Eru: 基于Docker的调度编排系统

张晔

2017/03/13

概述

- 发展历程
- Eru基本架构
- 核心调度算法: 需求与实现

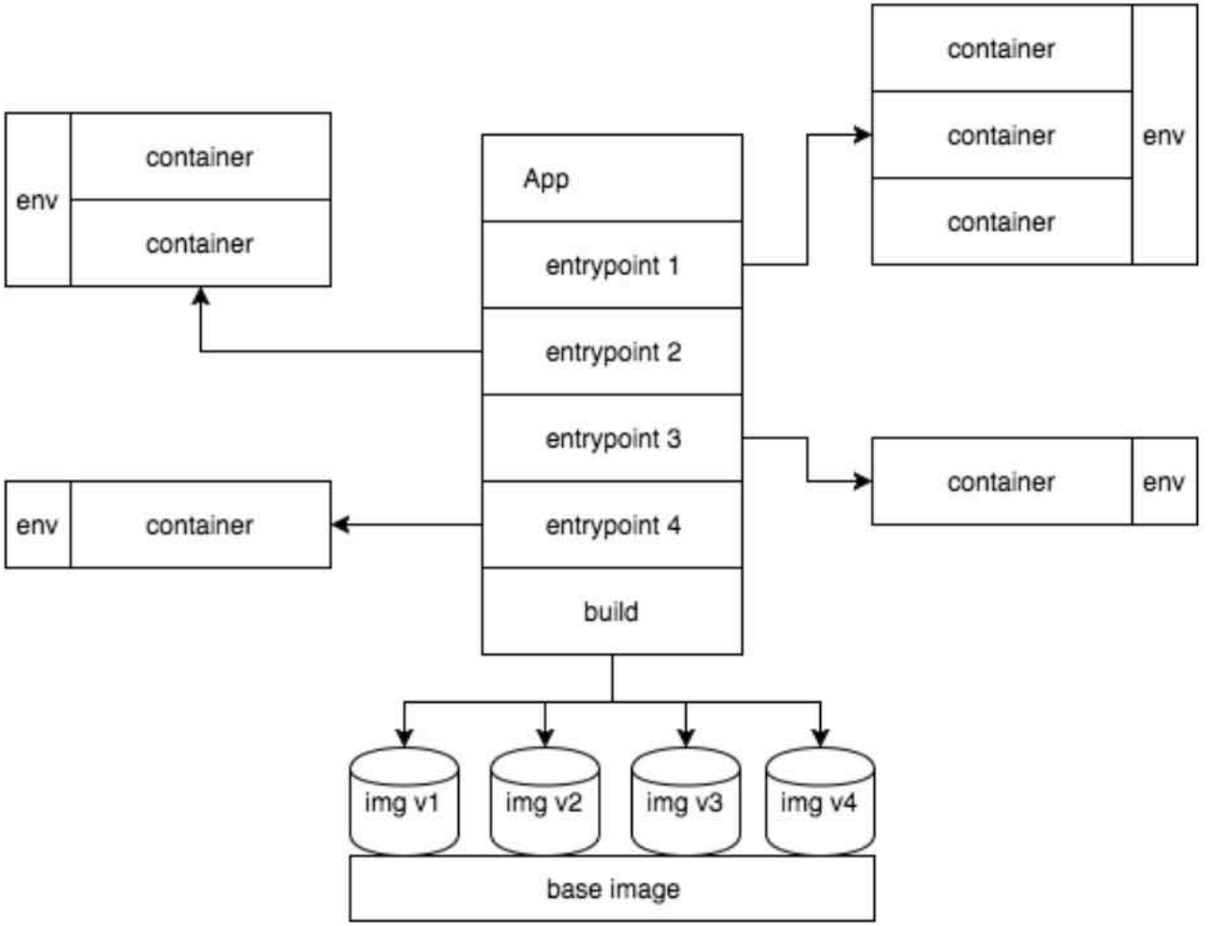
发展历史

- Version 1 (Python 实现);
- Version 2 (以 Go 为主);

