收流量。
  b. 就绪探针定期检查 `/tmp/live` 文件是否存在。
  c. 60 秒后，文件被删除，探针检测将失败，Pod 的状态将被标记为 `NotReady`。
- 删除 `/tmp/live` 后，容器仍然运行，但不会再被标记为 `Ready`，因此不会接收流量。

实验步骤：

YAML

```
[root@k8s-master01 6]# kubectl create -f readinessExec.pod.yaml
pod/readiness-exec-pod created
[root@k8s-master01 6]# kubectl get pod
NAME                   READY   STATUS    RESTARTS      AGE
readiness-exec-pod     1/1     Running   0             4s
```

创建后发现 pod 已经成功运行，等待 60s 后发现 pod 已经 `NotReady`：

YAML

```
[root@k8s-master01 6]# kubectl get pod
NAME                   READY   STATUS    RESTARTS      AGE
readiness-exec-pod     0/1     Running   0             98s
```

1.2.3 基于 TCP Check 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: readiness-tcp-pod
spec:
  containers:
  - name: readiness-exec-container
    image: wangyanglinux/myapp:v1.0
    readinessProbe:
      tcpSocket:
        port: 80
      initialDelaySeconds: 5
      timeoutSeconds: 1
```

具体配置说明

- **`containers`**:
  - `name: readiness-exec-container`：容器的名称。
  - `image: wangyanglinux/myapp:v1.0`：容器使用的镜像。
- **就绪探针 (`readinessProbe`)**:
  - 用于检测容器是否已准备好接收流量。
  - 配置：
    - **`tcpSocket`**:
      - 探测 TCP 端口 80 是否开放。
      - 通过向指定端口发送 TCP 连接请求，如果连接成功，则认为探针检测通过；否则检测失败。
    - **`initialDelaySeconds: 5`**:
      - 容器启动后等待 5 秒开始执行探针检查。
    - **`timeoutSeconds: 1`**:
      - 每次探针检测的超时时间为 1 秒。如果连接未能在 1 秒内成功，则认为检测失败。

运行机制

- 容器启动后，Kubernetes 等待 5 秒后开始执行 TCP 探针。
- 探针会尝试连接容器的 TCP 端口 80。

- 如果端口 80 可访问，则探针通过，Pod 被标记为 `Ready` 状态。
- 如果端口 80 不可访问，探针失败，Pod 的状态为 `NotReady`，不会接收流量。

三、生命周期钩子

Pod hook(钩子)是由 Kubernetes 管理的 **kubelet 发起的**，当容器中的进程启动前或者容器中的

进程终止之前运行，这是包含在容器的生命周期之中。**钩子是定义在容器之下的**。可以同时为 Pod 中的所有容器都配置 hook

Hook 的类型包括两种：

- exec: 执行一段命令
- HTTP: 发送 HTTP 请求

当 Kubernetes 决定停止容器时，它会首先向容器发送 SIGTERM 信号（终止信号）。此时容器可以选择通过 `preStop` 钩子来执行一些清理操作。

生命周期钩子实验

1. 基于 HTTP Get 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-exec-pod
spec:
  containers:
  - name: lifecycle-exec-container
    image: wangyanglinux/myapp:v1
    lifecycle:
      postStart:
        exec:
          command: ["/bin/sh", "-c", "echo postStart > /usr/share/mess
age"]
      preStop:
        exec:
          command: ["/bin/sh", "-c", "echo preStop > /usr/share/messag
e"]
```

配置说明：

- **containers**:
  - **name: lifecycle-exec-container**：容器的名称。
  - **image: wangyanglinux/myapp:v1**：容器使用的镜像。
- **生命周期钩子 (lifecycle)**:
  - **postStart**：
    - **exec**：
      - **command: ["/bin/sh", "-c", "echo postStart > /usr/share/message"]**：容器启动后执行的命令。在容器启动时，会在 `/usr/share/message` 文件中写入 `postStart` 字符串。
  - **preStop**：
    - **exec**：
      - **command: ["/bin/sh", "-c", "echo preStop > /usr/share/message"]**：容器停止前执行的命令。在容器停止时，会在 `/usr/share/message` 文件中写入 `preStop` 字符串。

