

蘑菇街基于Docker的 私有云实践

张振华

guojia@mogujie.com

关于我

- 花名：郭嘉 — 张振华
- 05年浙大硕士毕业
- 虚拟化团队负责人
- 热爱新技术, 开源
- guojia@mogujie.com



关于蘑菇街 ABOUT US

中国最大的女性时尚社交电商平台。

成立于2011年，总部位于浙江杭州，目前拥有1.3亿注册用户，日活跃用户超过800万，2014年全年实际交易额超过36亿元，团队总人数超过800人。无论在用户规模上，还是交易额上，都已经成长为中国最大的女性时尚社交电商平台。

自公司成立以来，蘑菇街一直坚持社交与电商相结合的发展方向，致力于开创全新的社交电商商业模式，面向新一代年轻时尚人群提供优质的社交和购物体验。蘑菇街的核心用户群体是18-26岁之间年轻时尚的都市女性，他们崇尚自由独立，个性解放，拥有独到的审美品位与时尚主张，以及巨大的消费潜力。



@杭州 @北京!

Docker的优势

- 轻量，秒级的快速启动速度
- 简单，易用，活跃的社区
- 标准统一的打包/部署/运行方案
- 镜像支持增量分发，易于部署
- 易于构建，良好的REST API，也很适合自动化测试和持续集成
- 性能，尤其是内存和IO的开销

简单易用

```
docker run -d
  --net=none

  --name=$name

  -h $name

  -v /var -v $tmpdir/resolv.conf:/etc/resolv.conf -v $tmpdir/hosts:/etc/
hosts

  --cpuset="$cpuset"

  -m ${mem}m

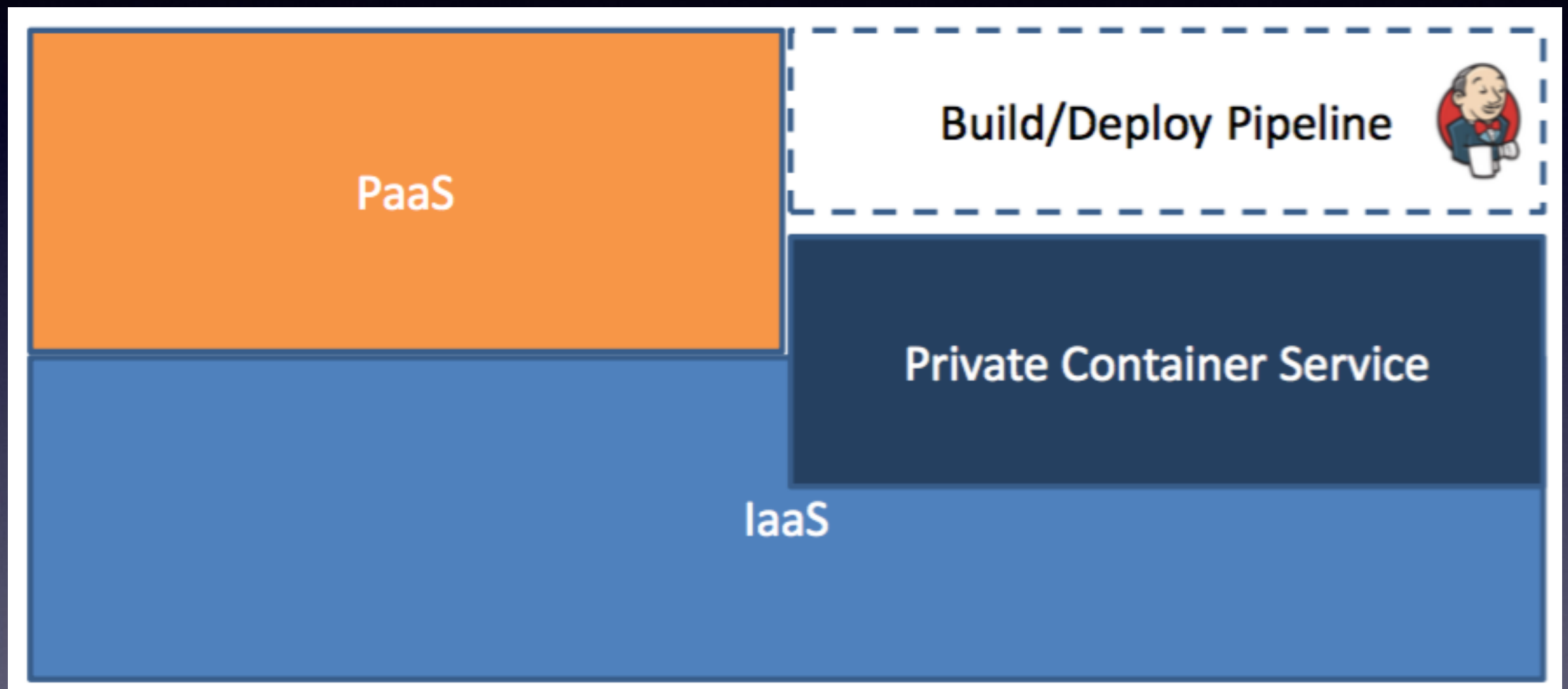
--privileged=true

$image
```

