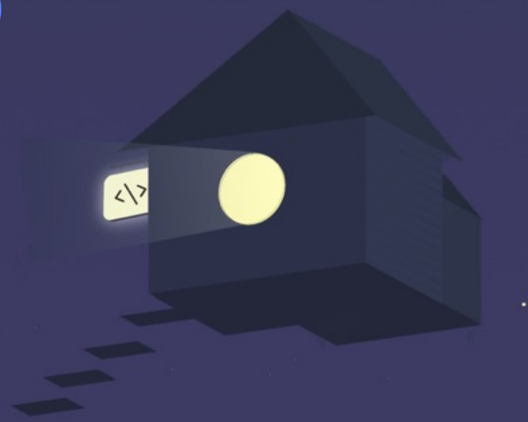
Coding Docker
实践之路

wuke@coding.net
吴柯



关于 Coding

CODING
CLOUD DEVELOPMENT

码市
CODE MART

- 云端开发/网络众包
- 一家B轮技术公司
- 深圳/上海/北京/成都

Git

WebIDE

协作

App

容器是什么

资源隔离

技术实现原理

应用打包的方式

表现形式

发展历程

都有谁在支持

资源隔离

基于进程的资源隔离，进程的资源包括：

- 静态视角

  :代码和数据的集合

  表现为:磁盘上的可执行镜像(Executable Image)

- 动态视角

  :程序运行起来的执行环境的总和

  表现为:指令与数据、寄存器值、堆栈、被打开文件数、I/O 设备状态等信息的动态变化

资源隔离

容器的资源隔离，依赖高版本 linux kernel 的如下几种技术实现：

- linux namespace 实现资源隔离如 PID,IPC,Network
- cgroups (control groups) 限制资源如 CPU,Memory
- capabilities 以及 selinux(MAC) 实现安全访问控制

资源隔离

相关技术在 Linux 的表现

- Linux namespace
- cgroups

```
ipc  mnt  net  pid  user  uts
root@vagrant-ubuntu-trusty-64:/proc/24/ns# ll
total 0
dr-x--x--x 2 root root 0 Sep  8 19:23 ./
dr-xr-xr-x 9 root root 0 Aug 19 13:20 ../
lrwxrwxrwx 1 root root 0 Sep  8 19:23 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 Sep  8 19:23 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 Sep  8 19:23 net -> net:[4026531956]
lrwxrwxrwx 1 root root 0 Sep  8 19:23 pid -> pid:[4026531836]
lrwxrwxrwx 1 root root 0 Sep  8 19:23 user -> user:[4026531837]
lrwxrwxrwx 1 root root 0 Sep  8 19:23 uts -> uts:[4026531838]
root@vagrant-ubuntu-trusty-64:/proc/24/ns#
```

```
root@vagrant-ubuntu-trusty-64:/proc# cat cgroups
#subsys_name    hierarchy       num_cgroups     enabled
cpuset  2       22      1
cpu     3       22      1
cpuacct 4       22      1
memory  5       22      1
devices 6       22      1
freezer 7       22      1
blkio   8       22      1
perf_event      9       22      1
hugetlb 10      22      1
root@vagrant-ubuntu-trusty-64:/proc#
```

```
root@vagrant-ubuntu-trusty-64:/proc/24# cat cgroup
11:name=systemd:/
10:hugetlb:/
9:perf_event:/
8:blkio:/
7:freezer:/
6:devices:/
5:memory:/
4:cpuacct:/
3:cpu:/
2:cpuset:/
root@vagrant-ubuntu-trusty-64:/proc/24#
```

