2017 China Kubernetes End User Conference
KEUC
kubernetes 中国用户大会 —2017—
2017.10.15 / 中国·杭州

kubernetes容器云平台迁移实践

李志伟 2017年10月

主办：CLOUD NATIVE COMPUTING FOUNDATION caicloud

目录

现有平台面临的挑战

- 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
- 业务迁移或扩展困难
- 服务总量繁多，管理复杂度高

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。
- 消除本地依赖，降低迁移成本，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

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/Helm）
- ✓ 分布式存储解决方案（ Glusterfs ）

组件选型

- Kubernetes v1.5.2　　主程序
- Docker v1.10.3　　容器
- Flannel　v0.7.0　　网络组件
- Etcd　　3.1.3　　数据库
- Kubernetes-Dashboard　　UI
- Kubedns　DNS组件
- Harbor　　容器私有镜像库 (Registry)
- Heapster监控
- GlusterFS　v3.7.9　　共享存储
- Jenkins v2.67　　CI/CD工具
- Helm v2.5.0　　package manager

业务迁移中使用的规范

- 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

容器镜像封装的基本原则

一. 尽可能地设计成无状态服务

二. 尽可能消除不必要的运行环境依赖

三. 需要持久化的数据写入到共享存储

四. 尽可能保持业务的单一性

五. 控制输出到stdout和stderr的日志写入量。

六. 配置与镜像内容分离

七. 容器中使用k8s内部dns代替ip地址配置形式

八. 日志采用集中化处理方案（EFK）

九. 采用独立的容器处理定时任务

NameSpace的使用

I. 实现在一个集群内部同时运行开发、测试、Staging、生产环境

II. 软件组件在不同的运行环境之间不会产生命名冲突。

III. 通过namespace实现资源隔离（Resource Quota） 。

Service name的命名规范

一. v1.5版之前不能超过24个字符，v1.5版后最多63个字符。

二. 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。

三. 命名方式：业务名-应用服务器类型-其他标识 ，例如 book-tomcat-n1 、book-mysql-m1 、book-redis-n1、book-zk-n1

应用健康检查规范

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

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对应的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

```
livenessProbe:
      httpGet:
        path: /healthz
        port: 8080
      initialDelaySeconds: 15
      timeoutSeconds: 1

readinessProbe:
      httpGet:
        path: /readiness
        port: 8080
      initialDelaySeconds: 5
      timeoutSeconds: 1
```

范例1

TCP Socket Action

```
livenessProbe:
    initialDelaySeconds: 15
    timeoutSeconds: 1
    tcpSocket:
      port: 80
```

范例2

Exec Action

```
livenessProbe:
      exec:
        command:
        - cat
        - /tmp/health
      initialDelaySeconds: 15
      timeoutSeconds: 1
```

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**

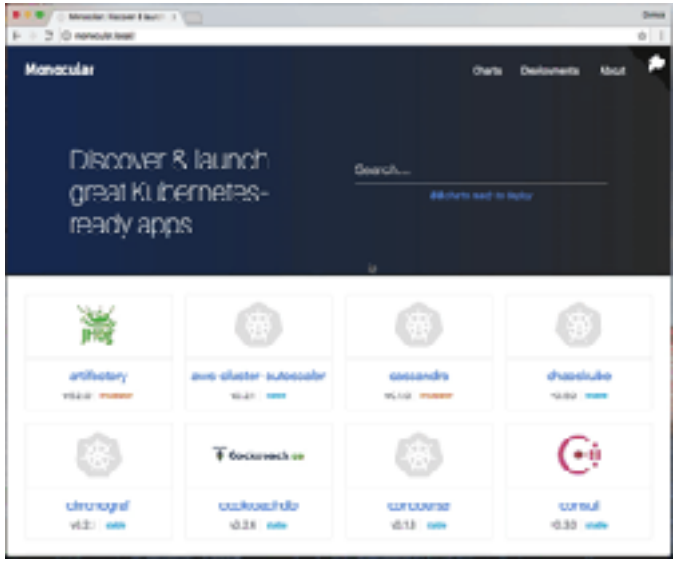
迁移中遇到的其他问题

- 关于CI/CD
- Docker-in-Docker
- 时区的配置问题
- 外部网络访问Service
- 网络问题

关于CI/CD

- 采用Jenkins实现CI/CD
- 通过Helm实现软件包管理

```
edanqdanq-hapi
├──charts
├──Chart.yaml
├──templates
│   ├──configmap.yaml
│   ├──deployment.yaml
│   ├── helpers.tpl
│   ├──ingress.yaml
│   ├──NOTES.txt
│   ├──pvc.yaml
│   └──service.yaml
└──values.yaml
```



Monocular

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，实际上是运行在host机器上。

```
apiVersion: v1
kind: Pod
metadata:
    name: dind
spec:
    containers:
        name: docker cmds
        image: docker:1.12.6
        command: ['docker', 'run', '-p', '80:80', 'httpd:latest']
        resources:
            requests:
                cpu: "1"
                memory: 256Mi
        volumeMounts:
          - mountPath: /var/run
            name: docker sock
    volumes:
        name: docker sock
        hostPath:
            path: /var/run
```