劣势

- 资源调度 / 集群管理还在春秋战国时期
- Docker daemon自身的发展也很快, 平滑升级困难
- 隔离性对内核的依赖很强 – 仅用cgroup / namespace够吗?
- 网络 / 存储支持完善吗?
- 不支持热迁移 – CRIU?
- 坑! 坑! 各种坑!

Docker模糊了边界



现实与理想

“Pets vs Cattle” (Scale Up vs Scale Out)



- 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



- 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

The above adapted from Tim Bell, CERN

<http://www.slideshare.net/noggin143/20121017-openstack-cern-accelerating-science>

Docker@蘑菇街

- 2014年圣诞节期间上线，OpenStack IceHouse + Docker 1.3.2。经历过5次 + 大促，包括双11 / 双12，线上运行稳定。
- Machine Container 或 “胖容器”，用supervisord管理容器中的多进程。
- 每个机房一个OpenStack集群，一个Docker Registry。每个集群可以**同时管理**KVM，Docker。
- 支持OpenvSwitch VLAN和Linux Bridge两种网络模式，不用Docker原生的NAT网络模式，other_args="—bridge=none"。
- 自研了基于OpenStack的PaaS平台，虚拟化交付系统，虚拟化管控控制台。

为什么都把Docker当成虚拟机？

	KVM	Machine Container	App Container
对业务的侵入性	无	低，内核参数配置会引起全局变化	高，需要写Dockerfile，业务需要适配容器的用法，环境配置，启动方式
性能	中，比较重。 IO性能损失相对较大	好	好
运维成本	低	中，top / sar等常用工具数据不正确，需要改造。	高，监控 / 日志需要对接，运维人员需要接受新事物