应用打包的方式

- 静态视角：

  镜像:程序、数据以及它们所依赖的所有文件、目录和配置的集合

  表现为: rootfs

  文件系统结构和 library

  目录: /dev /proc /bin /etc /lib /usr /tmp

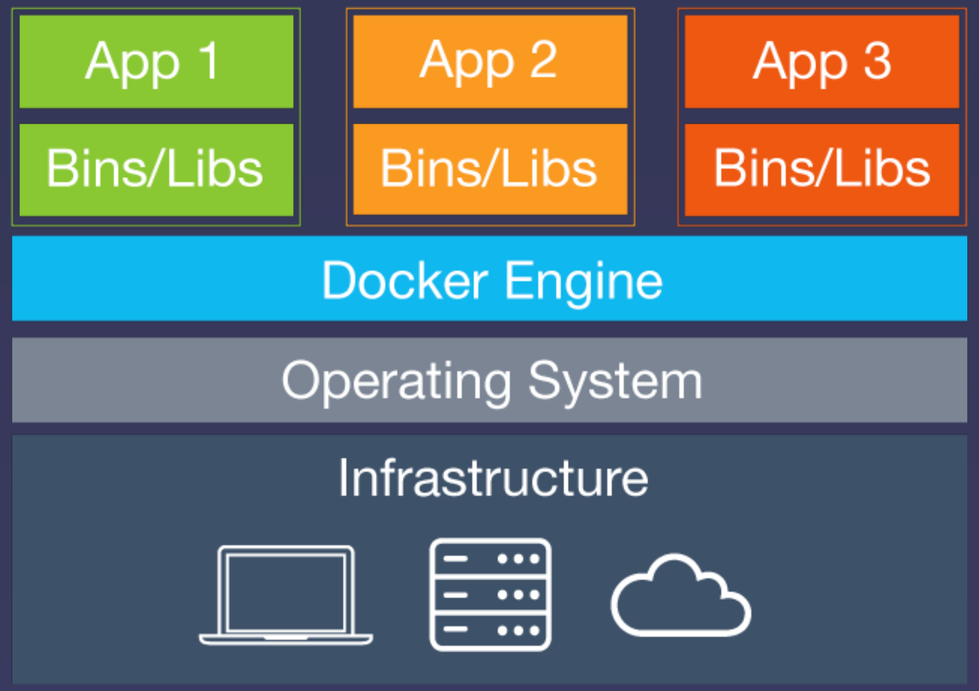
  指令: sh ls cp mv etc…

  配置: rc inittab fstab etc…

  设备: /dev/hd* /dev/tty* /dev/fd0 etc…

- 动态视角

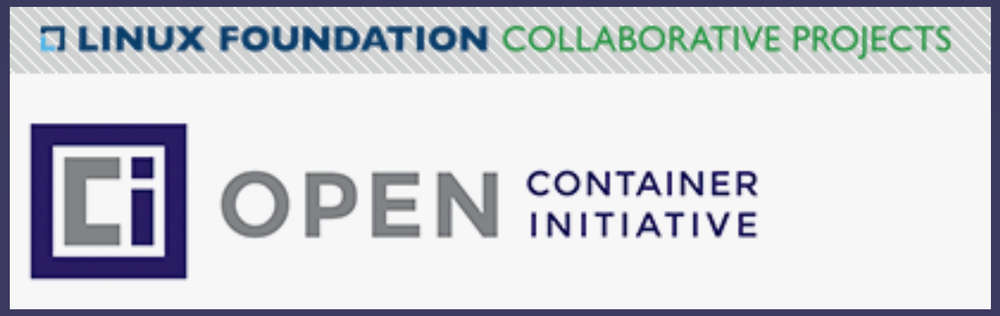
  容器:程序运行起来的执行环境的视图和边界



容器的发展过程

- Google 是最早研究容器技术的公司，并在内部有大规模部署
- libcontainer 是 docker 最初根据 Google 提出的一些关于容器相关的论文等做出得一个实现库，还有 rocket 等

Docker 把容器格式和运行时系统 runC 捐赠给了开放容器联盟(OCI)
https://github.com/opencontainers/runc.