时区的配置问题

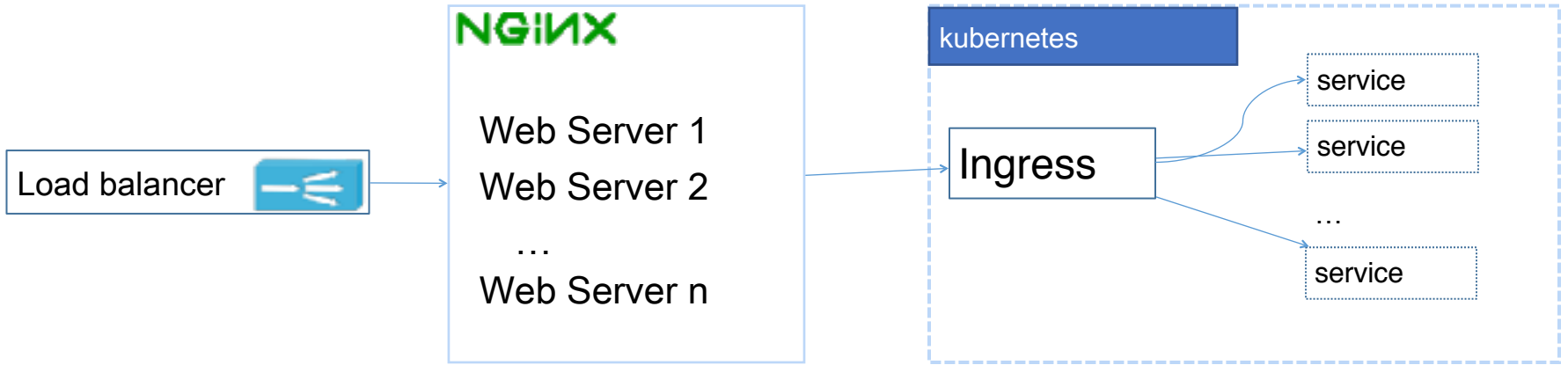
- 容器镜像的/etc/localtime根据需要设置为对应的时区 如：/usr/share/zoneinfo/Asia/Shanghai
- 采用 配置文件中的volume挂载宿主机对应的localtime文件方式
- 配置代码：

```
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

- 通过Ingress转发集群内部服务
- Ingress服务通过NodePort方式暴露给外部网络

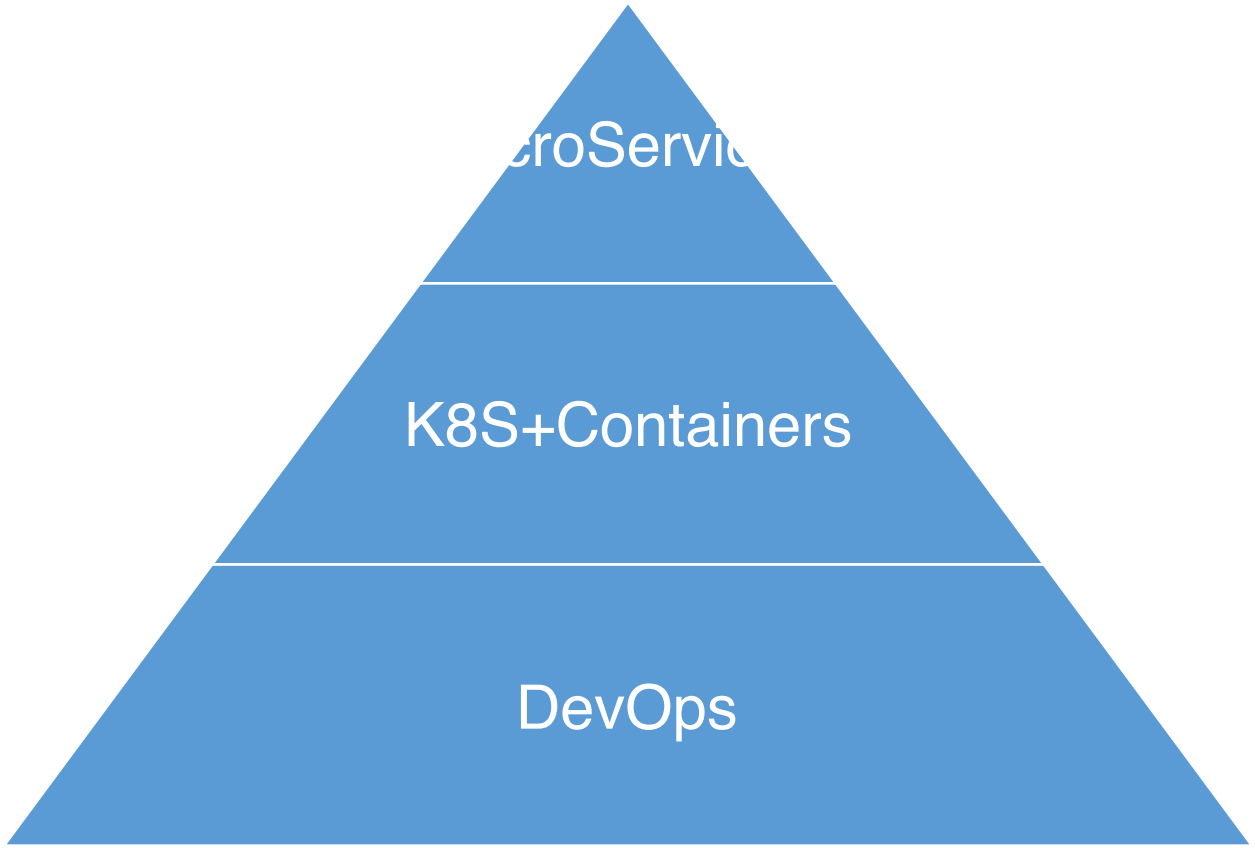


