

Cloud Native Application

演讲者 | 王渊命 @jolestar

Traditional vs Cloud Native Application



Scale up

Servers are like pets.
Pets are given names, are unique, lovingly hand raised and cared for. When they get ill, you nurse them back to health.

VS



Scale out

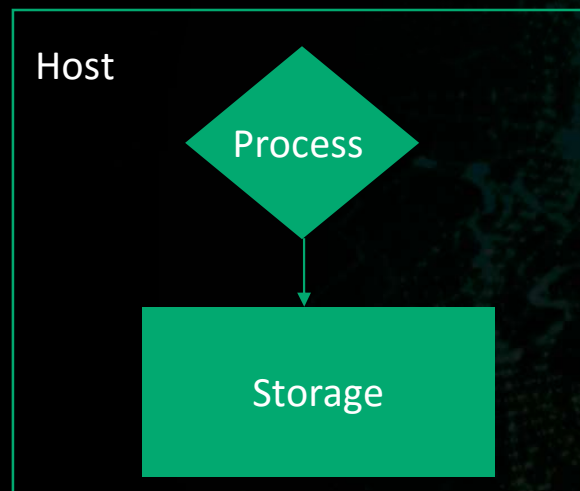
Servers are like cattle.
Cattle are given numbers and are almost identical to each other.
When they get ill, you get another one.

“Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed.”

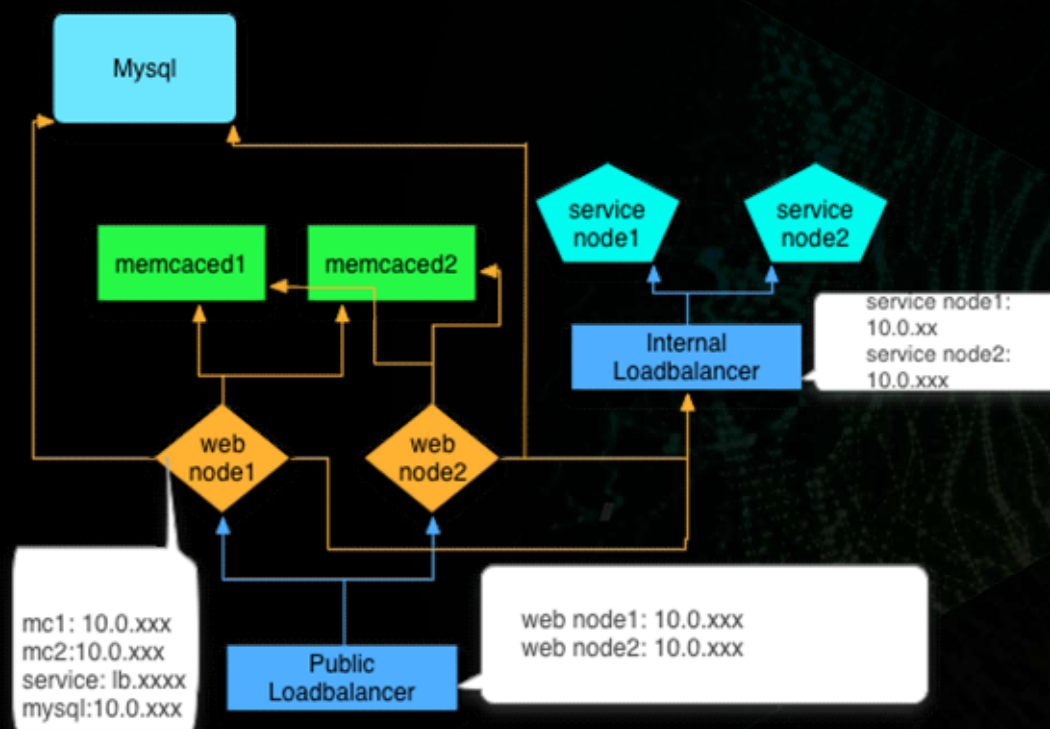
- Tim Bell, CERN

互联网应用的演进

单机应用



分布式集群应用



op tools(ansible/puppet)