只有Docker是不够的



集群管理

虚拟机管理

admin

项目

Compute

概况

实例

镜像

访问 & 安全

管理员

实例

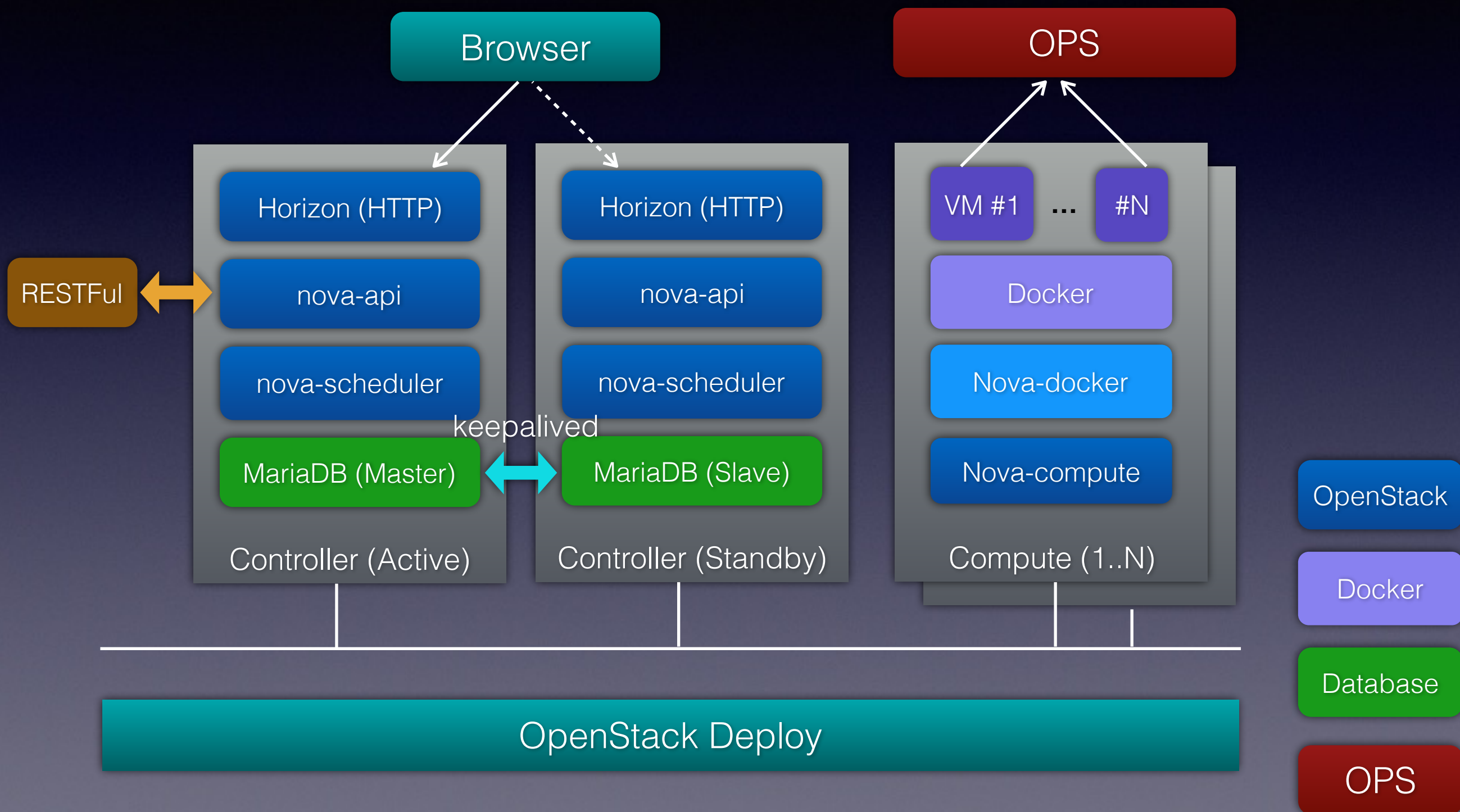
筛选

筛选

+

	云主机名称	镜像名称	IP 地址	配置	值对	状态	可用域	任务	电源状态
<input type="checkbox"/>	temp-test	web-service-1.7-logagent	10.15.3.96	6core 20GB 内存 6 虚拟内核 10.0GB 盘	-	Active	2AB10	None	Running
<input type="checkbox"/>	gmond-test	web-service-1.6-tesla	10.15.3.95	6core 20GB 内存 6 虚拟内核 10.0GB 盘	-	Active	2AB9	None	Running
<input type="checkbox"/>	web3053	web-service-1.6-tesla	10.15.3.53	8core 20GB 内存 8 虚拟内核 10.0GB 盘	-	Active	2AB9	None	Running

逻辑架构图



我们在容器启动前**做了很多事情**

- 根据ip段的规则来生成容器的hostname
- 动态添加ip:hostname映射到/etc/hosts
- 启动docker实例时指定CPU set / weight
- 自定义需要mount的folder/file volume
- 扩展OpenStack API，通过cgroup限制磁盘IO和网络IO
- 和Linux bridge / Openvswitch对接网络。
- 实现快照方式的冷迁移。

<https://github.com/openstack/nova-docker/>

监控

- 和已有监控系统的深度集成。
- 实时监控和阈值报警：节点存活性/语义监控，关键进程，内核日志，实时pid数量，网络连接跟踪数，容器oom报警。
- 阈值报警：短信报警，IM报警等多种形式。
- 健康检查：部署环境 / 配置的一致性检查。
- container-tools：
 - 参考内核算法实现容器内load值的计算。
 - 替换了uptime, top, free, df, tsar，类似docker stats。

```
#define FSHIFT      11          /* nr of bits of precision */
#define FIXED_1     (1<<FSHIFT) /* 1.0 as fixed-point */
#define LOAD_FREQ   (5*HZ+1)   /* 5 sec intervals */
#define EXP_1       1884        /* 1/exp(5sec/1min) as fixed-point */
#define EXP_5       2014        /* 1/exp(5sec/5min) */
#define EXP_15      2037        /* 1/exp(5sec/15min) */

#define CALC_LOAD(load,exp,n) \
    load *= exp; \
    load += n*(FIXED_1-exp); \
    load >>= FSHIFT;

#define LOAD_INT(x) ((x) >> FSHIFT)
#define LOAD_FRAC(x) LOAD_INT(((x) & (FIXED_1-1)) * 100)

unsigned long avenrun[3];

static unsigned long
calc_load(unsigned long load, unsigned long exp, unsigned long active)
{
    load *= exp;
    load += active * (FIXED_1 - exp);
    return load >> FSHIFT;
}

/*
 * * calc_load - update the avenrun load estimates 10 ticks after the
 * * CPUs have updated calc_load_tasks.
 * */
void calc_global_load(long active)
{
    active = active > 0 ? active * FIXED_1 : 0;
    avenrun[0] = calc_load(avenrun[0], EXP_1, active);
    avenrun[1] = calc_load(avenrun[1], EXP_5, active);
    avenrun[2] = calc_load(avenrun[2], EXP_15, active);
}
```


