

蘑菇街基于Docker的 私有云实践

@郭嘉

guojia@mogujie.com

关于我

- 花名：郭嘉 — 张振华
- 05年浙大毕业
- 14年加入蘑菇街
- 虚拟化团队负责人
- 热爱新技术, 开源。。。。

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!

天生爱占有
一生是杀手
喜欢就要独占，
找好货独揽

我们为什么想做私有云

- 越来越多的机器，集群管理，基础平台的建设
- 提高资源的利用率
- 服务化，平台化，可视化
- 提升发布和部署的效率
- 实现业务的弹性，水平扩展

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
- 隔离性－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@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，业务启动的方式不同了
性能	中，太重。 IO性能损失较大	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，运维需要重新熟悉



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

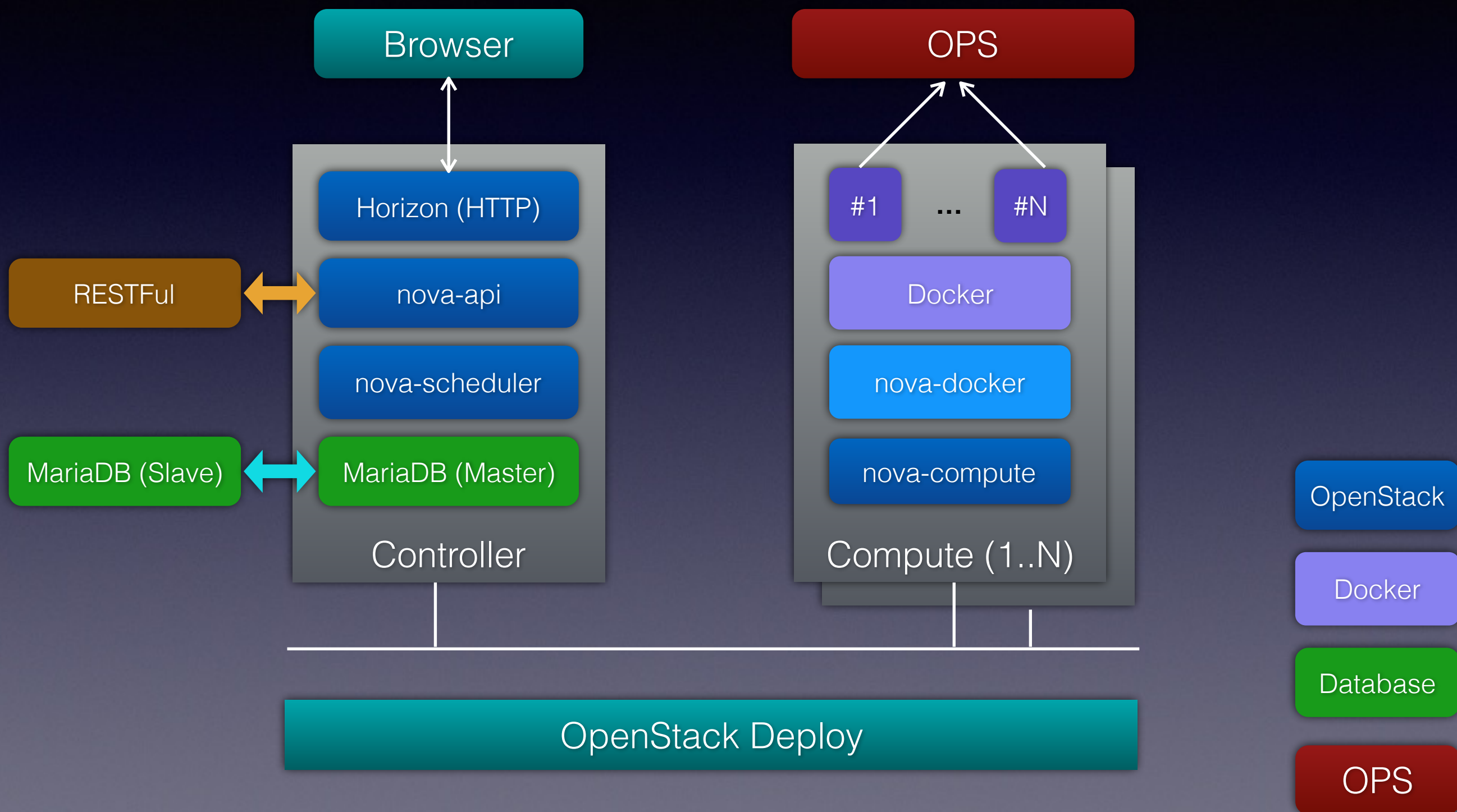
筛选

筛选

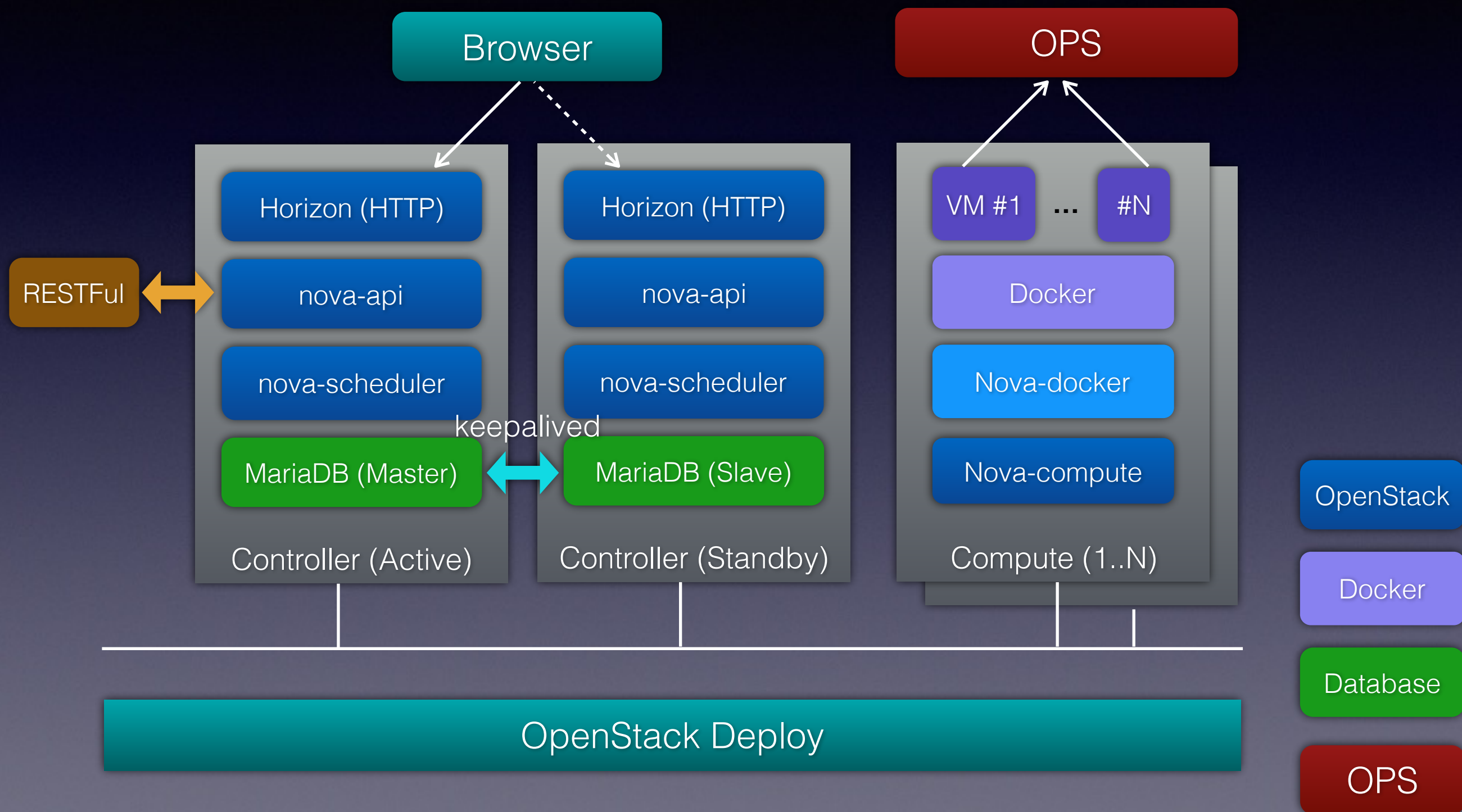
+

	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



逻辑架构图



nova-docker

- spawn: create -> start -> post_start docker
- 容器启动前**做的事情**:
 - 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, 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

