# Pod

## centos7 - How to reboot centos in docker? - Stack Overflow <https://stackoverflow.com/questions/54214670/how-to-reboot-centos-in-docker>

How to reboot centos in docker ?

I had pulled centos 7 from docker

use the 'shutdown -r now'

The console said 'Failed to talk to init daemon.'

I also used 'reboot -f' , and get the 'Rebooting. Failed to reboot: Operation not permitted' for rsp

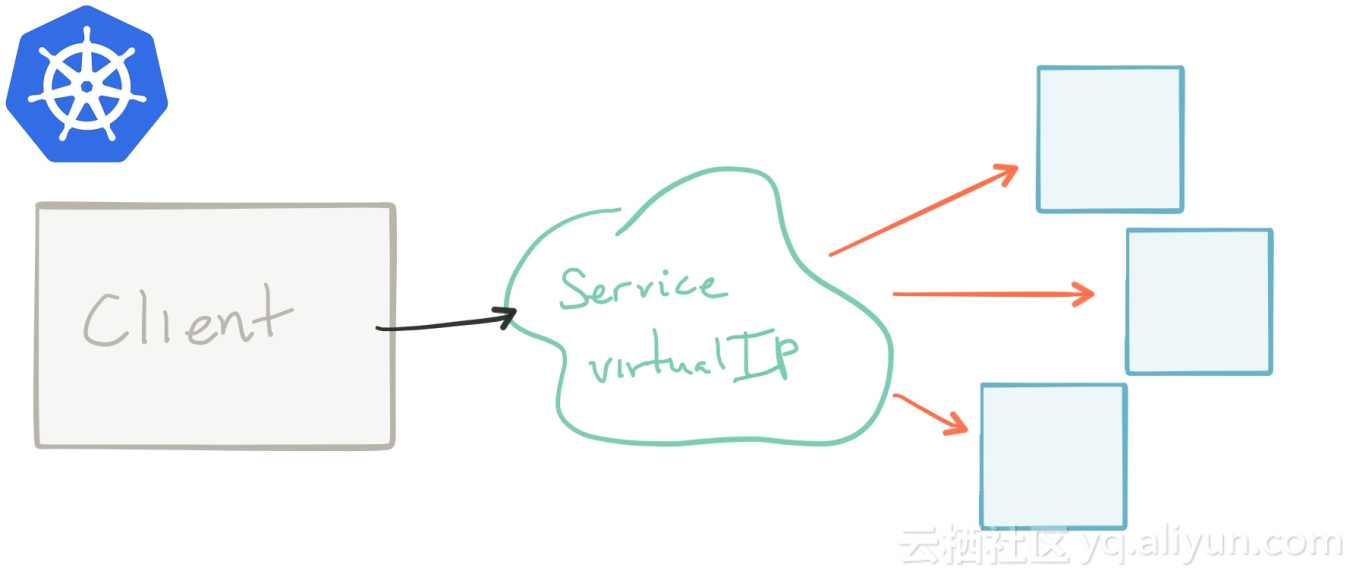
Why you want to restart container?

if it is necessary you can use docker exec -it <container name> reboot or you can stop container and start it again docker stop -f <container name> then docker start <container name>

## Kubernetes之路 3 - 解决服务依赖\_MySQL\_maoreyou的博客-CSDN博客

<https://blog.csdn.net/maoreyou/article/details/80050623>

摘要： 在容器服务的客户群中，一个经常被问起的问题就是如何处理服务间依赖。本文介绍了常见的解决方法来实现服务的依赖检查，还进一步用示例展示了如何利用init container， liveness/readiness探针等技术实现服务健康检查，依赖检查等等功能。



本系列文章记录了企业客户在应用Kubernetes时的一些常见问题

在容器服务的客户群中，一个经常被问起的问题就是如何处理服务间依赖。

在应用中，一个组件依赖指定的中间件服务和业务服务。在传统的软件部署方式中，应用启动、停止都要依照特定的顺序完成。

当采用 Kubernetes/Docker Swarm等容器编排技术在分布式环境下部署应用时，一方面不同组件之间并行启动无法保证其启动顺序，另一方面在应用运行时，其所依赖的服务实现有可能发生失败和迁移。如何解决容器之间的服务依赖就是一个非常常见的问题。