Container-tools

```
top - 14:55:08 up 7 days, 19:17, 1 user, load average: 0.00, 0.00, 0.00
Tasks: 11 total, 1 running, 10 sleeping, 0 stopped, 0 zombie
Cpu4 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu5 : 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4194304k total, 468536k used, 3725768k free, 0k buffers
Swap: 0k total, 0k used, 0k free, 453440k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	97.7m	13m	3840	S	0.0	0.3	2:30.46	/usr/bin/python2.6 /usr/bin/supervisord
19	root	20	0	4056	420	332	S	0.0	0.0	1:43.87	/usr/bin/container_load_calc
80	root	20	0	20460	1276	676	S	0.0	0.0	0:02.11	crond
116	root	20	0	66608	1284	556	S	0.0	0.0	0:00.00	/usr/sbin/sshd
161	root	20	0	249m	1976	1268	S	0.0	0.0	0:00.47	/sbin/rsyslogd -i /var/run/syslogd.pid -c
357	nsd	20	0	433m	2096	1176	S	0.0	0.0	0:02.00	/usr/sbin/nsd
376	ntp	20	0	32812	2104	1480	S	0.0	0.1	0:00.39	ntpd -u ntp:ntp -p /var/run/ntpd.pid -g

```
$ df
Filesystem 1K-blocks Used Available Use% Mounted on
rootfs 103081248 2127188 95711180 3% /
/dev/mapper/docker-8:1-15991635-a3e1ce9c8719b08e6e4df9845abd796e71e8f6f781dc0befe3eed88588f532d5 103081248 2127188 95711180 3% /
tmpfs 32992060 0 32992060 0% /dev
shm 32992060 56 32992004 1% /dev/shm
```

```
$ df -h
Filesystem Size Used Avail Use% Mounted on
rootfs 99G 2.1G 92G 3% /
tmpfs 32G 0 32G 0% /dev
tmpfs 32G 56K 32G 1% /dev/shm
```