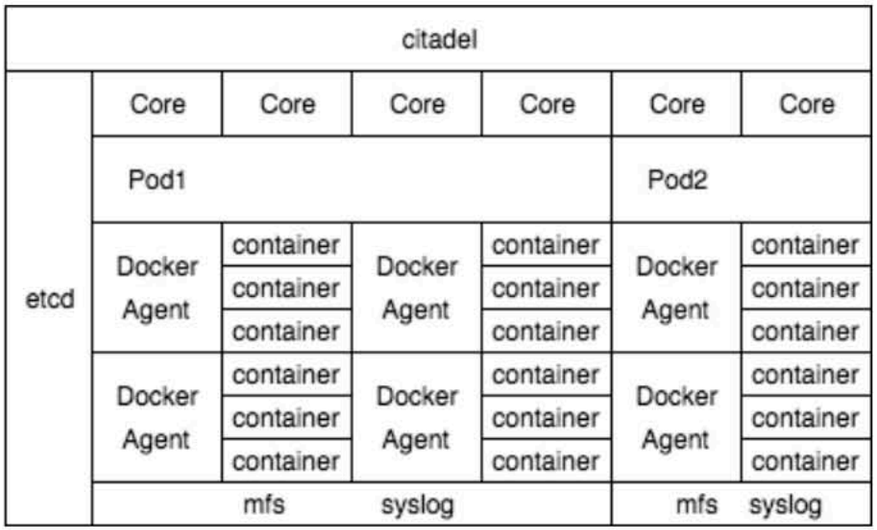
Eru 主要组件

- eru-core;
- agent;
- 前端UI系统 citadel;
- 其他(监控、日志收集等);

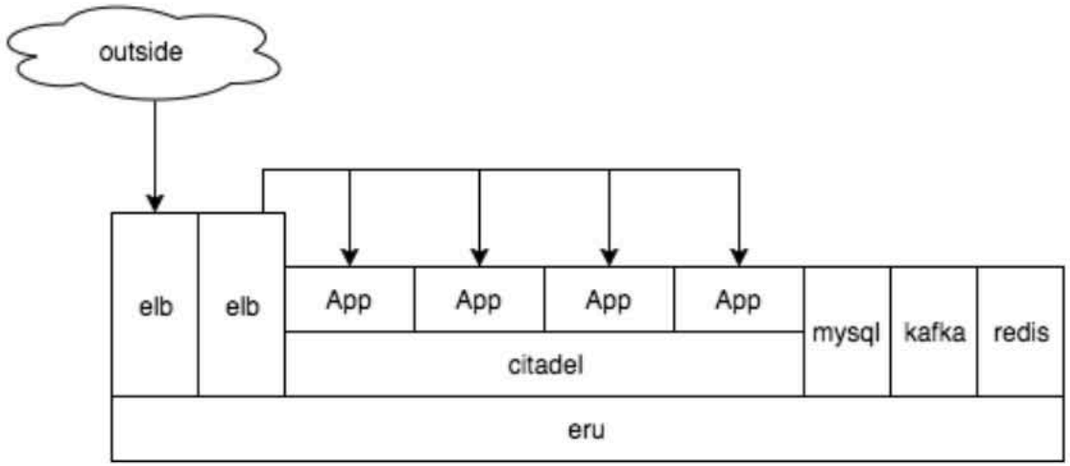
Eru 架构: App


container
container
container
env
App
entrypoint 1
entrypoint 2
entrypoint 3
entrypoint 4
build
env
container
container
env
container
container
env
img v1
img v2
img v3
img v4
base image

Eru 架构: 物理架构


citadel
etcd
Core
Core
Core
Core
Core
Core
Pod1
Pod2
Docker Agent
container
Docker Agent
container
Docker Agent
container
Docker Agent
container
Docker Agent
container
Docker Agent
container
mfs
syslog
mfs
syslog

Eru 架构: 业务层面


outside
elb
elb
App
App
App
App
mysql
kafka
redis
citadel
eru

eru-core: 高性能，无状态

Eru 系统核心，接收容器需求，根据已有资源和用户所需资源创建容器。

agent: 低能耗

容器健康检查、指标和日志收集。

citadel: user-friendly UI

鉴权，用户UI，相关API封装。

Eru Load Balancer

动态感知容器，高度灵活的分流规则。

日志收集与监控

[1] 日志收集: 基于 rsyslog, 高性能，多种查询方式。

[2] 监控: 基于 graphite, 可横向扩展，自动报警。

核心调度算法: 需求与场景

- 需求:

  [1] 资源利用率;

  [2] 稳定性;

- 限制条件:

  [1] CPU;

  [2] 内存;

核心调度算法:伪代码

伪代码

0. 寻找有足够资源的节点;
1. 计算每个可用节点的可以分配多少个容器;
2. 平均分配;