方法1 - 应用端服务依赖检查

我们可以在应用的启动逻辑中添加服务依赖检查逻辑，如果应用依赖的服务不可访问就重试，当错误超过一定次数后就自动退出。Kubernetes/Docker会根据所容器的重启策略(Restart Policy)在等待一段时间之后自动拉起。

下面就是一个简单的Golang应用示例，来检测所依赖的MySQL服务是否就绪。

1. ...
2. // Connect to database.
3. hostPort := net.JoinHostPort(config.Host, config.Port)
4. log.Println("Connecting to database at", hostPort)
5. dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?timeout=30s",
6. config.Username, config.Password, hostPort, config.Database)
8. db, err = sql.Open("mysql", dsn)
9. if err != nil {
10. log.Println(err)
11. }
13. var dbError error
14. maxAttempts := 20
15. for attempts := 1; attempts <= maxAttempts; attempts++ {
16. dbError = db.Ping()
17. if dbError == nil {
18. break
19. }
20. log.Println(dbError)
21. time.Sleep(time.Duration(attempts) \* time.Second)
22. }
23. if dbError != nil {
24. log.Fatal(dbError)
25. }
27. log.Println("Application started successfully.")
28. ...

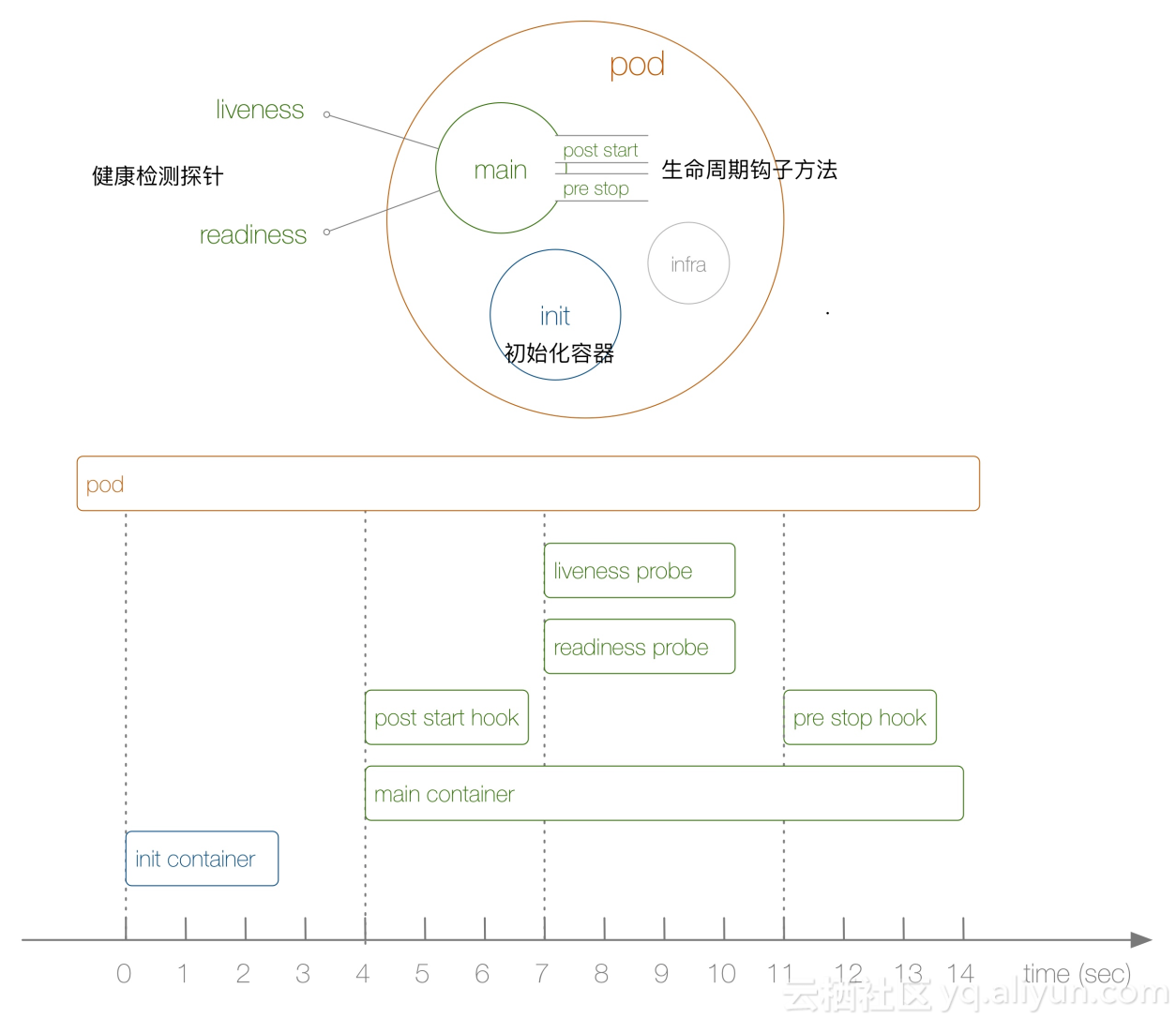
注：   
"Fail Fast" (快速失败)，是Design by Contract契约式设计的一种重要的原则，可以很好地保障系统的健壮性和可预测性。比如上文代码中，如果重试失败，就会由log.Fatal(dbError) 退出执行。而K8S/Docker的容器重启的回退机制可以保障不会因频繁拉起失败应用导致系统资源耗尽。