```
$ tsar --cpu --mem -n1 -l
```

Time	cpu						mem					
	user	sys	wait	hirq	sirq	util	free	used	cach	total	util	
19/03/16-07:31:34	0.37	0.50	0.00	0.00	0.00	0.87	16.1G	1.6G	2.3G	20.0G	8.07	
19/03/16-07:31:39	0.60	0.77	0.00	0.00	0.00	1.37	16.1G	1.6G	2.3G	20.0G	8.07	

实时监控和报警



容灾

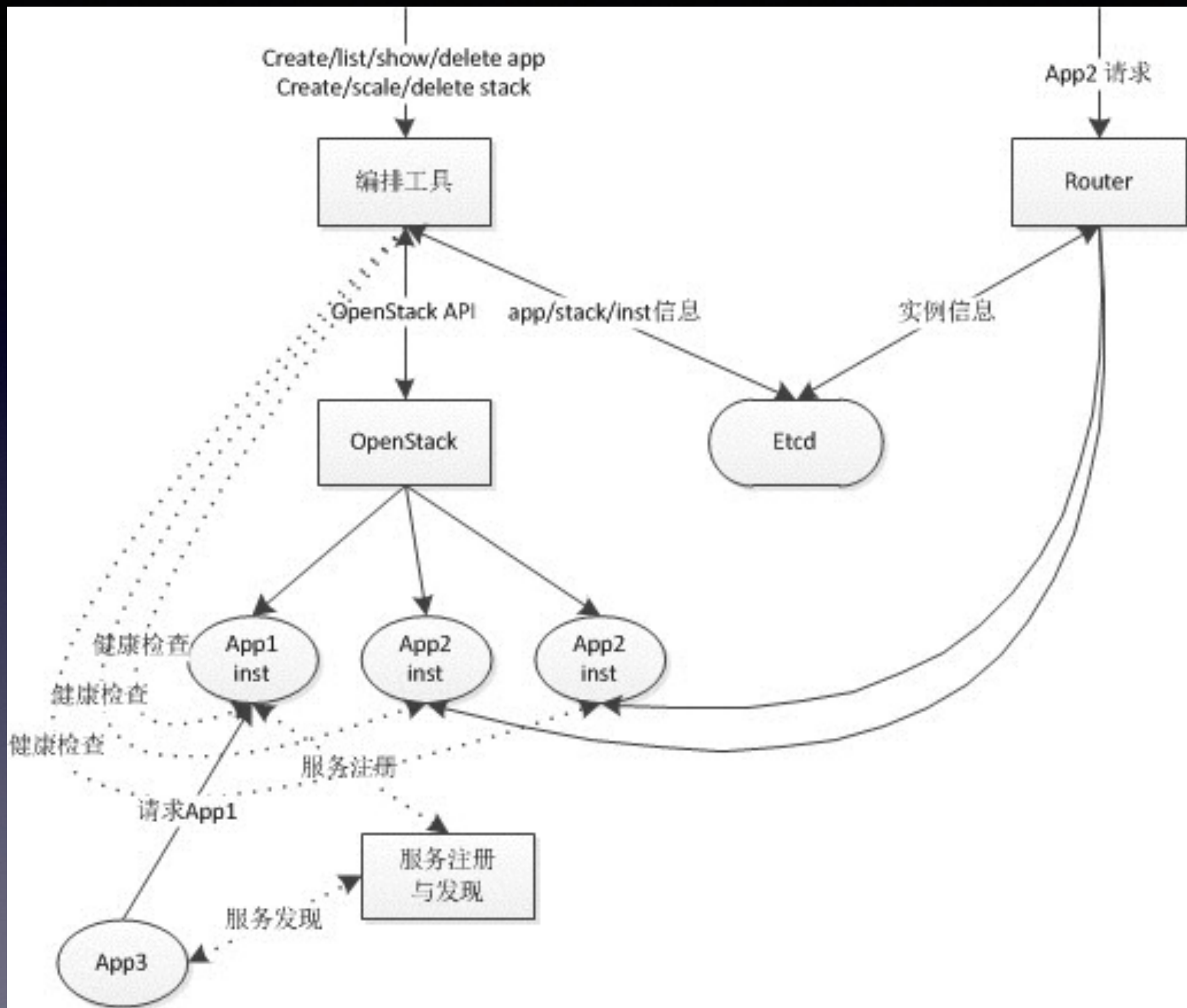
- 离线恢复docker容器中的数据的能力。
- Docker实例跨物理机的冷迁移: `docker commit`, `docker push`。
- 动态的CPU内存扩容: `cgroup`。
- 网络IO / 磁盘IO的限速: `cgroup/tc`。

PaaS




PaaS

- 目标：
 - 快速的系统构建
 - 业务的平滑部署升级
 - 自动的运维管理
- 自研的轻量PaaS，实现编排能力。
- 支持基于容器的持续集成：Jenkins + Docker，从编译到构建全自动化
- 概念：