为什么用 Docker

轻量的资源隔离方式

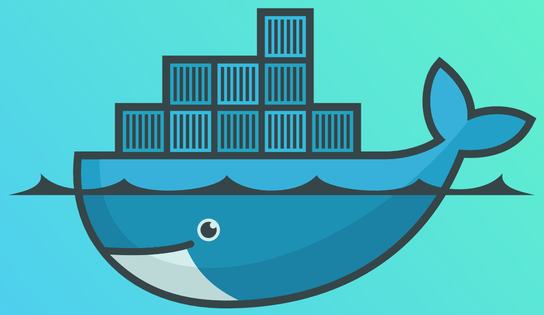
便捷的应用部署

健康的社区生态

Coding Docker 之路

使用 Docker 驱动所有服务

自建 Docker Registry



容器化

Build in Docker, Run in Docker

自建工具控制 Docker Daemon

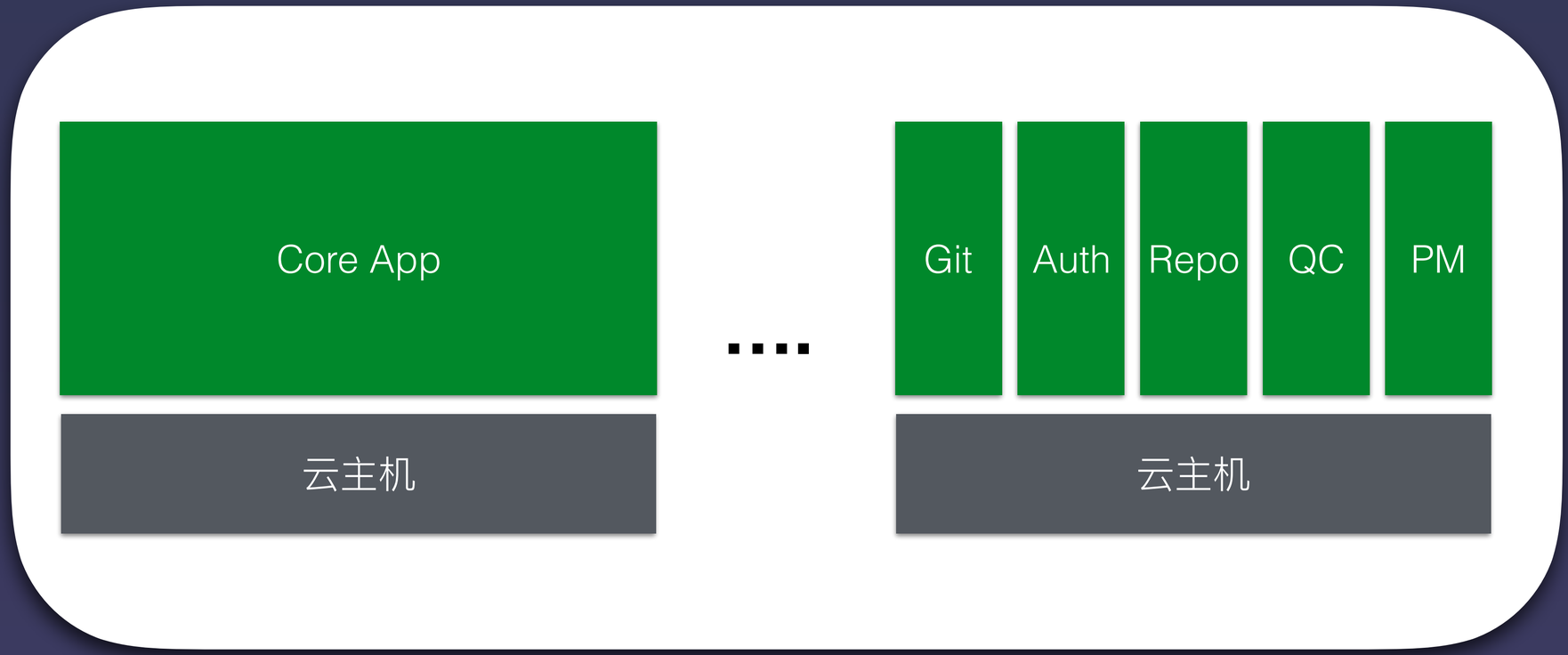
业务拆分架构调整：阶段一

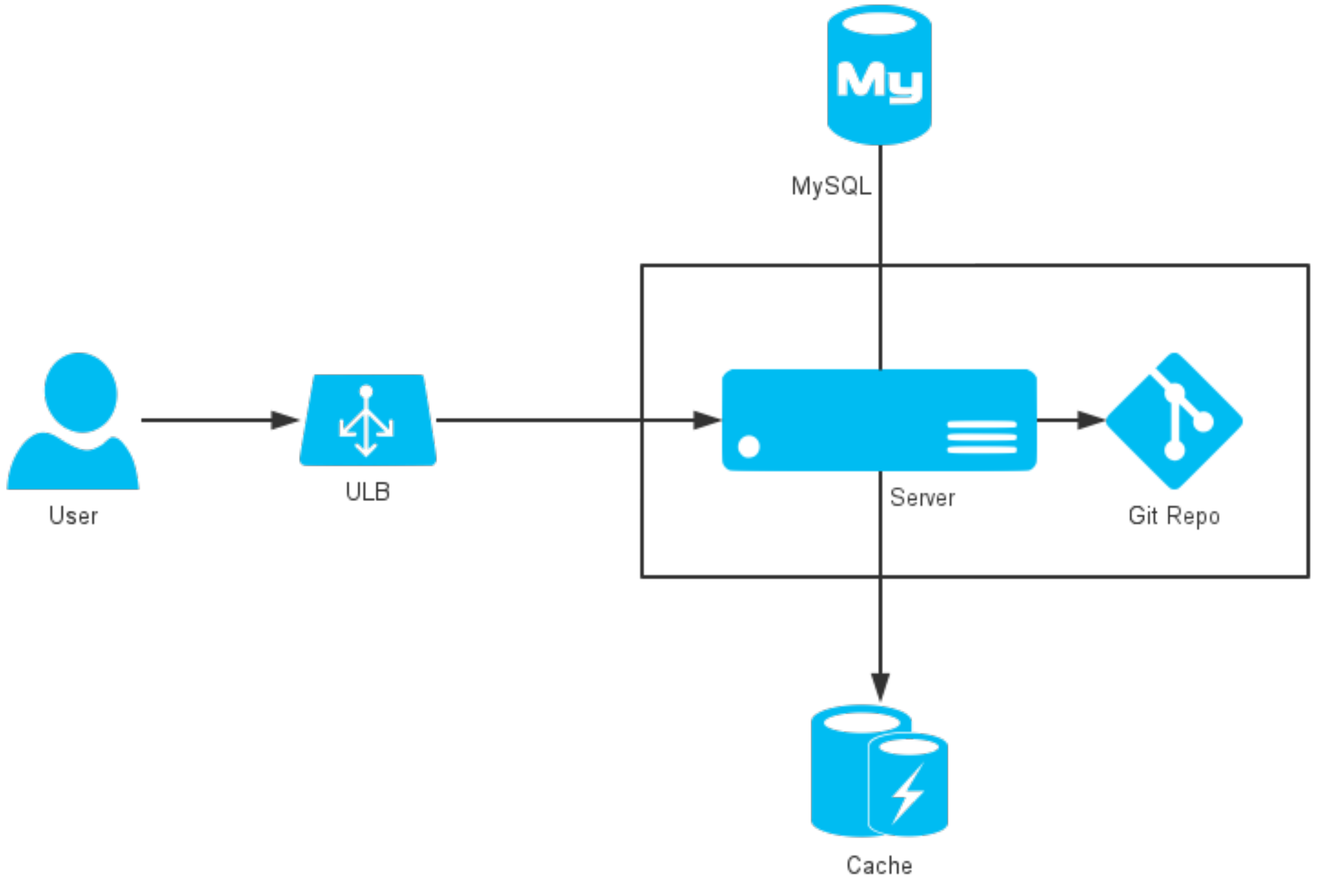
分拆主站业务和相关服务

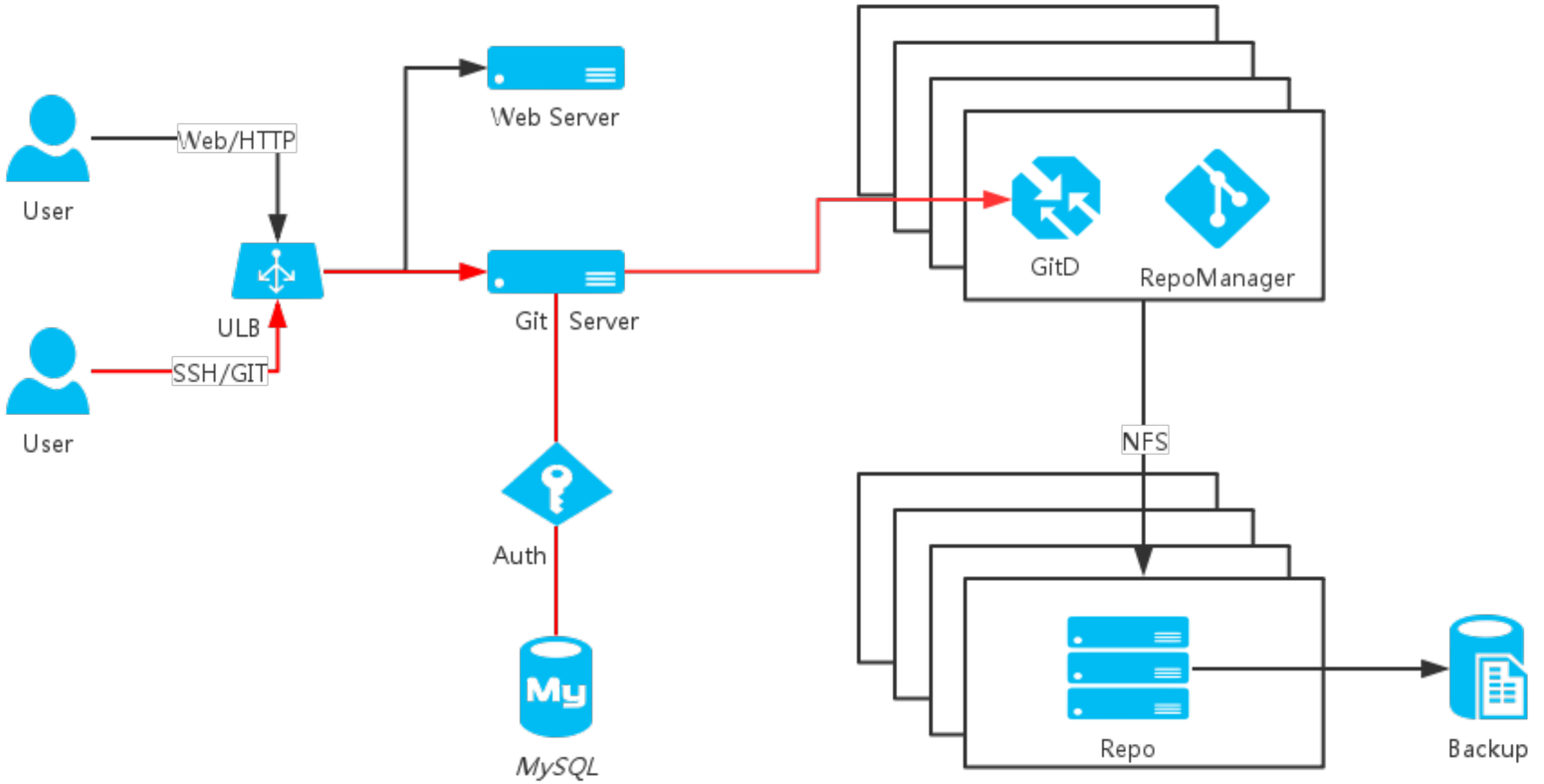
单实例到多实例

单一存储到多存储

业务分拆




MySQL
User
ULB
Server
Git Repo
Cache


User
Web/HTTP
User
SSH/GIT
ULB
Web Server
Git Server
Auth
My
MySQL
GitD
RepoManager
NFS
Repo
Backup