核心调度算法:实现

基于资源限制的两种调度算法:

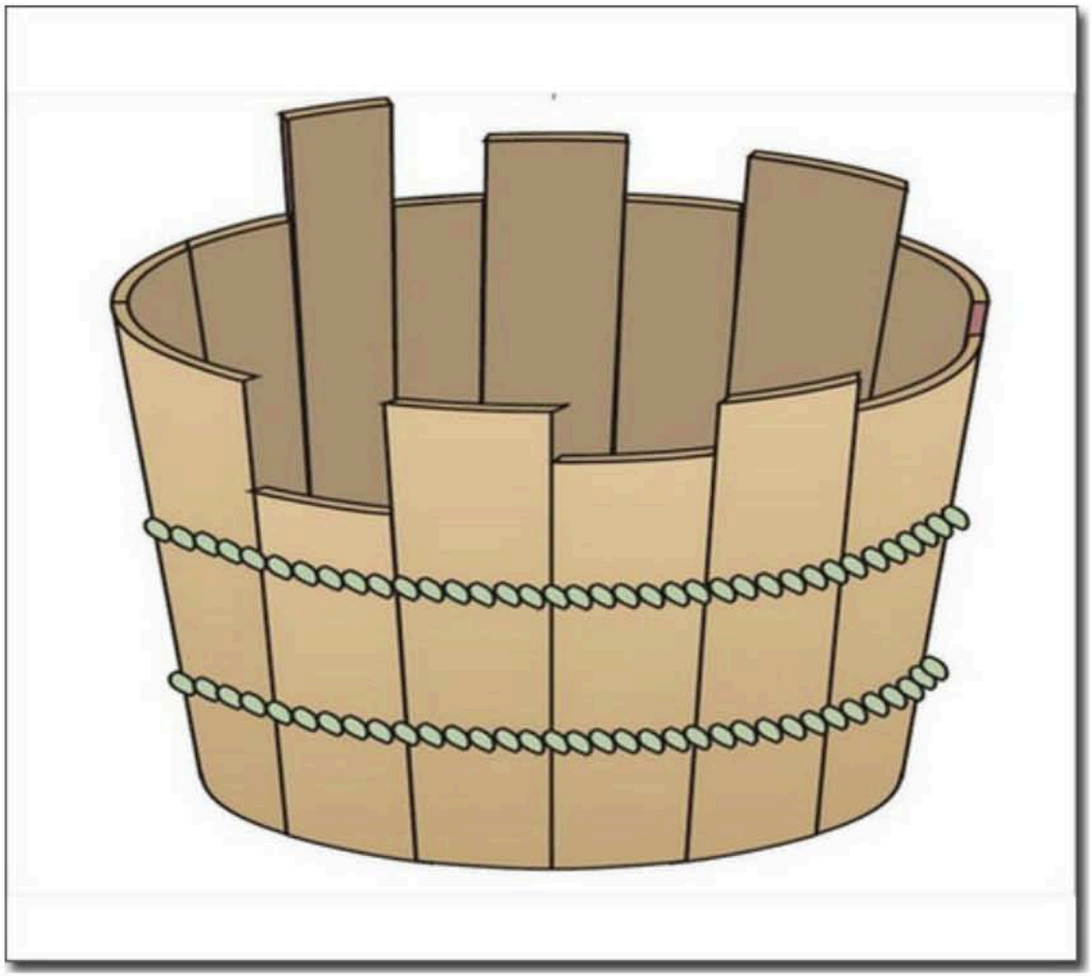
- 基于CPU的算法(**CPU-bind**): redis;
- 基于内存的算法(**MemCap**): Java 应用;

更为精确的调度算法

考虑一种可能的情况：假设我们有3台机器: A, B, C. 因为各种原因，A和B上面个有有1个容器，C上面有10个容器，此时资源分配极为不平衡！此时若我们要继续分配容器。按照前面的思路，将会继续加剧不平衡的状况。因此，我们设计了根据容器当下分布以及当前需求一起进行分配的调度算法。

更为精确的调度算法

这种更为精确的调度算法在实现上就像修补一个木板长短不一的木桶: 我们将节点上容器的个数看作木板的长度，不断地补上短木板的长度，从而让所有木板的长度尽量一致。



招聘时间

诚招：各种工程师(包括但不限于: Java、Python、Go) 有意者请发简历到
zhangye@ricebook.com