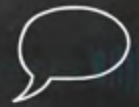
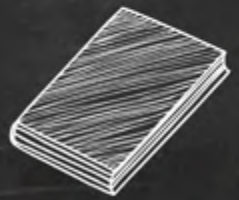
容器化技术在音视频领域的应用





张新宇

tutorabc

自我介绍

张新宇. Dellinger.Zhang

- 十几年IT行业的“老司机”
- 多年互联网行业架构师经验
- 关注在音视频、系统架构方面
- 曾混迹过盛大游戏、沪江网、途牛等互联网企业

欢迎添加微信参与讨论

分享主题

前言

在线音视频现状

5.69亿 2016年在线视频用户

700+亿 2016年市场规模

直播 点播 短视频 …

泛娱乐 教育 游戏 电商 生活 …

在线教育领域

直播课

即时反馈
有效激励

>

录播课

可重复
关注细节

>

媒体课件

理论学习
动手实验

底层技术挑战

互动性

1 + N 实时视频

Lobby模式互动

桌面分享

用户体验

各种实时通讯需求

高效传输、码率控制

移动端优化

可用性

丢包补偿、Qos侦测（抖动&漂移）

服务监控（响应时间、吞吐量…）

网络优化、节点分配...

本地、异地灾备、服务转移…

遇到的麻烦…

服务众多、环境复杂

- 各式各样的配置
- 复杂的网络连接
- 各种版本的软件支持

部署、更新成本

- Patch更新
- Infra (OS, boost, JVM …)
- Scale out/in

整合成本

- 外部服务同步、连接
- Dev, Stage, Pre-Product, Product …
- SQL, No-SQL…

故障恢复

- 问题重现
- 发布回滚
- 现场保留

Solution：容器化

理由一：唯轻、唯快

理由二：大笔成熟案例和支持

成本：要求较高、性能疑问


02
Docker & Tutormeet +

Tutorabc业务特点

强互动性、多人视频

国际化在线教育平台

7 x 24 x 365 模式

TM+系统划分

白板

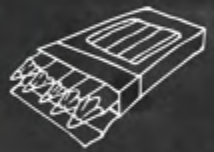
网关

教材

消息

监控

媒体



房间

转码

TM+技术特点

90%　实现了容器化，包含转码、推流服务

主要开发语言Golang & C++　85%

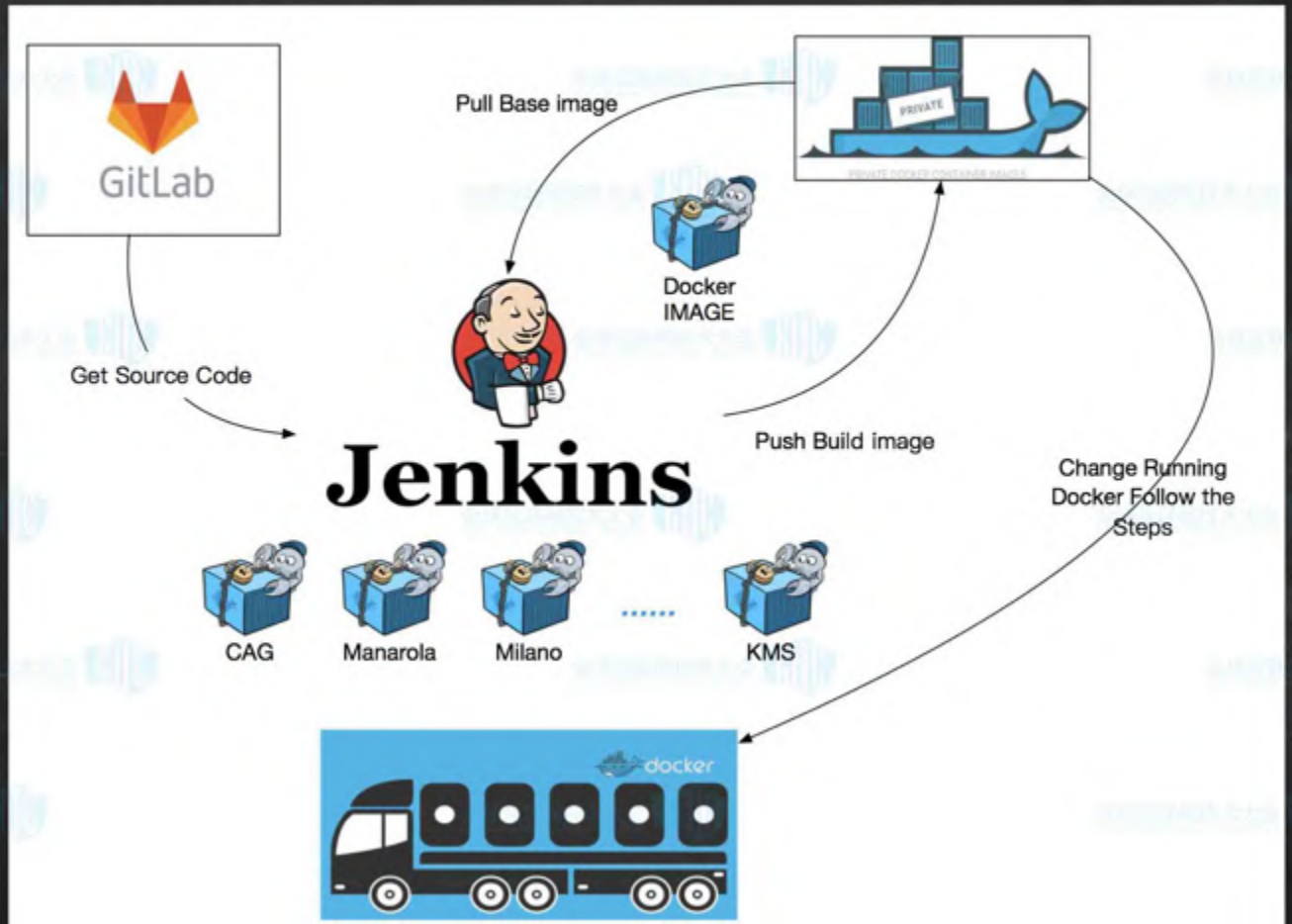
>20%　抗丢包率

PC, Android, ios覆盖　100%

100ms　端到端延迟

支持全高清模式　1080p

TM+系统架构


GitLab
Pull Base image
PRIVATE
Docker
IMAGE
Get Source Code
Jenkins