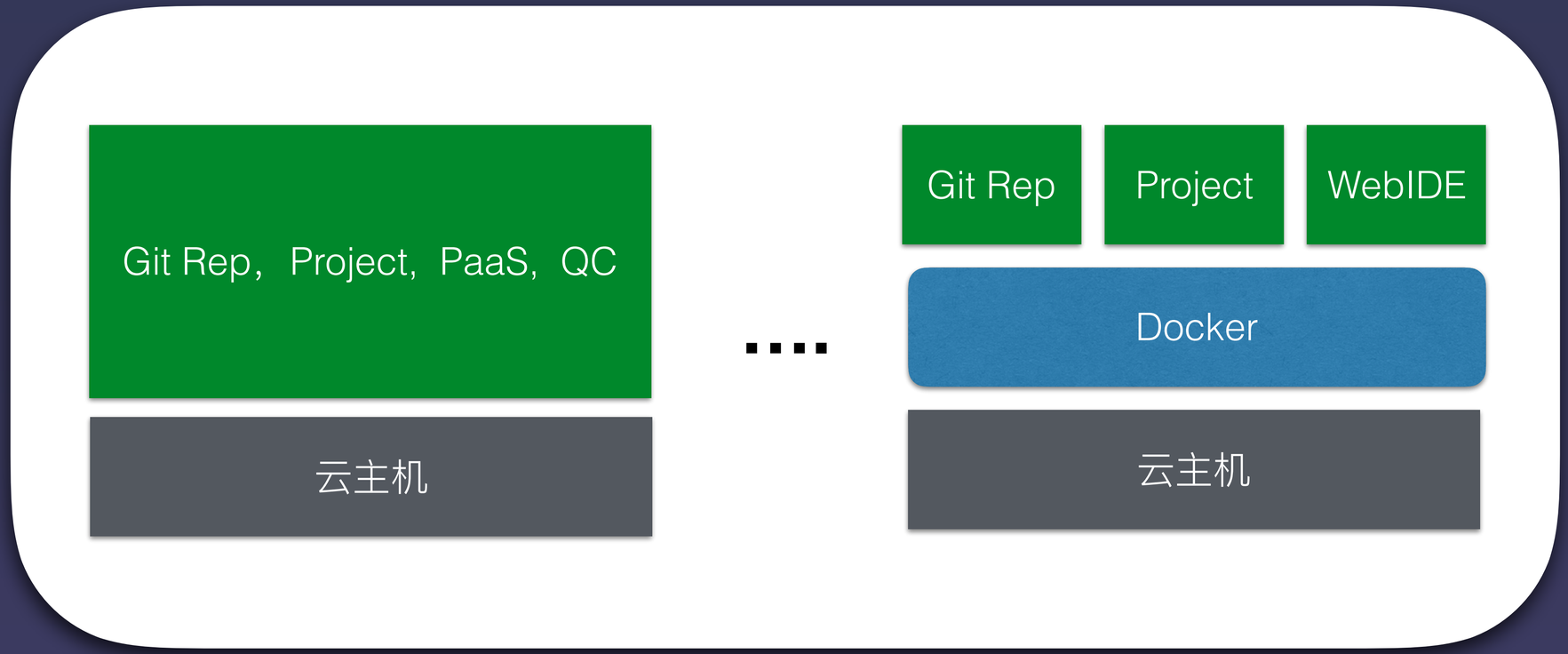
应用容器化：阶段二

应用打包成 image

维护个位数 base image

构建自动化

应用容器化


Git Rep，Project，PaaS，QC
云主机
....
Git Rep
Project
WebIDE
Docker
云主机

容器配置

单进程, 微服务，没必要那么多限制

trust-cooperative 环境，我们不是IAAS

Host 网络模式

不趟SDN, proxy这样的浑水/雷区，性能也不受影响

host上数据持久化

未来可能可以考虑 data container

直接 mount Host locale, timezone, passwd 等配置.

统一开发到构建

- Coding-local 统一本地环境，即使用一个 Vagrant VM，配合 Coding-local 这样一套构建脚本和 Dockerfile，运行起整个 Coding 系统
- 生成产机器通过拉取 Git 仓库代码编译并 build image， 并 push 到 docker-register
- docker-registry 分发到目标服务器

踩的几个 docker 大坑

- docker container 在 stdout 有大量数据传输会导致内存泄露，直至 docker daemon 被 OOM
- docker daemon 在频繁创建 container 后，会在文件系统中遗留很多垃圾文件不清理，导致磁盘 inode 被耗尽
- docker 是推荐单个容器里面只有一个进程的，但是容器技术本质上是没这个限制，如果在 docker 容器里面 fork 很多进程，会在系统中出现很多僵尸进程，最终导致 docker daemon 出现问题。

Docker 也有一些问题

管理编排难题

简单粗暴的打包，冗余呢

故障排查和分析麻烦

Docker File 的问题

RUN 的问题

apt-get update –y && apt-get upgrade –y

每句加一个layer, 轻松来个十几G的镜像

没完没了的等，继续等，使劲等

```
# Base
FROM java:8-jdk

COPY ./.src/target/RepoManager-1.0.jar /app/

RUN useradd -m -u 1000 coding
USER coding

WORKDIR /app
CMD [ "java", "-server", \
      "-Xms2048m", "-Xmx2048m","-XX:+UseFastAccessorMethods", \
      "-jar", "./CodingBlog-1.0.jar" \
]
```