运行机制：

- **postStart**：容器启动后立即执行的命令，将 `postStart` 写入文件。
- **preStop**：pod 停止前执行的命令，将 `preStop` 写入文件。这可以用于容器清理或在容器停止前做一些自定义操作。

实验步骤：

YAML

```
[root@k8s-master01 6]# kubectl create -f lifecycle-exec-pod.pod.yaml
pod/lifecycle-exec-pod created
[root@k8s-master01 6]# kubectl get pod
NAME                  READY   STATUS    RESTARTS       AGE
lifecycle-exec-pod    1/1     Running   0              56s
```

可以进入容器内部查看 `/usr/share/message` 相应记录

YAML

```
[root@k8s-master01 6]# kubectl exec -it lifecycle-exec-pod -c lifecycle
-exec-container -- /bin/sh
/ # cat /usr/share/message
postStart
```

由于在 Kubernetes 中，容器的停止过程通常是由 Kubernetes 控制平面触发的，通常而不能手动停止容器（就算去 pod 所在的 node 手动 docker stop 相应容器，也不会触发 `preStop`，kubernetes 会马上重启相应容器），可以 delete pod 来触发容器停止

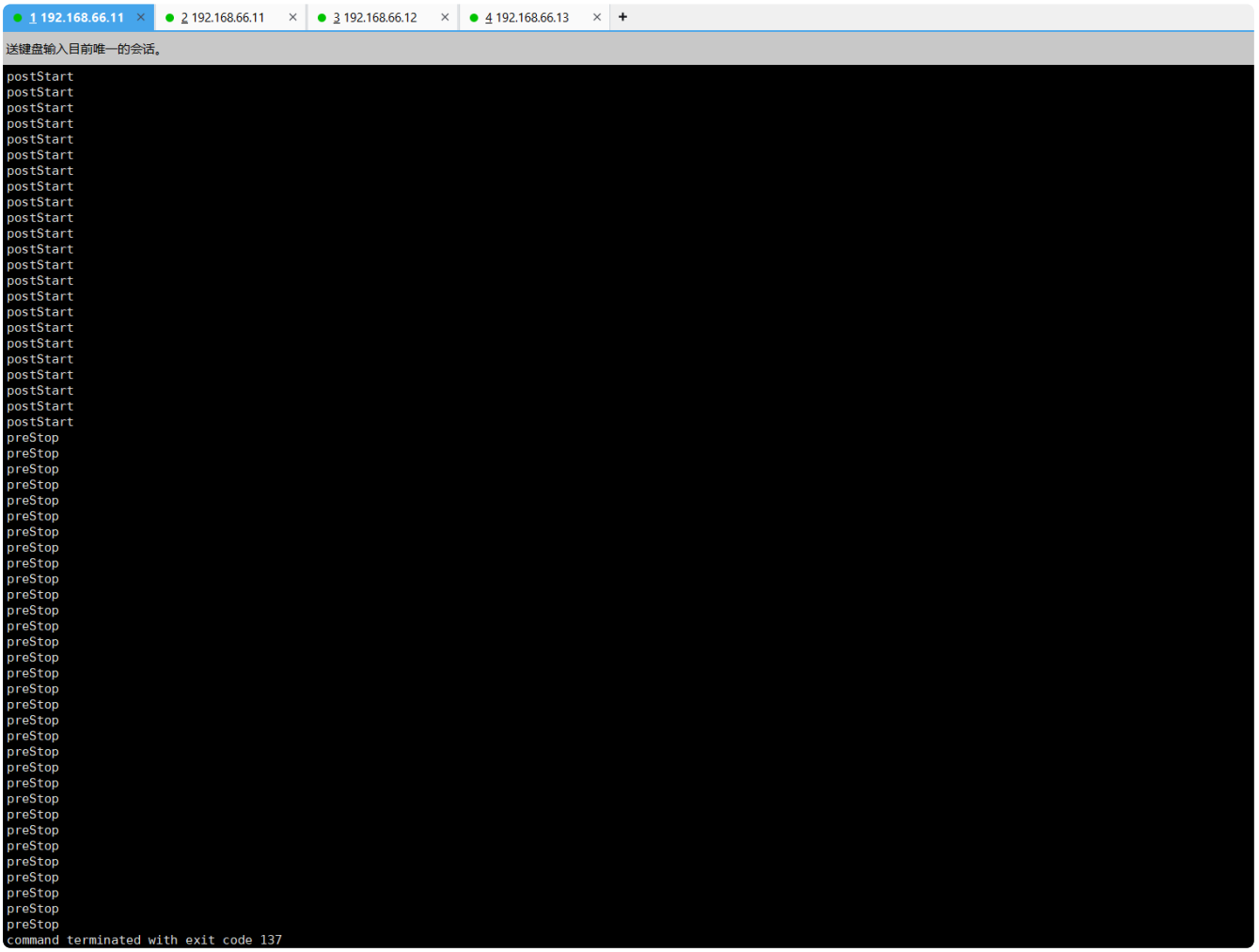
进入容器，执行下面的代码，循环打印 /usr/share/message 中的内容，再另一个窗口中将该 pod 杀死，之后可以发现触发了 prestop