Push Build image
Change Running
Docker Follow the
Steps
CAG
Manarola
Milano
.....
KMS
docker

TM+的环境分配

Dev farm

开发
- 前端
- 后端
- 媒体

压测

Stage farm

BETA验证
分支验证

Pro-Product

灰度验证

Product

线上及Backup

TM+系统架构


Chrome
Class Room
Android
Pad
iOS
Pad
External System
Service & Application
Load Balance
OpenResty
Application Layer
Golang
Global Service API Service
TutorConsole+
web app
Third Service
API Service
Basic Service Layer (Manarola & Milano)
Service
Trans-codec
Service
Live Stream Push
Golang
Whiteboard
Golang
PPT
Service
File Recorder
Service
API Service
Golang
Signal & IM
Nginx
TutormeetPlus.com
Nginx
TutormeetPlus.com
Unit A
Docker
Docker
Cagliari
TMS
Unit B
Docker
Docker
Cagliari
TMS
Infrastructure Layer
Service
File Storage
Redis
Cluster
Elasticsearch
Log Cluster
MySQL
Master
MySQL
Slave

TM+相关的开源项目

TutorMeet+

TM+系统监控


TM_Plus | System Overview
System Uptime
8.3 weeks
Virtual CPUs
32
RAM
62.80 GiB
Memory Available
94%
CPU Usage
Load Average
Memory
Memory Distribution
Forks
Processes
Docker Dashboard
Memory usage
18%
CPU usage
0.12%
Filesystem usage
0.95%
Container CPU usage
Container Memory Usage
Container Network Input
Container Network Output

03

经验 & 总结

容器化转变历程

理想中...