容器编排管理：阶段三

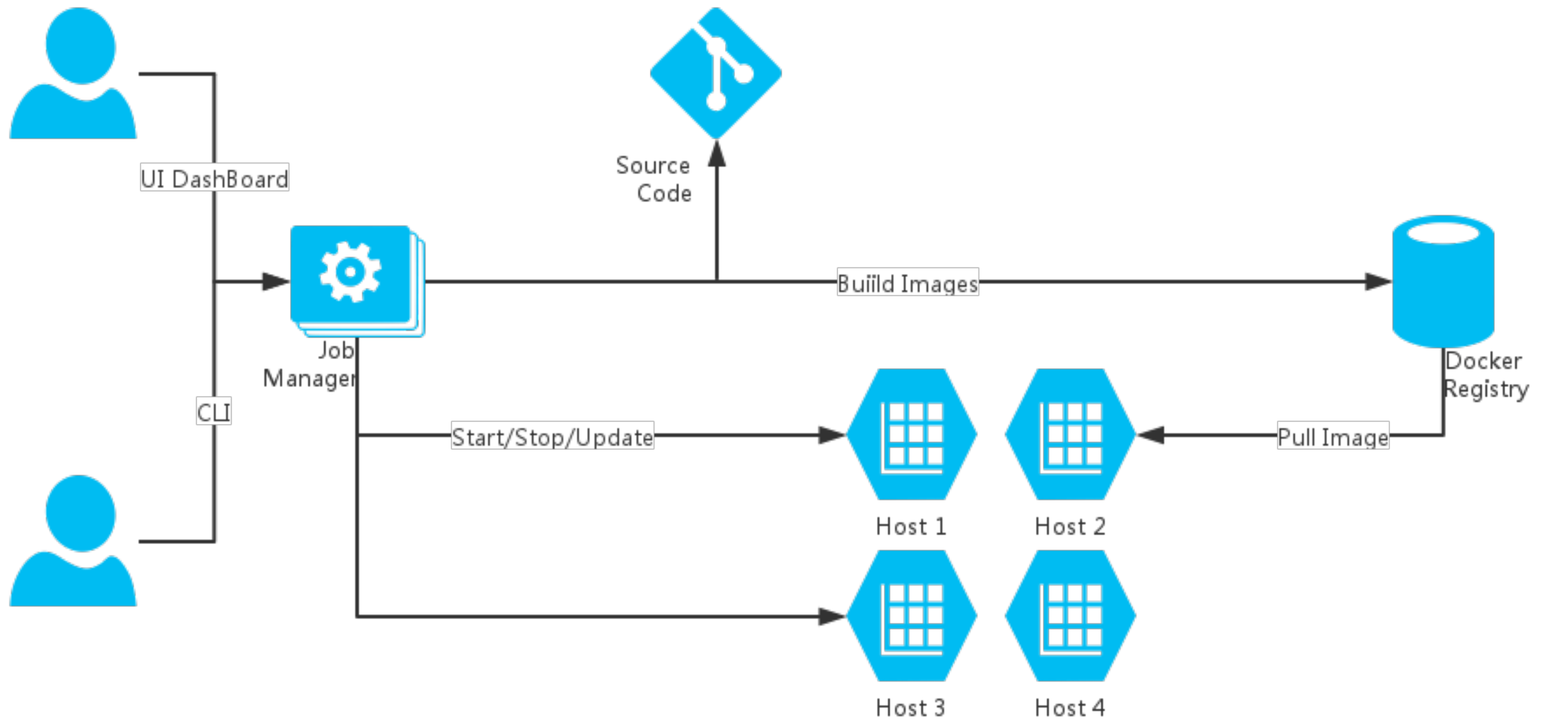
管理容器运行的状态

批量化可视化的操作

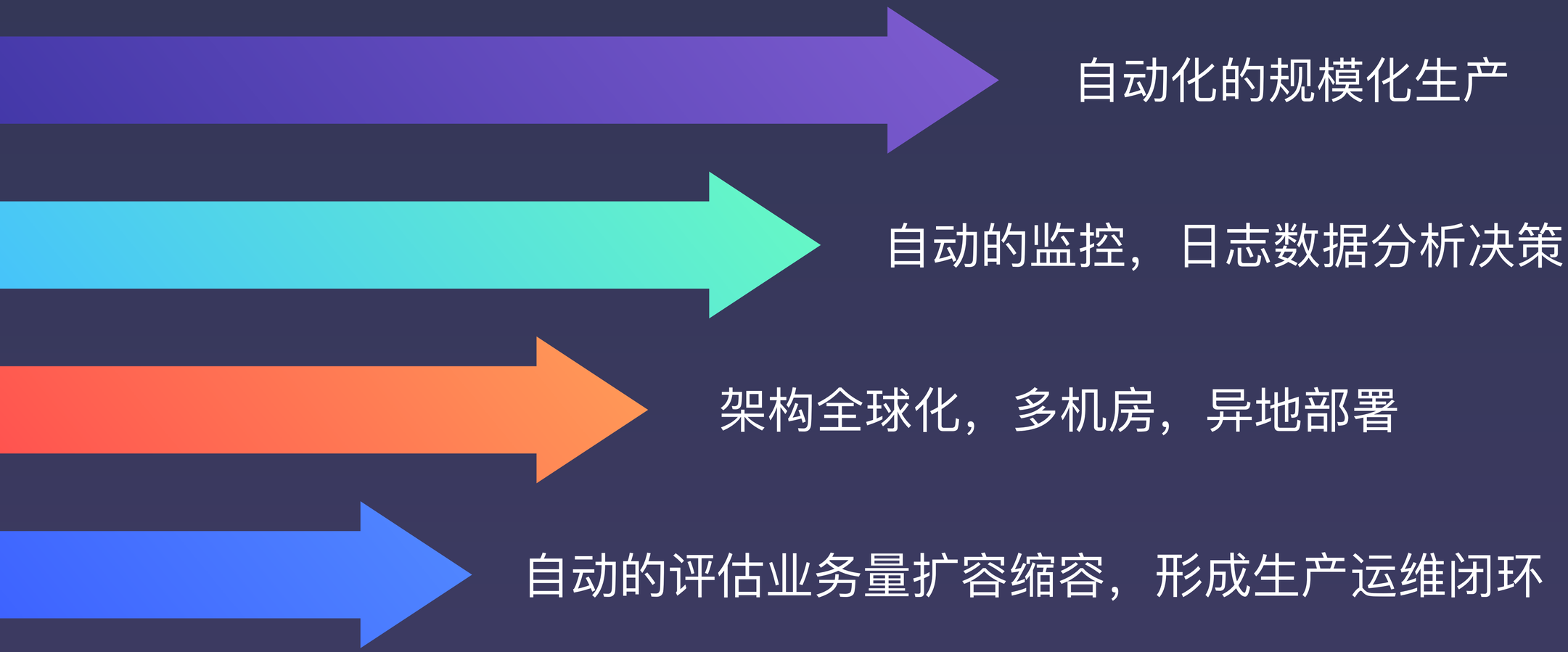
自动化的监控机制

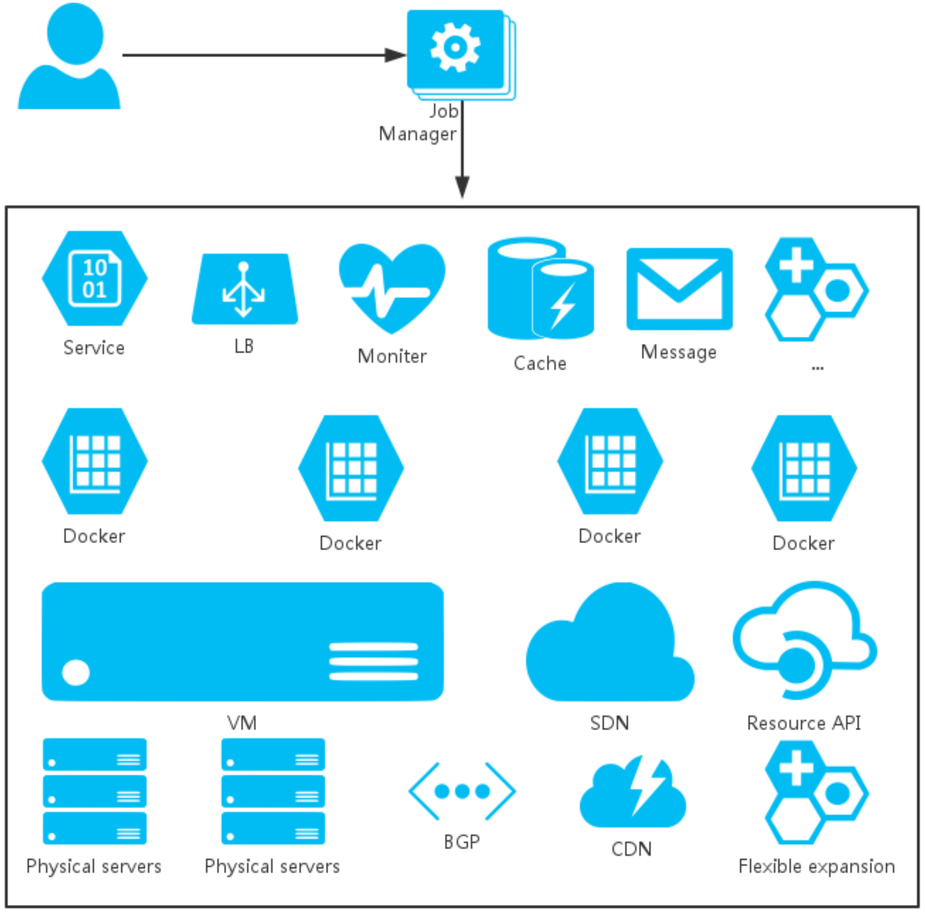
容器集群编排技术

- Apache Mesos 其本身适合分布式计算，大数据场景，擅长的是资源调度，本身跟容器技术没有关系，容器只是可以作为其计算单元的一个存在
- Swarm 是 docker 官方推出的一个 docker 集群工具，可以将多台 host 模拟成一个单个的 docker daemon。基本上做到了90% 的兼容。
- kubernetes 是 Google 推出一套容器编排技术，是 Google 内部庞大容器框架 Borg 的一个简单抽象和模拟


UI DashBoard
CLI
Job
Manager
Source
Code
Buiild Images
Docker
Registry
Start/Stop/Update
Pull Image
Host 1
Host 2
Host 3
Host 4

下一阶段


自动化的规模化生产
自动的监控，日志数据分析决策
架构全球化，多机房，异地部署
自动的评估业务量扩容缩容，形成生产运维闭环


Job
Manager
Service
LB
Moniter
Cache
Message
...
Docker
Docker
Docker
Docker
VM
SDN
Resource API
Physical servers
Physical servers
BGP
CDN
Flexible expansion

Thanks