YAML

```
while true;
> do
> cat /usr/share/message
> done
```

YAML

```
kubectl delete pod lifecycle-exec-pod
```



2. 基于 HTTP Get 方式

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-httpget-pod
  labels:
    name: lifecycle-httpget-pod
spec:
  containers:
  - name: lifecycle-httpget-container
    image: wangyanglinux/myapp:v1.0
    ports:
      - containerPort: 80
    lifecycle:
      postStart:
        httpGet:
          host: 192.168.66.11
          path: /index.html
          port: 1234
      preStop:
        httpGet:
          host: 192.168.66.11
          path: /hostname.html
          port: 1234
```

配置说明：

- **containers**:
  - **name: lifecycle-httpget-container**：容器的名称。
  - **image: wangyanglinux/myapp:v1.0**：容器使用的镜像。
  - **ports**:
    - **containerPort: 80**：容器暴露的端口。
- **生命周期钩子 (lifecycle)**:
  - **postStart**：
    - **httpGet**：
      - **host: 192.168.66.11**：请求的目标主机 IP 地址。
      - **path: /index.html**：请求的路径。
      - **port: 1234**：请求的目标端口。
  - **preStop**：

- **httpGet**：
  - **host: 192.168.66.11**：请求的目标主机 IP 地址。
  - **path: /hostname.html**：请求的路径。
  - **port: 1234**：请求的目标端口。

运行机制：

- **postStart**：容器启动后，通过 HTTP GET 请求访问 `http://192.168.66.11:1234/index.html` 路径。如果请求成功，容器被视为启动完成。
- **preStop**：容器停止前，通过 HTTP GET 请求访问 `http://192.168.66.11:1234/hostname.html` 路径。容器将在请求成功后开始停止。

实验步骤：

创建一个测试的 webServer：

YAML

```
docker run -it --rm -p 1234:80 wangyanglinux/myapp:v1.0
```

之后再创建下面的 pod

YAML

```
[root@k8s-master01 4]# kubectl create -f  lifecycle-httpget-pod.yaml
pod/lifecycle-httpget-pod created
[root@k8s-master01 4]# kubectl get pod
NAME                    READY   STATUS              RESTARTS         AGE
lifecycle-httpget-pod   1/1     Running             0                4s
```

会出现下面的 postStart 访问记录