方法2 - 独立的服务依赖检查逻辑

在现实世界里，有些遗留应用或者框架无法进行调整。我们就会希望将依赖检查策略和应用逻辑进行解耦。

一个常见的方法是在容器的Dockerfile的启动脚本里加入相应的服务依赖检查逻辑，可以参见[Docker文档](https://docs.docker.com/compose/startup-order/)获得更多信息。另一种方法是利用Kubernetes Pod自身机制添加依赖检查逻辑。

首先我们需要对Pod的生命周期有一定的理解，下图来自于 <https://blog.openshift.com/kubernetes-pods-life/> 一文



首先在Pod中有三类容器

* infra container: 这就是著名的pause容器
* init container: [初始化容器](https://kubernetes.io/docs/concepts/workloads/pods/init-containers/) 通常用于应用的初始化准备，只有等所有的初始化容器正常执行完毕之后，才会启动应用容器
* main container: 应用容器

Kubernetes的最佳实践中，通常是利用初始化容器来进行依赖服务的检查。下面我们通过一个Wordpress的实例来展示其使用方法。

1. apiVersion: v1
2. kind: Service
3. metadata:
4. name: mysql
5. spec:
6. clusterIP: None
7. ports:
8. - name: mysql
9. port: 3306
10. selector:
11. app: mysql
12. ---
13. apiVersion: v1
14. kind: Service
15. metadata:
16. name: wordpress
17. spec:
18. ports:
19. - name: wordpress
20. port: 80
21. targetPort: 80
22. selector:
23. app: wordpress
24. type: NodePort
25. ---
26. apiVersion: apps/v1
27. kind: StatefulSet
28. metadata:
29. name: mysql
30. spec:
31. selector:
32. matchLabels:
33. app: mysql
34. serviceName: mysql
35. replicas: 1
36. template:
37. metadata:
38. labels:
39. app: mysql
40. spec:
41. containers:
42. - name: mysql
43. image: mysql:5.7
44. env:
45. - name: MYSQL\_ALLOW\_EMPTY\_PASSWORD
46. value: "true"
47. livenessProbe:
48. exec:
49. command: ["mysqladmin", "ping"]
50. initialDelaySeconds: 30
51. periodSeconds: 10
52. timeoutSeconds: 5
53. readinessProbe:
54. exec:
55. # Check we can execute queries over TCP (skip-networking is off).
56. command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]
57. initialDelaySeconds: 5
58. periodSeconds: 2
59. timeoutSeconds: 1
60. ---
61. apiVersion: apps/v1
62. kind: Deployment
63. metadata:
64. name: wordpress
65. spec:
66. replicas: 1
67. selector:
68. matchLabels:
69. app: wordpress
70. template:
71. metadata:
72. labels:
73. app: wordpress
74. spec:
75. containers:
76. - name: wordpress
77. image: wordpress:4
78. ports:
79. - containerPort: 80
80. env:
81. - name: WORDPRESS\_DB\_HOST
82. value: mysql
83. - name: WORDPRESS\_DB\_PASSWORD
84. value: ""
85. initContainers:
86. - name: init-mysql
87. image: busybox
88. command: ['sh', '-c', 'until nslookup mysql; do echo waiting for mysql; sleep 2; done;']