 - App：一个应用包含一个或多个Stack
 - Stack：相同的Docker实例，一个Stack中的不同实例尽量部署在不同的物理机上















Docker Registry

 Registries Images

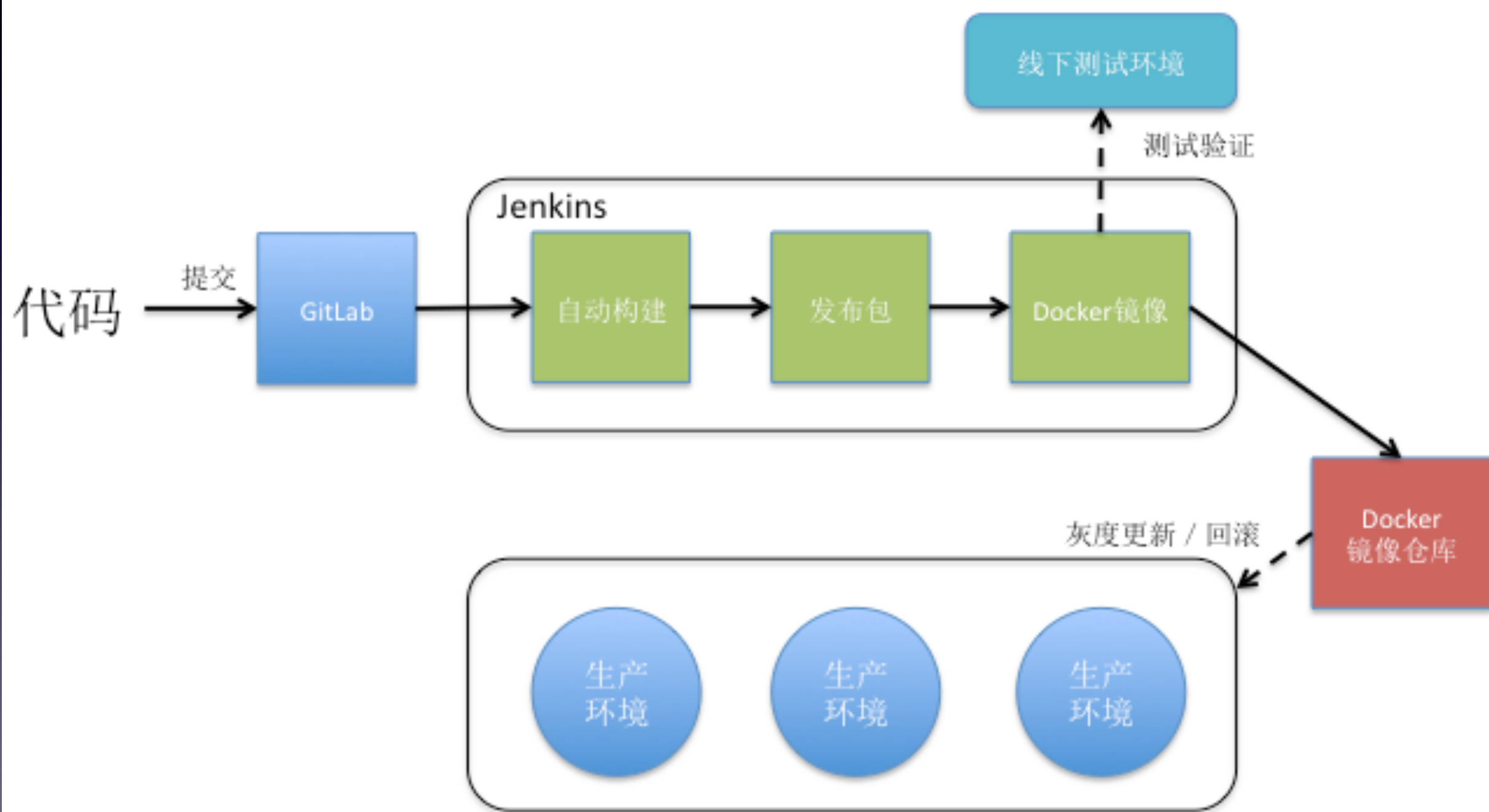
Search Images...

Registry registry.service.mogujie.org

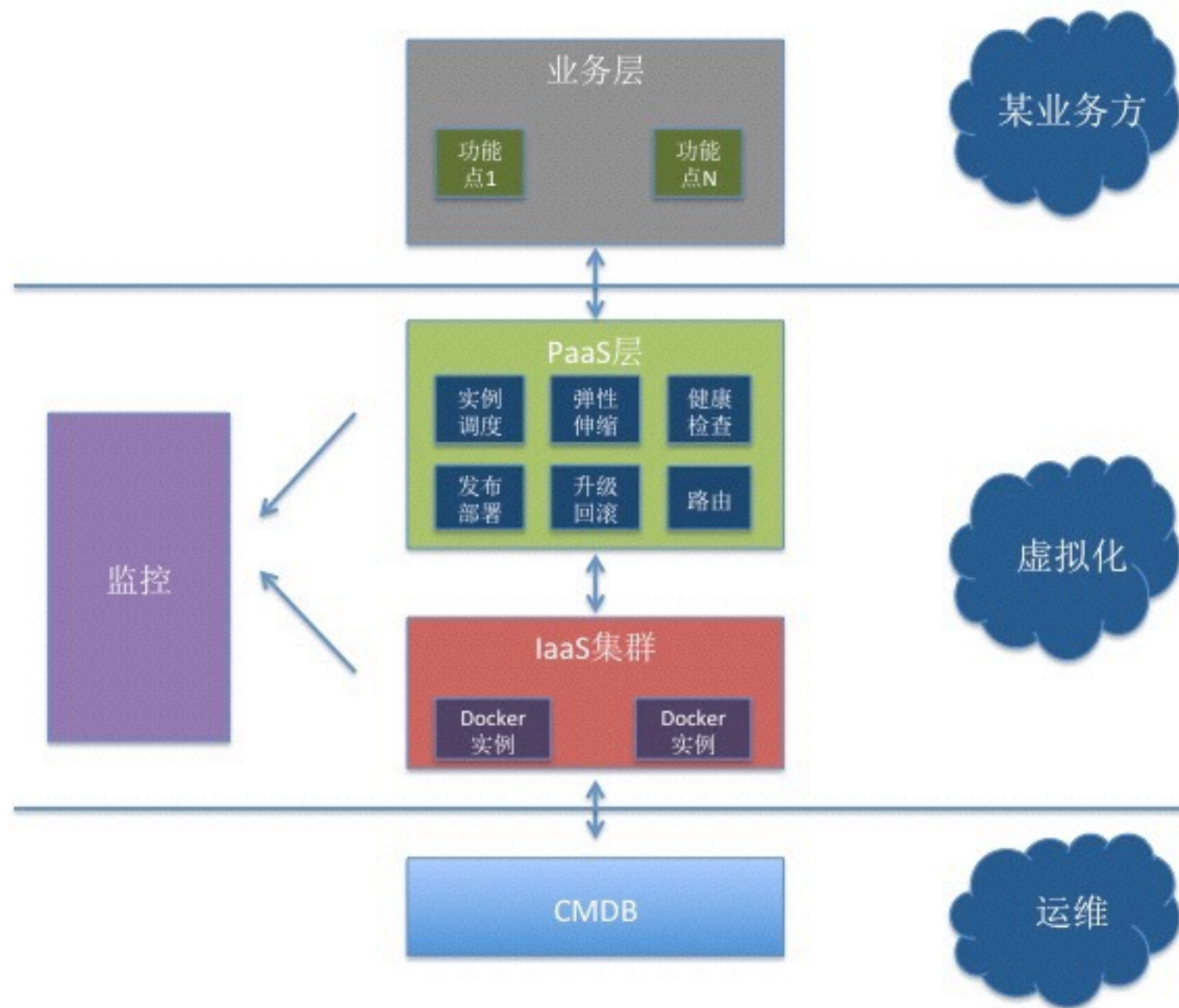
Tag

registry.service.mogujie.org/library/centos6-mgj-themis:0.1	  
registry.service.mogujie.org/library/web-service-2.0:latest	  
registry.service.mogujie.org/eless/tesla:20151124	  
registry.service.mogujie.org/eless/tesla:latest	  

基于Docker的持续集成



基于PaaS的业务平台



遇到的坑

- 物理机上无法执行命令操作。“bash: fork: Cannot allocate memory”