```
^C[root@k8s-master01 6]# docker run -it --rm -p 1234:80 wangyanglinux/myapp:v1.0
192.168.66.13 - - [15/Dec/2024:19:54:36 +0800] "GET /index.html HTTP/1.1" 200 48 "-" "kube-lifecycle/1.29"
```

删除 pod，又会出现 preStop 访问记录

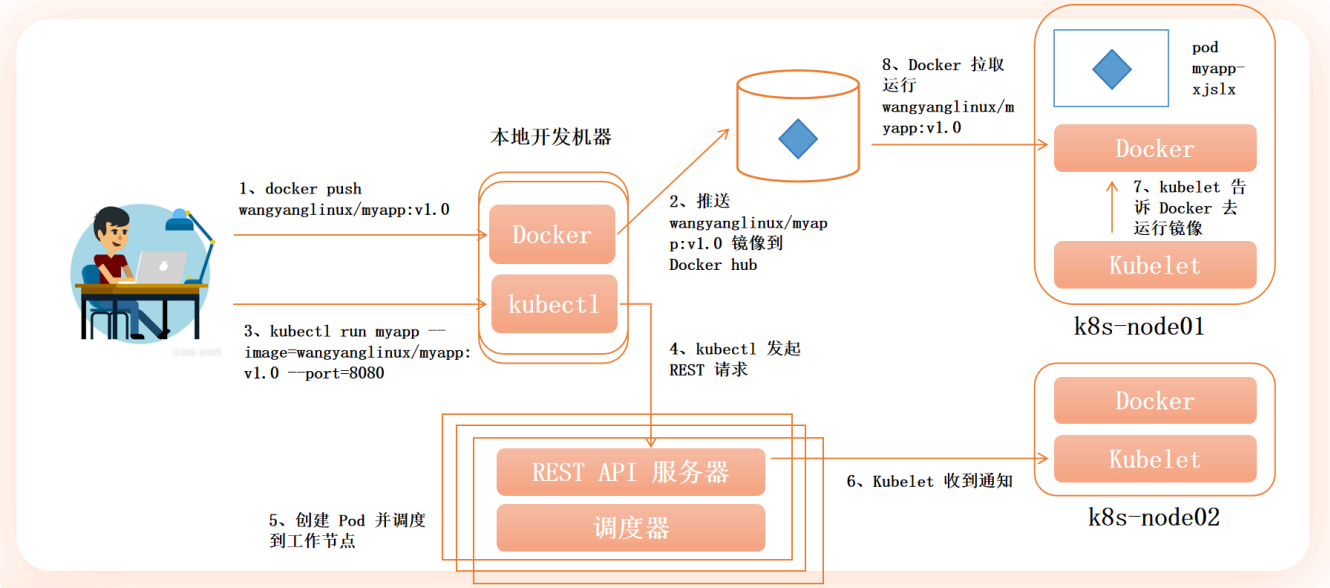
```
^C[root@k8s-master01 6]# docker run -it --rm -p 1234:80 wangyanglinux/myapp:v1.0
192.168.66.13 - - [15/Dec/2024:19:54:36 +0800] "GET /index.html HTTP/1.1" 200 48 "-" "kube-lifecycle/1.29"
192.168.66.13 - - [15/Dec/2024:19:57:36 +0800] "GET /hostname.html HTTP/1.1" 200 13 "-" "kube-lifecycle/1.29"
```

也可以直接在相应 node 查看 logs：

YAML

```
[root@k8s-master01 4]# docker logs 0dbb058d2abc
192.168.66.13 - - [15/Dec/2024:19:54:36 +0800] "GET /index.html HTTP/1.1" 200 48 "-" "kube-lifecycle/1.29"
192.168.66.13 - - [15/Dec/2024:19:57:36 +0800] "GET /hostname.html HTTP/1.1" 200 13 "-" "kube-lifecycle/1.29"
```

四、Pod 是如何被调度运行的



五、生命周期整合实验

Pod 生命周期中的 initC、startupProbe、livenessProbe、readinessProbe、hook 都是可以同时存在的，可以选择全部、部分或者完全不用。

YAML

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: lifecycle-pod
  labels:
    app: lifecycle-pod
spec:
  containers:
    - name: busybox-container
      image: wangyanglinux/tools:busybox
      command: ["/bin/sh", "-c", "touch /tmp/live; sleep 600; rm -rf /tmp/live; sleep 3600"]
      livenessProbe:
        exec:
          command: ["test", "-e", "/tmp/live"]
        initialDelaySeconds: 1
        periodSeconds: 3
      lifecycle:
        postStart:
          httpGet:
            host: 192.168.66.11
            path: /index.html
            port: 1234
        preStop:
          httpGet:
            host: 192.168.66.11
            path: /hostname.html
            port: 1234

    - name: myapp-container
      image: wangyanglinux/myapp:v1.0
      livenessProbe:
        httpGet:
          port: 80
          path: /index.html
        initialDelaySeconds: 1
        periodSeconds: 3
        timeoutSeconds: 3
      readinessProbe:
        httpGet:
          port: 80
          path: /index1.html
```

```
42          initialDelaySeconds: 1
43          periodSeconds: 3
44
45    initContainers:
46      - name: init-myservice
47        image: wangyanglinux/tools:busybox
48        command: ['sh', '-c', 'until nslookup myservice; do echo waitin
   g for myservice; sleep 2; done;']
49
50      - name: init-mydb
51        image: wangyanglinux/tools:busybox
52        command: ['sh', '-c', 'until nslookup mydb; do echo waiting for
   mydb; sleep 2; done;']
53
```