yum -y install bind-utils

until语句 ：提供循环执行,它与while循环相反， until循环将反复执行直到条件为真就结束循环，而while循环条件为假就结束循环。

使用格式：

until [ 测试条件 ]

do

语句 #可以是多条语句

done

例1:

#!/bin/sh

var=10

until [ $var -gt 1 ]

do

echo $var

var=$(($var - 1))

done

exit 0

例2：

#!/bin/bash

#filename:hk

echo "This example is for testuntil....do "

echo "If you input [exit] then quitthe system "

echo -n "please input:"

read EXIT

until [ $EXIT = "exit" ]

do

read EXIT

done

echo "OK!"

shell脚本中判断上一个命令是否执行成功

shell中使用符号“$?”来显示上一条命令执行的返回值，如果为0则代表执行成功，其他表示失败。

结合if-else语句实现判断上一个命令是否执行成功。

示例如下：

if [ $? -ne 0 ]; then

echo "failed"

else

echo "succeed"

fi

或者：

if [ $? -eq 0 ]; then

echo "succeed"

else

echo "failed"

fi

我们在Wordpress Deployment的Pod定义中添加了initContainers，它会通过检查 mysql 域名是否可以解析来判断所依赖的mysql服务是否就绪。

同时，在MySQL StatefulSet中我们也引入了readinessProbe 和 livenessProbe探针，它们会判定是否MySQL进程已经业务就绪。在K8S中，只有健康的Pod才可以通过ClusterIP访问或者DNS解析。

1. $ kubectl create -f wordpress.yaml
2. service "mysql" created
3. service "wordpress" created
4. statefulset "mysql" created
5. deployment "wordpress" created
6. $ kubectl get pods
7. NAME READY STATUS RESTARTS AGE
8. mysql-0 0/1 Running 0 5s
9. wordpress-797655cf44-w4p87 0/1 Init:0/1 0 5s
10. $ kubectl get pods
11. NAME READY STATUS RESTARTS AGE
12. mysql-0 1/1 Running 0 11s
13. wordpress-797655cf44-w4p87 0/1 Init:0/1 0 11s
14. $ kubectl get pods
15. NAME READY STATUS RESTARTS AGE
16. mysql-0 1/1 Running 0 14s
17. wordpress-797655cf44-w4p87 0/1 PodInitializing 0 14s
18. $ kubectl get pods
19. NAME READY STATUS RESTARTS AGE
20. mysql-0 1/1 Running 0 17s
21. wordpress-797655cf44-w4p87 1/1 Running 0 17s
22. $ kubectl describe pods wordpress-797655cf44-w4p87
23. ...

注：

* Liveness探针：主要用于判断Container是否处于运行状态，比如当服务死锁或者响应缓慢等情况。
* Readiness探针：主要用于判断服务是否已经正常工作。
* 在init container中不允许使用readiness探针。
* 如果Pod重启了，其所有init Container都需重新运行。

总结

本文介绍了常见的解决方法来实现服务的依赖检查，还进一步用示例展示了如何利用init container， liveness/readiness探针等技术实现服务健康检查，依赖检查等等功能。

Kubernetes提供了非常灵活的Pod生命周期管理机制，由于篇幅有限我们就不再展开介绍 postStart/preStop等生命周期钩子方法。

[阿里云Kubernetes服务](https://www.aliyun.com/product/kubernetes) 全球首批通过Kubernetes一致性认证，简化了Kubernetes集群生命周期管理，内置了与阿里云产品集成，也将进一步简化Kubernetes的开发者体验，帮助用户关注云端应用价值创新。

[**原文链接**](http://click.aliyun.com/m/46855/)