需求

弹性
伸缩

高可用

故障
恢复

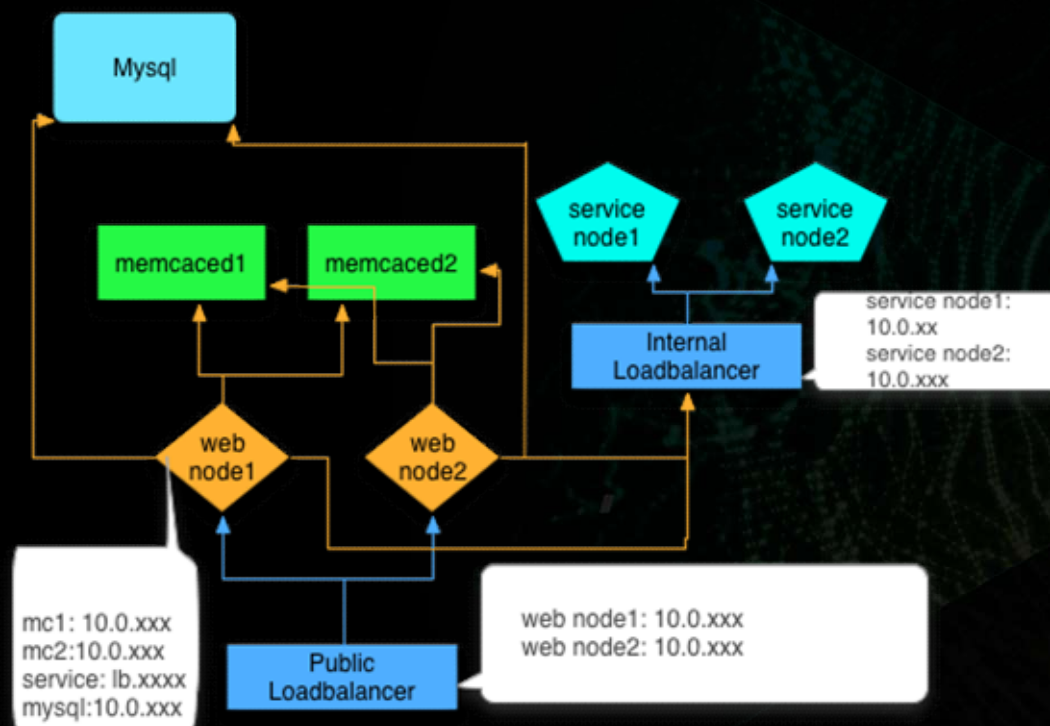
弹性伸缩 - 第一步

- ▶ 更多的机器
- ▶ 更便捷的方式购买计算和存储资源

IaaS 应运而生

- ▶ 接管硬件资源的运维
- ▶ 提供可编程接口来购买管理资源
- ▶ 提供SDN模拟硬件网络
- ▶ 提供SDS模拟硬件存储
- ▶ 对应用无侵入
- ▶ 面向资源
- ▶ 资源层面的容灾

达到目标了么？



op tools(ansible/puppet)

分布式系统之痛

- ▶ 开发测试复杂
- ▶ 管理机器（虚拟机或者物理机）
- ▶ 管理节点（机器上的进程）
- ▶ 异构的主机环境以及网络环境
- ▶ 主机上进程间的冲突（端口，文件系统）
- ▶ 日志收集/状态监控
- ▶ 服务之间的依赖

当前写分布式系统应用的复杂度相当于没有线程库的时候写并发程序，
甚至是没有操作系统的时候写应用程序。

——by 老王

分布式操作/调度系统（DCOS）

- ▶ 管理主机
- ▶ 接管分布式系统的调度层
- ▶ 调度节点（机器上的进程）
- ▶ 面向应用 屏蔽资源细节
- ▶ 侵入应用

应用与云的关系

- ▶ 应用需要依赖云提供的能力，让渡一部分职能
- ▶ 云需要理解应用以实现更好的调度以及容灾

几个例子

► Yarn

► Mesos

► Kubernetes

Cloud Native Application

让渡一部分功能给云，以实现弹性，高可用，故障恢复，降低研发运维成本的应用

具体表现

- ▶ Devops
- ▶ Package & Dependency
- ▶ Config
- ▶ Log
- ▶ Service Discovery/Dependency
- ▶ Service Architecture
- ▶ SelfManager

DevOps

Traditional	Cloud Native
CI/CD 很难标准化 (脚本大法)	CI/CD 标准化
单主机混合部署多个服务	单隔离单元 (vm/container) 部署一个服务
应用代码和部署脚本分离	应用代码和部署脚本一体, 部署结构的描述本身是代码的一部分
集群自然生长而成, 缺少变更追踪机制	集群的变更有严格追踪, 可实现一键clone

Package & Dependency (安装包以及依赖)

Traditional	Cloud Native
os / 语言相关	镜像
依赖主机环境	

Config (配置)

Traditional	Cloud Native
集群相关配置	中心化 (etcd/zookeeper)
实例相关配置	启动参数化/镜像模板
依赖服务相关配置	集群的依赖服务注入

Log (日志)

Traditional	Cloud Native
本地文件	进程默认输出，进程管理器收集
	集群收集 中心化展示
	格式化 事件化

Service Discovery/Dependency (服务发现/依赖)

Traditional	Cloud Native
静态配置	配置中心
	集群API

Service Architecture (服务架构)

Traditional	Cloud Native
Monolithic	Microservice

SelfManager (自管理)

```
int main(void)
{
    pid_t pid = fork();

    int status;
    (void)waitpid(pid, &status, 0);

    return EXIT_SUCCESS;
}
```

```
int main(void)
{
    nid_t nid = fork_node();

    int status;
    (void)waitnid(nid, &status, 0);

    return EXIT_SUCCESS;
}
```

问题解决了么？

Cloud Native Application

► 标准

► 成熟度

► 生态

当前能做什么

- ▶ 架构应用的时候充分考虑以后的迁移方案
- ▶ 尽量借鉴 Cloud Native Application 的思想，技术变革最难变革的是人的思维方式和做事习惯

青云QingCloud 的容器策略

- ▶ 支持主流开源的容器调度平台
- ▶ 让容器平台更方便的运行在青云之上
- ▶ 通过 AppCenter 对第三方开放

关注我们



 QingCloud-IaaS

 青云QingCloud

www.qingcloud.com

QING
CLOUD
INSIGHT
2016 科技洞见未来

Thank you.

@jolestar

QING
CLOUD
INSIGHT
2016 科技洞见未来