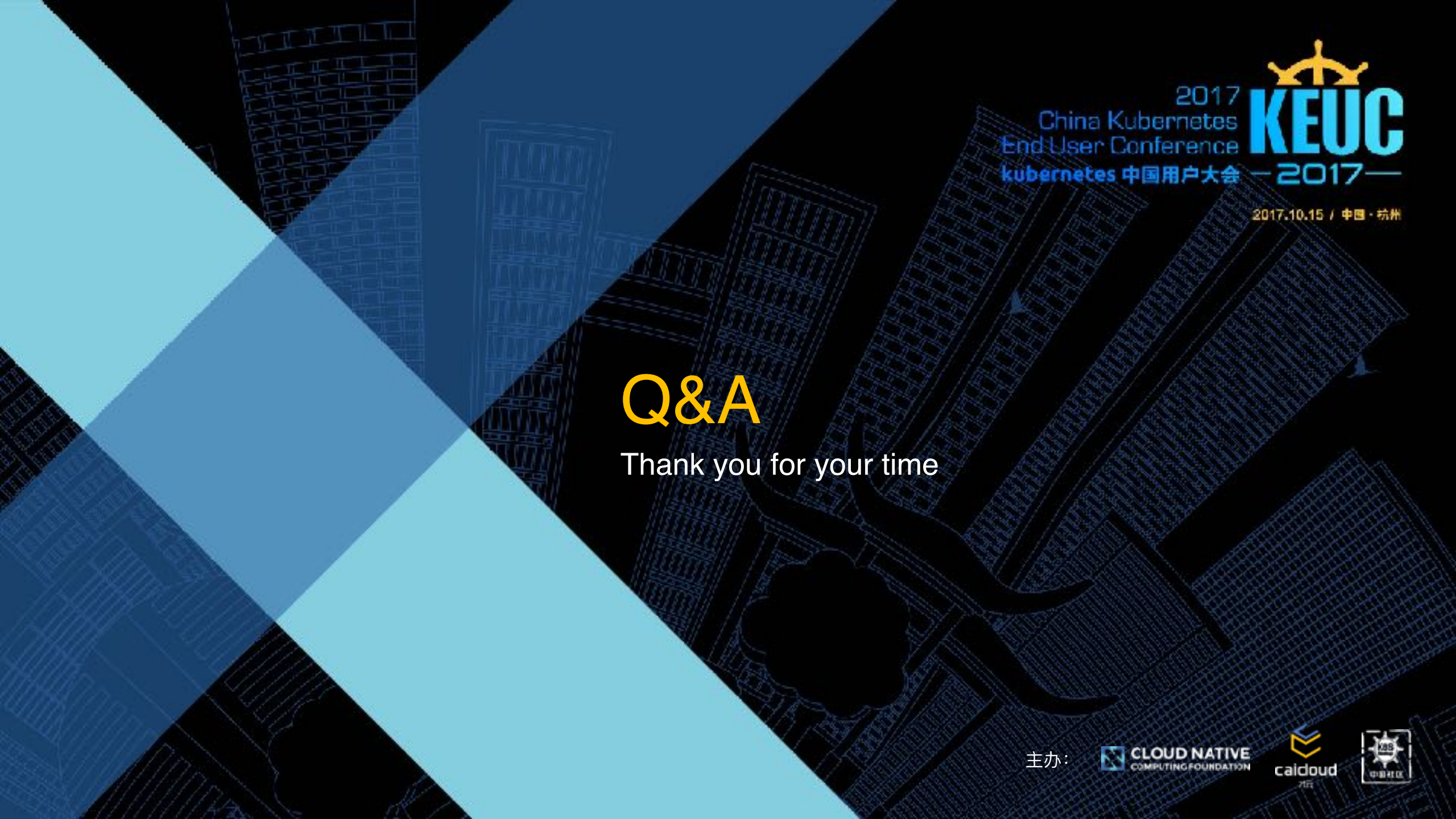
网络问题：关于nf_conntrack: table full, dropping packet

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 。

```
# 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





Q&A

Thank you for your time

主办：

CLOUD NATIVE COMPUTING FOUNDATION

caicloud