01

一次BUILD，到处运行...

02

启动超快，操作非常容易…

03

有大量的成熟容器管理工具可用...

容器化转变历程

实际中...

01

Image越来越大，网络部署成本增加…

Volume，network管理，服务发现…怎么做？

02

03

编排工具学习成本、资源需求、可用性…

容器化转变历程

实际中...

04 Docker registry的服务高可用问题

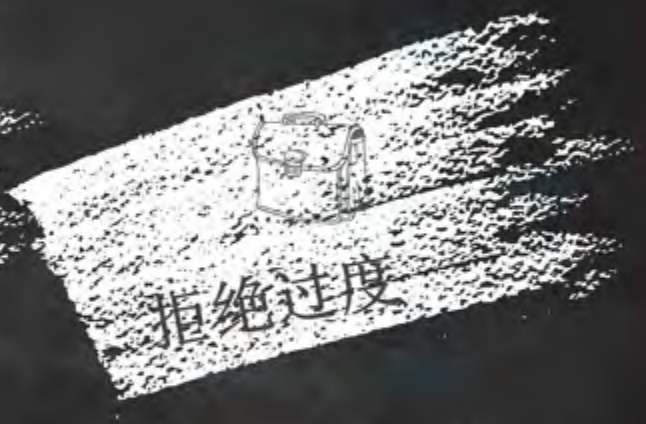
05 Docker动静转化怎么做？

06 容器的资源如何有效限制？

基本原则

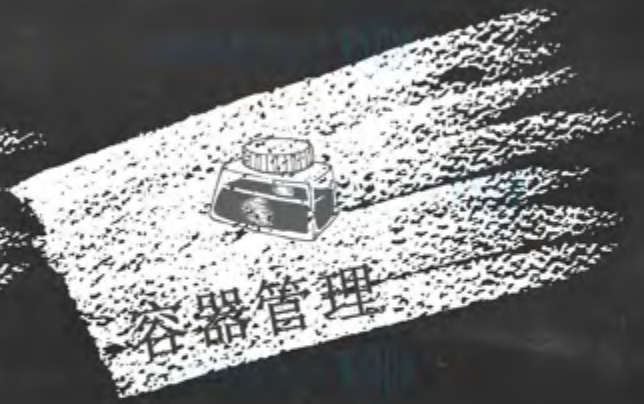
- 应用不宜过于复杂，处理单一问题，尽量拆分业务逻辑。
- 配置外部导入，由容器创建时外部传入。
- 输出尽量统一（日志格式、数据契约）

- 一切皆是脚本，Dev 与 Ops之间以代码交付
- 不要迷信记忆力和文档



- 容器内只做好自己的事，不要涉及外部（比如：负载均衡、日志文件管理）
- 容器外部不要干涉内部的逻辑

基本原则



- 微容器化，满足基本需求即可
- 维持基础镜像的单一化
- 及时清理过期容器和镜像

- 保证容器内环境的一致性（systime、locale...）
- 动静分离，会修改的内容都mount volume
- run之前做好限制和报警，防止抢夺计算资源

04

未来规划

TM+容器化发展规划

01

传统模式 （开发 -> 测试 -> 打包 -> 部署 -> 更新）

02

混合模式 （Docker + 私有云 + 传统模式）

03

微服务化 + 混合云，按需分配

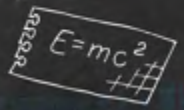
容器管理

Kubernetes

- Google出品、功能强大、文档丰富
- 复杂、部署不易，学习成本高

Rancher

- 轻量、功能完整、易用性高，兼容几乎所有第三方工具
- 比较新，文档较少



Swarm

- 原生编排引擎、简单方便
- 功能略简单，中小型应用足够

未来规划

01

网络建设
更快、更灵活的传输

02

4k / 8k 高清
vp9，h265支持

03

微服务化
动态Scale out，更快响应

04

与AI、VR/AR结合
更加丰富的教学场景。

05

更多微信支持
短视频支持

感谢各位聆听

Q & A