`initContainers` 中 `until nslookup myservice` 通过 `nslookup` 查询该服务名是否能解析到一个有效的 ip 地址，直到解析成功再启动主应用容器

如果不创建 myservice，Init 将一直是 0/2 的状态

```
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS     RESTARTS        AGE
lifecycle-pod           0/2     Init:0/2   0               20m
```

而当创建了 myservice 后，Init 将变成 1/2 的状态

Bash

```
1 kubectl create svc clusterip myservice --tcp=80:80
```

```
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS     RESTARTS        AGE
lifecycle-pod           0/2     Init:1/2   0               22m
```

而当创建了 mydb 后，pod 就会进入初始化状态

Bash

```
1 kubectl create svc clusterip mydb --tcp=80:80
```

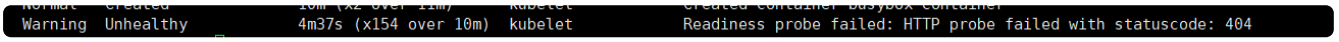
```
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS            RESTARTS      AGE
lifecycle-pod           0/2     PodInitializing   0             23m
```

之后变成如下图所示，Pod 的状态为 `CrashLoopBackOff` 时，表示容器在启动时发生了崩溃，并且 Kubernetes 会尝试重新启动该容器。如果容器持续崩溃并达到一定次数的重启限制，Kubernetes 会停止尝试重新启动容器并进入 `CrashLoopBackOff` 状态（还是会重启）。

```
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS             RESTARTS      AGE
lifecycle-pod           0/2     CrashLoopBackOff   3 (20s ago)   26m
```

因为在 busybox-container 容器定义了 postStart 探针，需要访问 192.168.66.11 的服务

通过 `kubectl describe` 可以发现相应服务还不存在，访问出现了 404



YAML

```yaml
lifecycle:
  postStart:
    httpGet:
      host: "192.168.66.11"
      path: "/index.html"
      port: 1234
```

创建相应服务

YAML

```
docker run --name test -p 1234:80 -d wangyanglinux/myapp:v1.0
```

等待一会儿即可恢复 Running

```
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS             RESTARTS        AGE
lifecycle-pod           0/2     CrashLoopBackOff   6 (3m40s ago)   36m
pod-1                   1/1     Running            0               135m
pod-2                   1/1     Running            0               134m
pod-3                   1/1     Running            0               130m
readiness-exec-pod      0/1     Running            1 (23m ago)     84m
readiness-httpget-pod   1/1     Running            0               97m
[root@k8s-master01 6]# kubectl get pod
NAME                    READY   STATUS    RESTARTS       AGE
lifecycle-pod           1/2     Running   7 (6m9s ago)   38m
```

但 Pod 的 ready 就绪状态仍然一直处于 1/2，因为 myapp-container 容器中添加了 readinessProbe 就绪探测，需要访问该容器中的 index1.html 文件，才能通过就绪探测

Bash

```
kubectl exec -it lifecycle-pod -c myapp-container -- /bin/bash
```

执行上面的命令进入 myapp-container 容器，进入 /usr/local/nginx/html 目录下创建 index1.html 文件

创建后可以发现该 pod 已经 ready2/2

```
[root@k8s-master01 ~]# kubectl get pod
NAME                    READY   STATUS    RESTARTS       AGE
lifecycle-pod           2/2     Running   75 (10m ago)   7h1m
```

同时由于定义了下面的存活探针

Bash

```
livenessProbe:
  httpGet:
    port: 80
    path: /index.html
  initialDelaySeconds: 1
  periodSeconds: 3
  timeoutSeconds: 3
```

可以尝试将其中的 /index.html 删除，等待一会儿，容器就会被强制退出，因为存活探针检测到容器已经未存活，就将当前进入容器的状态踢下线了

```
lifecycle-pod:/usr/local/nginx/html# mv index.html index.html.back
lifecycle-pod:/usr/local/nginx/html# command terminated with exit code 137
[root@k8s-master01 ~]#
```

