

kubernetes容器云平台迁移实践



李志伟 2017年7月

为什么选择容器云平台

面临的问题：

- ▶ 基础设施故障率高，运维自动化水平低
- ▶ 系统架构不一致，维护复杂度高
- ▶ 计算资源利用率低
- ▶ 业务迁移或水平扩展困难。
- ▶ 服务总量繁多，管理复杂度高。



我们的目标



- ▶ 一个高效的服务编排平台
- ▶ 一个高可用的运维平台
- ▶ 一个高度的自动化运维平台

为什么选择Kubernetes

- ▶ 提高计算资源的利用率和服务的弹性扩展能力。
- ▶ 业务容器镜像一次构建，可运行在多种环境，配合CI/CD,可大幅提升研发效率。
- ▶ 消除本地依赖，迁移成本低，解决服务商锁定问题。
- ▶ 遵循Borg架构思想，规范网站系统架构设计。
- ▶ 实现运维自动化



kubernetes
by Google

向容器云平台迁移前的准备工作

- ▶ 搭建kubernetes集群
- ▶ 构建私有Docker镜像仓库（ Harbor ）
- ▶ 集群监控Heapster
- ▶ CI/CD基础设施（ Jenkins ）
- ▶ 分布式存储解决方案（ Glusterfs ）

组件选型

- ▶ Kubernetes 1.5.2 主程序
- ▶ Docker 1.10.3 容器
- ▶ Flannel 0.7.0 网络组件
- ▶ Etcd 3.1.3 数据库
- ▶ Kubernetes-Dashboard UI
- ▶ KubeDNS DNS组件
- ▶ Registry 容器私有镜像库
- ▶ Heapster 监控
- ▶ GlusterFS 3.7.9 共享存储

业务迁移中使用的规范

目录：

- 容器镜像封装的基本原则
- NameSpace的使用规范
- service name的命名规范
- 健康检查规范
- Image tag配置规范
- 使用ConfigMap

容器镜像封装的基本原则

- ▶ 尽可能地设计成无状态服务
- ▶ 尽可能消除不必要的运行环境依赖
- ▶ 需要持久化的数据写入到共享存储
- ▶ 尽可能保持业务的单一性
- ▶ 控制输出到stdout和stderr的日志写入量。
- ▶ 配置与镜像内容分离
- ▶ 容器中使用k8s内部dns代替ip地址配置形式
- ▶ 日志采用集中化处理方案（EFK）
- ▶ 采用独立的容器处理定时任务

关于Namespace的使用

- ▶ 实现在一个集群内部同时运行开发、测试、预发、生产环境
- ▶ 软件组件在不同的运行环境之间不会产生命名冲突。
- ▶ 通过namespace实现资源隔离（Resource Quota）。

关于service name的命名规范

- ▶ v1.5版之前不能超过24个字符, v1.5版后最多63个字符。
- ▶ 需要满足正则regex `[a-z]([-a-z0-9]*[a-z0-9])?` 的要求, 意味着首字母必须是a-z的字母, 末字符不能是-, 其他部分可以是字母数字和-符号。
- ▶ 命名方式: 业务名-应用服务器类型-其他标识, 例如 book-mysql-m1、book-redis-n1、book-tomcat-n1、book-zk-n1

关于应用健康检查

健康检查是系统故障自动发现和自我恢复最重要的手段

一、进程级健康检查

- ▶ 检验容器进程是否存活，进程级的健康检查都是默认启用的

二、业务级健康检查：

- I. 检查自身核心业务是否正常
- II. 健康检查程序执行时间要小于健康检查周期
- III. 健康检查程序消耗资源要合理控制，避免出现服务抖动

关于健康检查程序的实现

- ▶ **WEB服务**：采用HTTPGET方式进行健康检查，需要实现一个“/healthz” URI，这个URI程序需要检查所有核心业务服务是否正常，健康检查程序还应该在异常情况下输出每一个检查项的状态明细。
- ▶ **其他网络服务**：可以采用探查容器的指定端口状态来判断容器健康状态。
- ▶ **非网络服务**：在容器内部执行特定命令，根据退出码判断容器健康状态。

HttpGetProb	TCP Socket Action	Exec Action
livenessProbe: httpGet: path: /healthz port: 8080 initialDelaySeconds: 15 timeoutSeconds: 1	livenessProbe: initialDelaySeconds: 15 timeoutSeconds: 1 tcpSocket: port: 80	livenessProbe: exec: command: - cat - /tmp/health initialDelaySeconds: 15 timeoutSeconds: 1
范例1	范例2	范例3

.yaml中Image tag配置规范

- ▶ 部署容器镜像时避免使用:latest tag形式
 - 难以跟踪当前运行的images版本
 - 难以进行回滚操作
 - imagePullPolicy为IfNotPresent时镜像不会更新。
- ▶ 每个容器images的tag应该用版本号来标识（如 myimage:v1）

使用ConfigMap实现应用平滑迁移

配置应该与镜像内容分离以保持容器化应用程序的可移植性

使用场景：

- ▶ 填充环境变量的值
- ▶ 设置容器内的命令行参数
- ▶ 填充卷的配置文件

迁移中遇到的其他问题

关于CI/CD

- ▶ 采用Jenkins实现CI/CD
- ▶ 降低开发人员使用K8S的门槛
- ▶ CI/CD提升研发效率

Docker-in-Docker on Kubernetes

- 在docker中运行docker，实质上是将host的docker.sock挂载到容器中，在容器中创建的容器，其实是运行在host机器上。

```
apiVersion: v1
kind: Pod
metadata:
  name: dood
spec:
  containers:
    - name: docker-cmds
      image: docker:1.12.6
      command: ['docker', 'run', '-p', '80:80', 'httpd:latest']
      resources:
        requests:
          cpu: 10m
          memory: 256Mi
      volumeMounts:
        - mountPath: /var/run
          name: docker-sock
  volumes:
    - name: docker-sock
      hostPath:
        path: /var/run
```