- 容器内如果创建大量的进程，并且不回收。是会导致系统内核的pid_max达到上限。
- 内核中的pid_max(/proc/sys/kernel/pid_max)是全局共享的。
- Process Number Controller:
 - 仅最新的4.3-rc1支持， pid-max per containers。
 - <https://www.kernel.org/doc/Documentation/cgroups/pids.txt>

遇到的坑

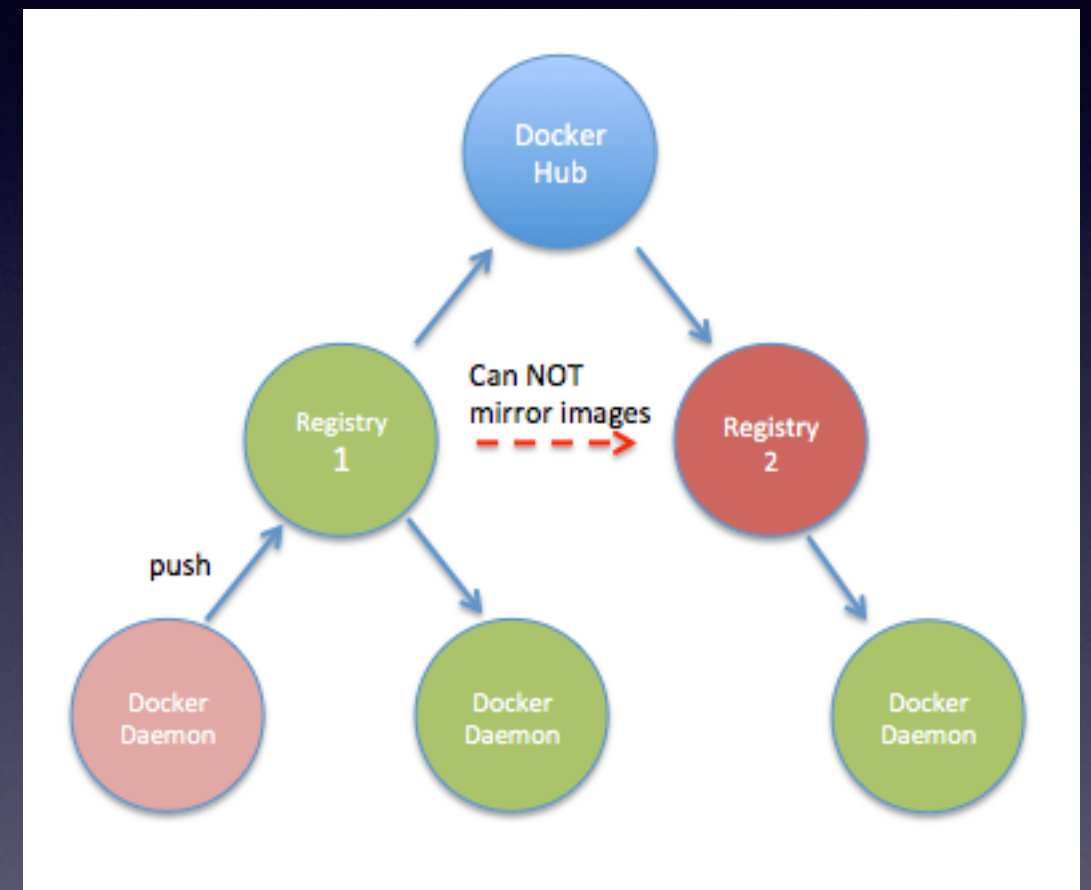
- devicemapper中thin-provisioning的discard功能引起的kernel crash: http://mogu.io/docker_crash-79
- devicemapper overcommit导致文件系统read-only的问题
- nf_conntrack连接跟踪表满了会导致容器无法ping通: 扩大连接跟踪表的最大数目, `/proc/sys/net/ipv4/netfilter/ip_conntrack_max`
- Docker内创建网桥导致内核crash: ver 431 -> 504

遇到的坑

