链滴

# K8S 服务健康检测

作者：someone61489
原文链接：https://ld246.com/article/1604631072633
来源网站：链滴

# K8S服务健康检测机制

# 1.业务探针 readinessProbe

## 1.1 为什么自动扩容导致请求失败?

一个新Pod创建后，Service就能立即选择到它，并会把请求转发给Pod，那问题就来了，通常一个Po启动是需要时间的，如果Pod还没准备好（可能需要时间来加载配置或数据，或者可能需要执行 一个热程序之类），这时把请求转给Pod的话，Pod也无法处理，造成请求失败。

## 1.2 如何解决?

加入业务探针，可以理解为类似ws的心跳检测机制.

Kubernetes中解决这个问题的方法就是给Pod加一个业务就绪探针Readiness Probe，当检测到Pod绪后才允许Service请求转给Pod。

Readiness Probe周期性检测Pod，然后根据响应来判断Pod是否就绪，Service根据就绪状态分发流。

## 1.3 配置详解

### 1.3.1 EXEC(shell)检测

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
```

原文链接：K8S 服务健康检测

```
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        name: container-0
        resources:
          limits:
            cpu: 500m
            memory: 1024Mi
          requests:
            cpu: 500m
            memory: 1024Mi
    // 重点内容
    // 声明一个 业务探针 去检测 当前服务是否准备完毕
        readinessProbe:      # Readiness Probe
          exec:
       // 在容器中先后执行 ls，/ready 命令,如果命令完整执行完毕，则标志POD为就绪状态，可以从S
rvice接收流量
            command:
            - ls
            - /ready
      imagePullSecrets:
      - name: imagepull-secret
```

## 1.3.2 HTTP

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        name: container-0
        resources:
          limits:
            cpu: 500m
```

```
          memory: 1024Mi
        requests:
          cpu: 500m
          memory: 1024Mi
    // 这里是重点
    // 声明一个业务 探针，向本POD发起 /read请求，根据响应码判断POD是否准备就绪
      readinessProbe:
        httpGet:
          path: /read
          port: 80
    imagePullSecrets:
    - name: imagepull-secret
```

## 1.3.3 可选参数

- initialDelaySeconds: 10  # 容器启动后多久开始探测
- timeoutSeconds: 2  # 表示容器必须在2s内做出相应反馈给probe，否则视为探测失败
- periodSeconds: 30  # 探测周期，每30s探测一次
- successThreshold: 1  # 连续探测1次成功表示成功
- failureThreshold: 3  # 连续探测3次失败表示失败

## 1.3.4 配置实例

实例根据业务需求检测服务就绪状态，涉及到Springboot内容不做讲解

```
@RestController
@RequestMapping("/checkpoint")
public class CheckController {

    // 构造一个请求,如果可以请求通,返回Http200和响应
    @GetMapping(value = "/service")
    public ResponseEntity<ReturnEntity<JSONObject>> service() {
        try {
            JSONObject json = new JSONObject();
            json.put("type", "service");
            json.put("result", "ok");
            json.put("data", "request is healthy");
            return Return.build(true, "success", json);
        } catch (Exception e) {
            e.printStackTrace();
            return Return.build(false, "failed");
        }
    }

}
```

```
spec:
      serviceAccount: cloud
      containers:
        - image:  url
          name: wechat
```

```
    // 声明业务指针
        readinessProbe:
          httpGet:
        // Spring端口
            port: 8080
       // 请求路径
            path: /wechat/checkpoint/service
    // 容器启动后30秒开始检测,这里根据自身服务启动时间配置
          initialDelaySeconds: 30
    // 5秒内没有响应为失败
          timeoutSeconds: 5
   // 探测周期，每30s探测一次
          periodSeconds: 30
   // 请求3次如果都失败了那么POD不能为准备就绪状态
          failureThreshold: 3
```

# 2.存活探针 livenessProbe

## 2.1 为什么服务假死?

由于代码或者使用的第三方包对异常处理的问题所以导致在外面 kubectl get pod看服务正在运行 Running但 log容器的时候发现服务已经无法正常运行

## 2.2 如何解决

Kubernetes提供了自愈的能力，具体就是能感知到容器崩溃，然后能够重启这个容器。但是有时候如Java程序内存泄漏了，程序无法正常工作，但是JVM进程却是一直运行的，对于这种应用本身业务了问题的情况，kubernetes提供了liveness probe机制，通过检测容器响应是否正常来决定是否重启这是一种很好的健康检查机制。

毫无疑问，每个pod最好都定义liveness probe，否则Kubernetes无法感知Pod是否正常运行。

## 2.3 配置详解

### 2.3.1 EXEC(shell)

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
   // 这里是重点
```

```
  // 这里定义了一个存活探针,先后执行 cat,/temp/healthy命令去检测服务是否健康
   livenessProbe:          # liveness probe
     exec:                 # Exec定义
       command:
       - cat
       - /tmp/healthy
```

## 2.3.2 HTTP

```
apiVersion: v1
kind: Pod
metadata:
  name: liveness-http
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/liveness
    args:
    - /server
   // 这里是重点
  // 声明了一个存活探针，发送/healthz获取http状态码来判断服务是否健康
    livenessProbe:          # liveness probe
      httpGet:              # HTTP GET定义
        path: /healthz
        port: 8080
```

## 2.3.3 可选参数

- initialDelaySeconds: 10    # 容器启动后多久开始探测
- timeoutSeconds: 2          # 表示容器必须在2s内做出相应反馈给probe，否则视为探测失败
- periodSeconds: 30          # 探测周期，每30s探测一次
- successThreshold: 1        # 连续探测1次成功表示成功
- failureThreshold: 3        # 连续探测3次失败表示失败

## 2.3.4 配置实例

实例根据业务需求检测服务是否丢失redis和oracle 连接，涉及到Springboot内容不做讲解

```
@RestController
@RequestMapping("/checkpoint")
public class CheckController {

    @Resource
    private RedisTemplate<String, String> redisTemplate;

    @Resource
    private WechatPointRepo repo;

    @GetMapping(value = "/conncp")
    public ResponseEntity<ReturnEntity<JSONObject>> connectionCheckPoint() {
```

```
        try {
            JSONObject json = new JSONObject();
            json.put("type", "redis");
            redisTemplate.opsForValue().set("checkpoint_redis", "live");
            String val = redisTemplate.opsForValue().get("checkpoint_redis");
          List<WechatCheckPoint> pointList = repo.findAll();
            json.put("result", "ok");
            return Return.build(true, "success", json);
        } catch (Exception e) {
            e.printStackTrace();
            return Return.build(false, "failed");
        }
    }

}

spec:
      serviceAccount: cloud
      containers:
        - image: url
          name: wechat
    // 声明一个存活探针
          livenessProbe:
            httpGet:
        // POD内部Spring 服务端口8080
              port: 8080
        // 路径
              path: /wechat/checkpoint/conncp
    // POD启动30秒后进行存活探测
            initialDelaySeconds: 30
    // 5s内服务必须返回response
            timeoutSeconds: 5
    // 30秒探测一次
            periodSeconds: 30
    // 如果连续失败3次 标志服务不健康了，自动重启
            failureThreshold: 3
```

原文链接：K8S 服务健康检测