存活探针检测到容器已经未存活后，就会重新启动一个全新的 myapp-container 容器，`get pod`：

```
[root@k8s-master01 ~]# kubectl get pod
NAME                    READY   STATUS    RESTARTS       AGE
lifecycle-pod           1/2     Running   78 (29s ago)   7h17m
```

可以发现 pod 又变回了前面的 ready 就绪状态处于 1/2 的情况，因为全新启动的容器中是没有 index1.html 的，无法通过 myapp-container 容器的就绪检测，需要重新进入 myapp-container 容器，创建 index1.html 文件

```bash
kubectl exec -it lifecycle-pod -c myapp-container -- /bin/bash
cd /usr/local/nginx/html/
touch index1.html
```

之后可以发现容器已就绪

```
[root@k8s-master01 ~]# kubectl get pod
NAME                  READY   STATUS    RESTARTS        AGE
lifecycle-pod         2/2     Running   78 (6m3s ago)   7h22m
```

由于定义了启动后钩子 postStart，可以通过 `docker logs test` 查看该访问记录

（docker logs test 显示的是前面我们在 192.168.66.11 的 80 端口起 test 容器中的 http 服务对应 docker 容器的访问日志，前面所有对 192.168.66.11 的 http 探测都是访问的 test 容器的服务，其中封装了 nginx）

```bash
postStart:
  httpGet:
    host: 192.168.66.11
    path: /index.html
    port: 1234
```

```
[root@k8s-master01 ~]# docker logs test
192.168.66.12 - - [14/Dec/2024:16:10:47 +0800] "GET /index.html HTTP/1.1" 200 48 "-" "kube-lifecycle/1.29"
```

六、补充

在 k8s 中，理想的状态是 pod 优雅释放，但是并不是每一个 Pod 都会这么顺利

- Pod 卡死，处理不了优雅退出的命令或者操作
- 优雅退出的逻辑有 BUG，陷入死循环
- 代码问题，导致执行的命令没有效果

对于以上问题，k8s 的 Pod 终止流程中还有一个"最多可以容忍的时间"，即 grace period (在 pod. spec.terminationGracePeriodSeconds 字段定义)，这个值默认是 30 秒，当我们执行 kubectl

delete` 的时候也可以通过 **--grace-period** 参数显示指定一个优雅退出时间来覆盖 Pod 中的配置，如果我们配置的 grace period 超过时间之后，k8s 就只能选择强制 kill Pod。值得注意的是，这与 preStop

Hook 和 SIGTERM 信号并行发生。**k8s 不会等待 preStop Hook 完成**。如果你的应用程序完成关闭并在

terminationGracePeriod 完成之前退出，k8s 会立即进入下一步清理与 Pod 相关的资源。

触发终止信号：

- 当 `kubectl delete pod` 命令被执行后：
  - Kubernetes 会向 Pod 内的容器发送 `SIGTERM` 信号。
  - 如果定义了 `preStop` Hook，它会立即触发，和 SIGTERM 并行运行。
- 容器需要在 `terminationGracePeriodSeconds` 内完成终止操作。

优雅终止：

- 应用程序在收到 `SIGTERM` 信号后，应执行清理任务（如保存数据、关闭连接等）。
- 如果应用程序**未在** `terminationGracePeriodSeconds` **内退出**，Kubernetes 将进入下一步强制终止。
- 如果应用程序在 **terminationGracePeriodSeconds** 内成功退出，Kubernetes 将立即进入下一步清理与 Pod 相关的资源，而不会等待剩余的优雅终止时间。这意味着 Pod 的终止流程会更快完成，

强制终止（Force Kill）：

- 如果在 **grace period** 时间结束后：
  - 容器仍未退出（可能因为处理 SIGTERM 耗时太长或逻辑卡死）。
  - Kubernetes 会向容器发送 `SIGKILL` 信号。
  - `SIGKILL` 会立即强制终止容器的进程，无法被拦截或忽略。
- 强制终止后，容器会被移除，Pod 的状态会从 `Terminating` 转为 `Succeeded` 或 `Failed`。

清理 Pod 的资源：

- 在容器被终止后，Kubernetes 会清理 Pod 相关的资源（如挂载的存储卷、网络配置等）。
- 如果 Pod 是通过控制器（如 Deployment）创建的，控制器会自动启动一个新的 Pod 来替代被删除的 Pod。