时区的配置问题

- ▶ 容器镜像的/etc/localtime根据需要设置为对应的时区 如：
/usr/share/zoneinfo/Asia/Shanghai。
- ▶ 采用 配置文件中的volume挂载宿主机对应的localtime文件方式
或使用ConfigMap方式实现。
- ▶ 配置代码：

volumes:

- name: local-time

hostPath:

path: /usr/share/zoneinfo/Asia/Shanghai

readOnly: true

volumeMounts:

- name: local-time

mountPath: /etc/localtime

readOnly: true

外部网络如何访问到Service

方案一

- ▶ 在集群内部搭建Nginx容器服务负责集群内部service的对外转发
- ▶ Nginx服务通过NodePort方式暴露给外部网络

方案二

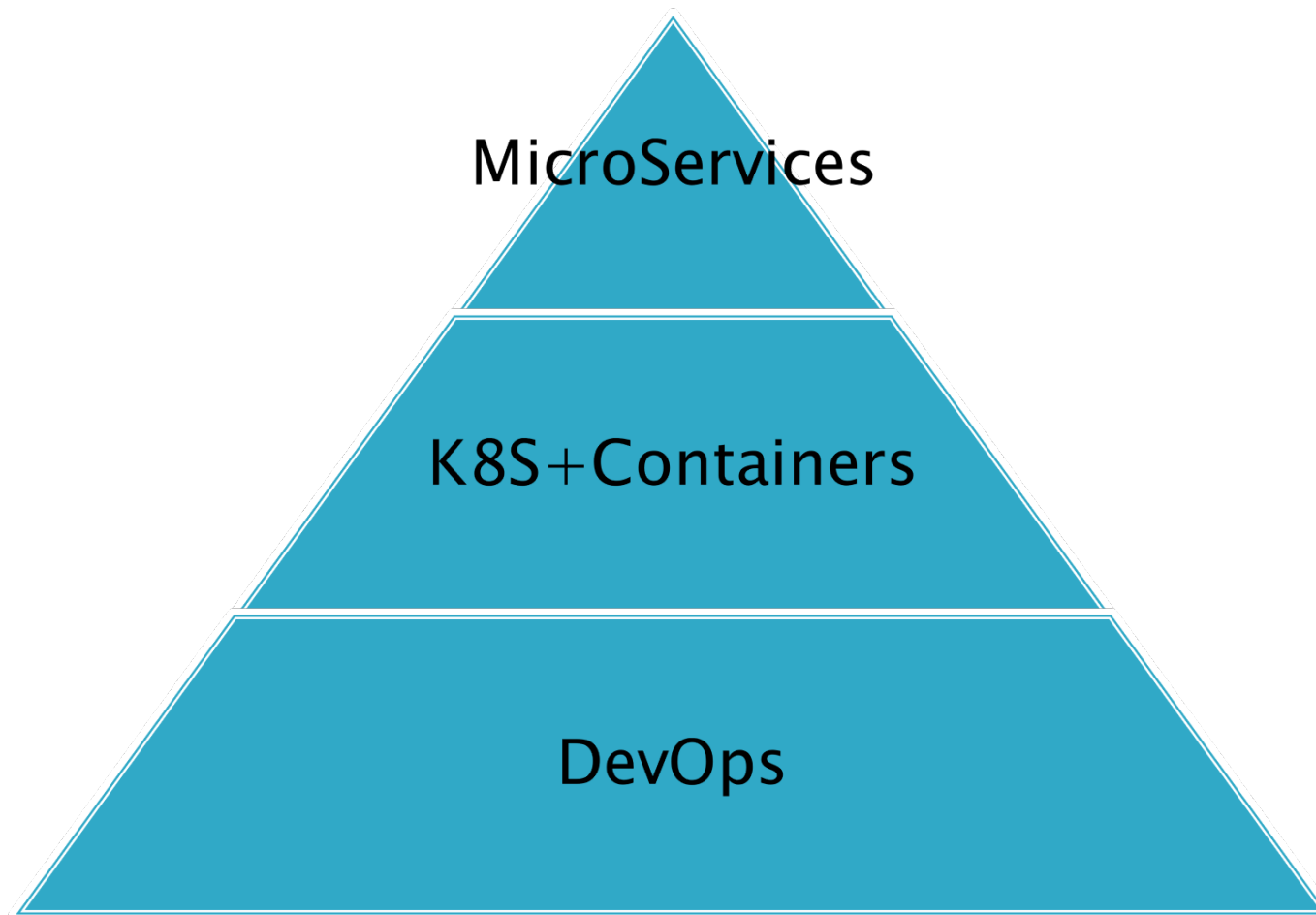
通过Ingress转发集群内部服务给外部网络

网络问题：关于nf_conntrack: table full, dropping packet 的问题

- ▶ ip_conntrack 是 Linux 系统内 NAT 的一个跟踪连接条目的模块。ip_conntrack 模块会使用一个哈希表记录 tcp 通讯协议的 established connection 记录，当这个哈希表满了的时候，便会导致 nf_conntrack: table full, dropping packet 错误。

```
# vi /etc/sysctl.conf
#哈希表项最大值
net.netfilter.nf_conntrack_max = 655350
#超时时间，默认情况下 timeout 是5天（432000秒）
net.netfilter.nf_conntrack_tcp_timeout_established = 1200
```

最佳组合



谢谢
Q&A