- 容器内的内存值计算不准确: `/cgroup/memory/docker/<id>/memory.usage_in_bytes`
- 内核日志乱序问题: 修改容器里面的syslog的配置文件, 让容器中的syslog不取内核日志
- 业务方重启supervisord导致容器退出: `supervisorctl reload`
- JDK读取的是物理机的内核数目:
`Runtime.getRuntime().availableProcessors()`

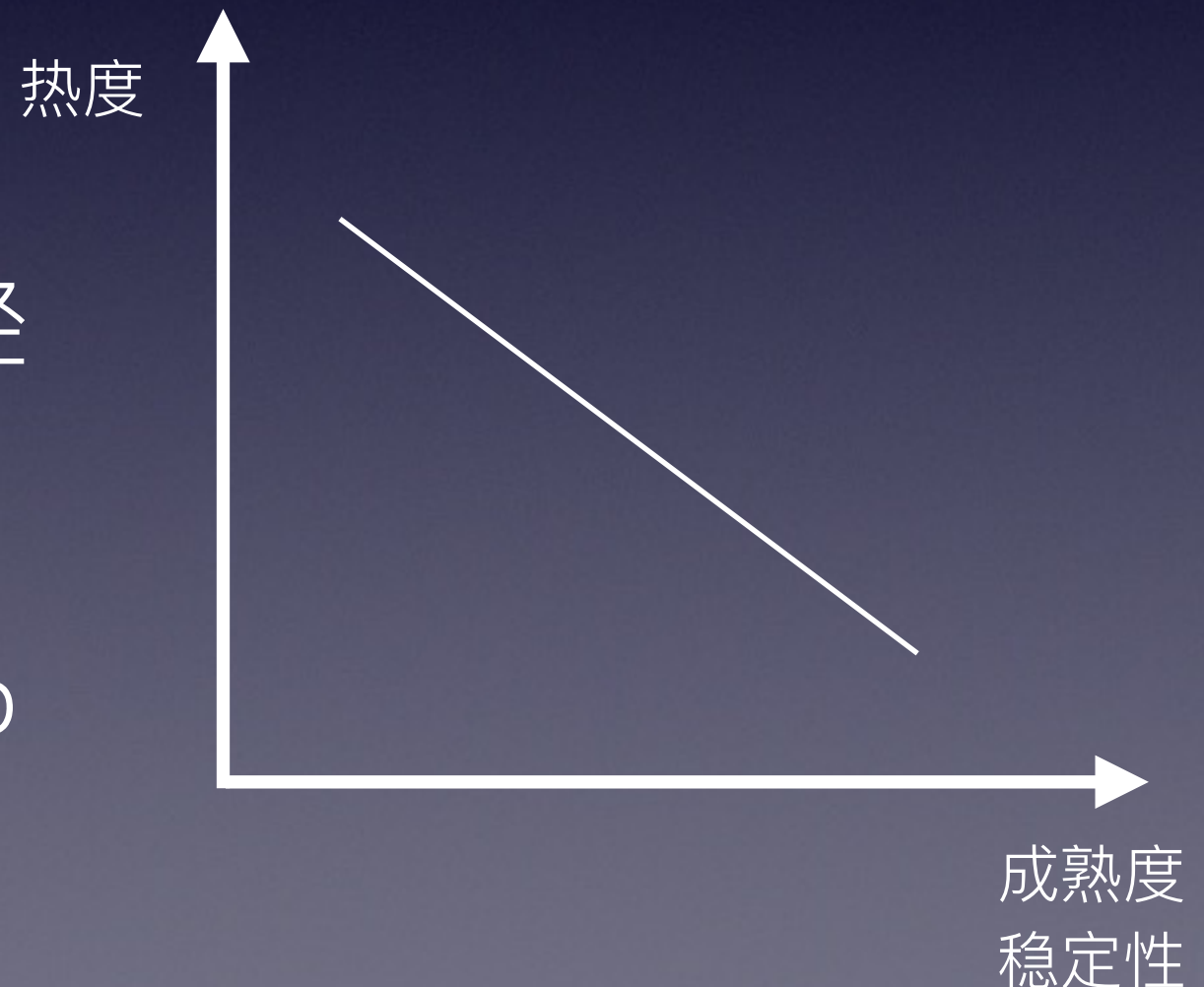
Docker目前的局限

- 系统 / 内核层面的隔离性
- 缺乏成熟的集群管理 (K8S/ Swarm/Mesos)
- 业务无感知的升级, Docker daemon live upgrade
- 多机房docker registry同步的问题



体会和思考

- 相比KVM，容器还有很多不完善的地方。
- 内核的依赖。
- 容器下的运维手段和运维经验的冲击。
- Machine Container -> App Container。



未来的畅想

- PaaS + App Container + CI/CD
- Kubernetes/Swarm/Mesos
- 更高效更便捷的运维
- 统一的部署方式，弹性的资源交付
- 公有云平台