- mount /cgroup到容器内。
- 容器启动时用docker exec创建一个文件记录每个容器使用的cpuset, 实例uuid等信息。
- 修改uptime, top, free, df, tsar等源码, 读取/cgroup下的cpu / memory/IO值信息。
- 曾经尝试过lxcfs, 但问题很多, 限制也很多。

```
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	<u>/usr/bin/container_load_calc</u>
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	nsld	20	0	433m	2096	1176	S	0.0	0.0	0:02.00	/usr/sbin/nsld
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
380	nobody	20	0	6416	748	532	S	0.0	0.0	0:00.00	/usr/local/sbin/dnsmasq

实时监控和报警



网络

- NAT — 20% performance lost
- Host mode? No network isolation
- Linux bridge **without** iptables
- OVS VLAN **without** iptables
- other_args="—**bridge=none**"

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。

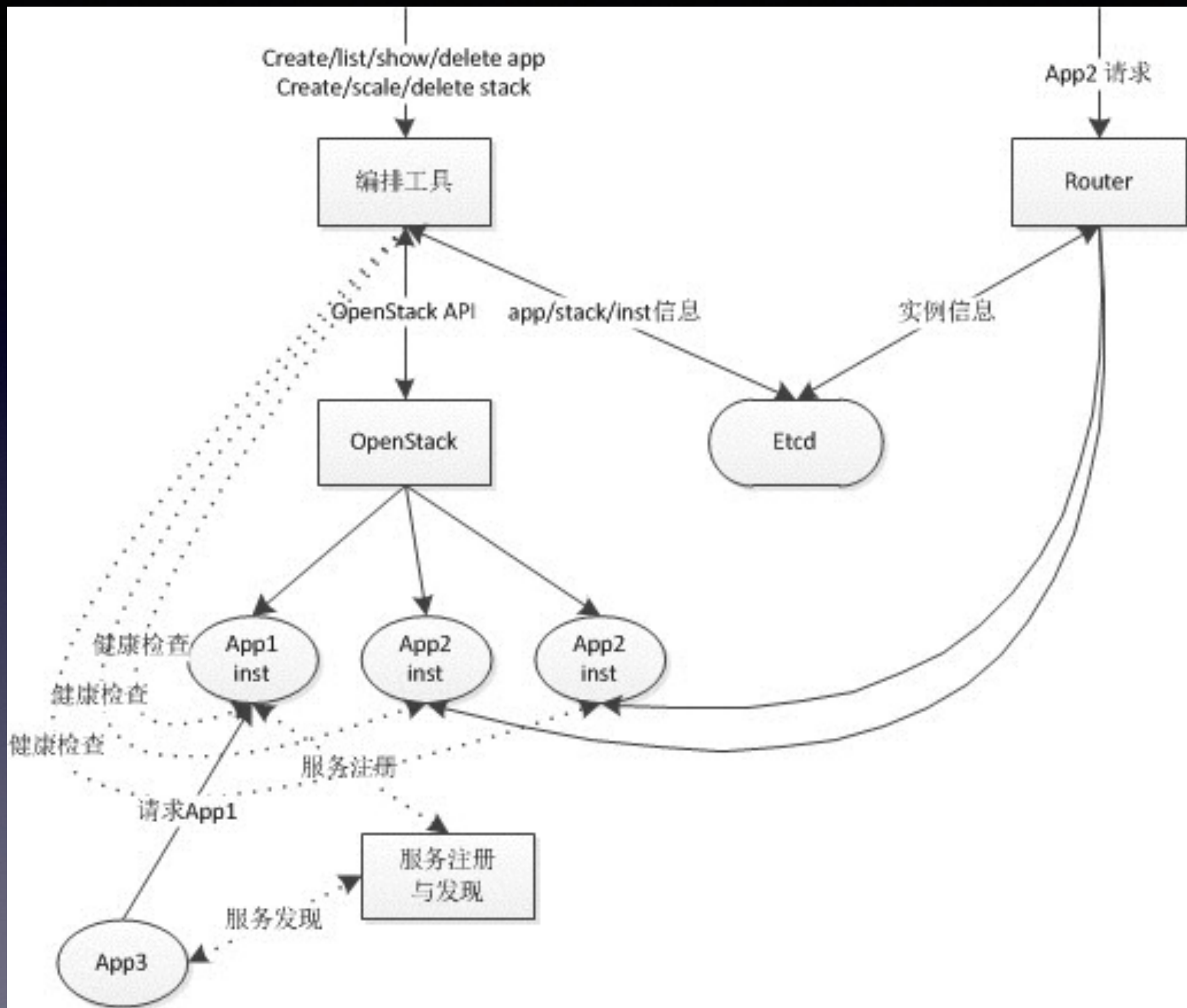
<http://mogu.io/127-127>

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

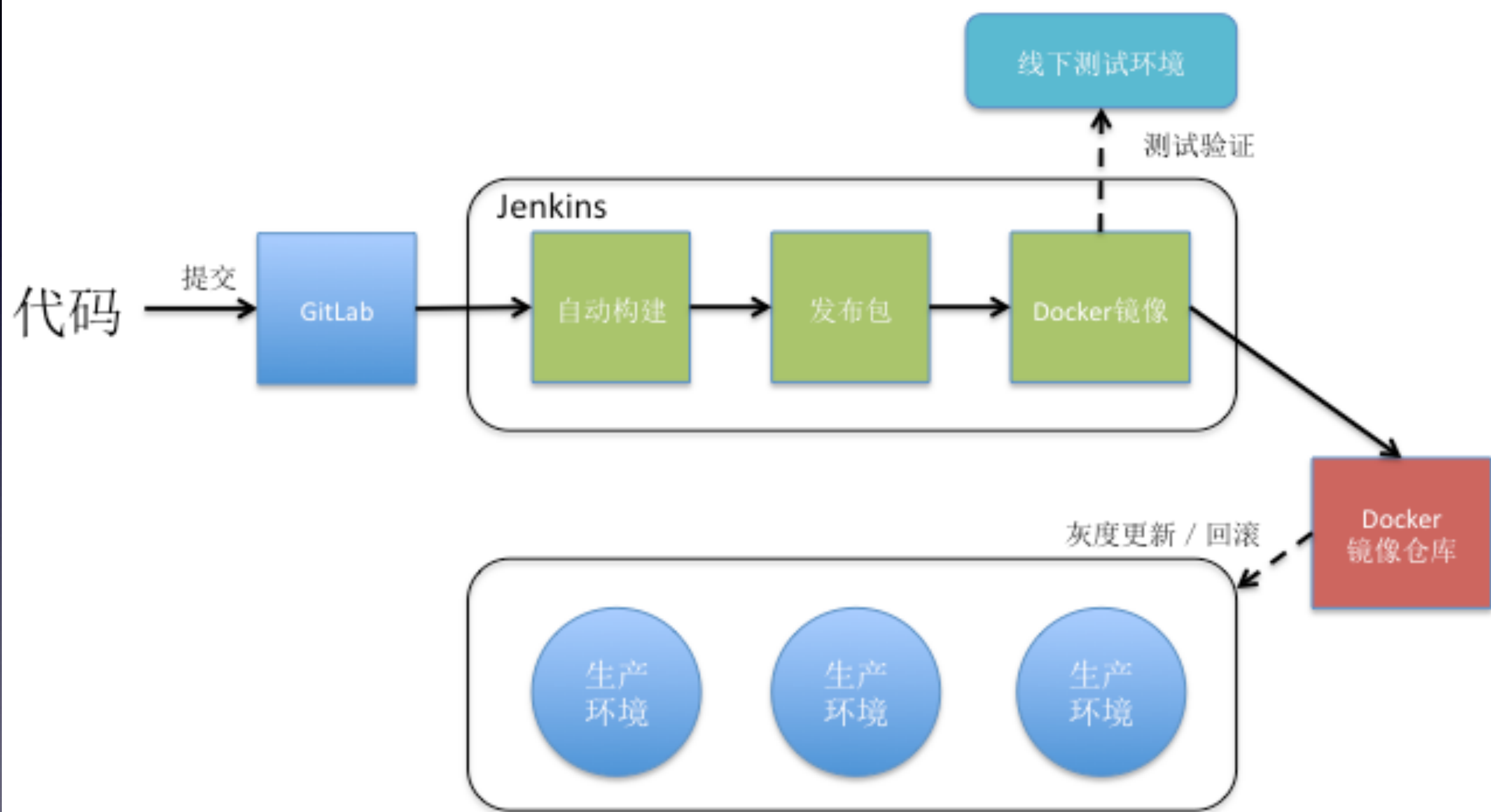
Search Images...

Registry registry.service.mogujie.org

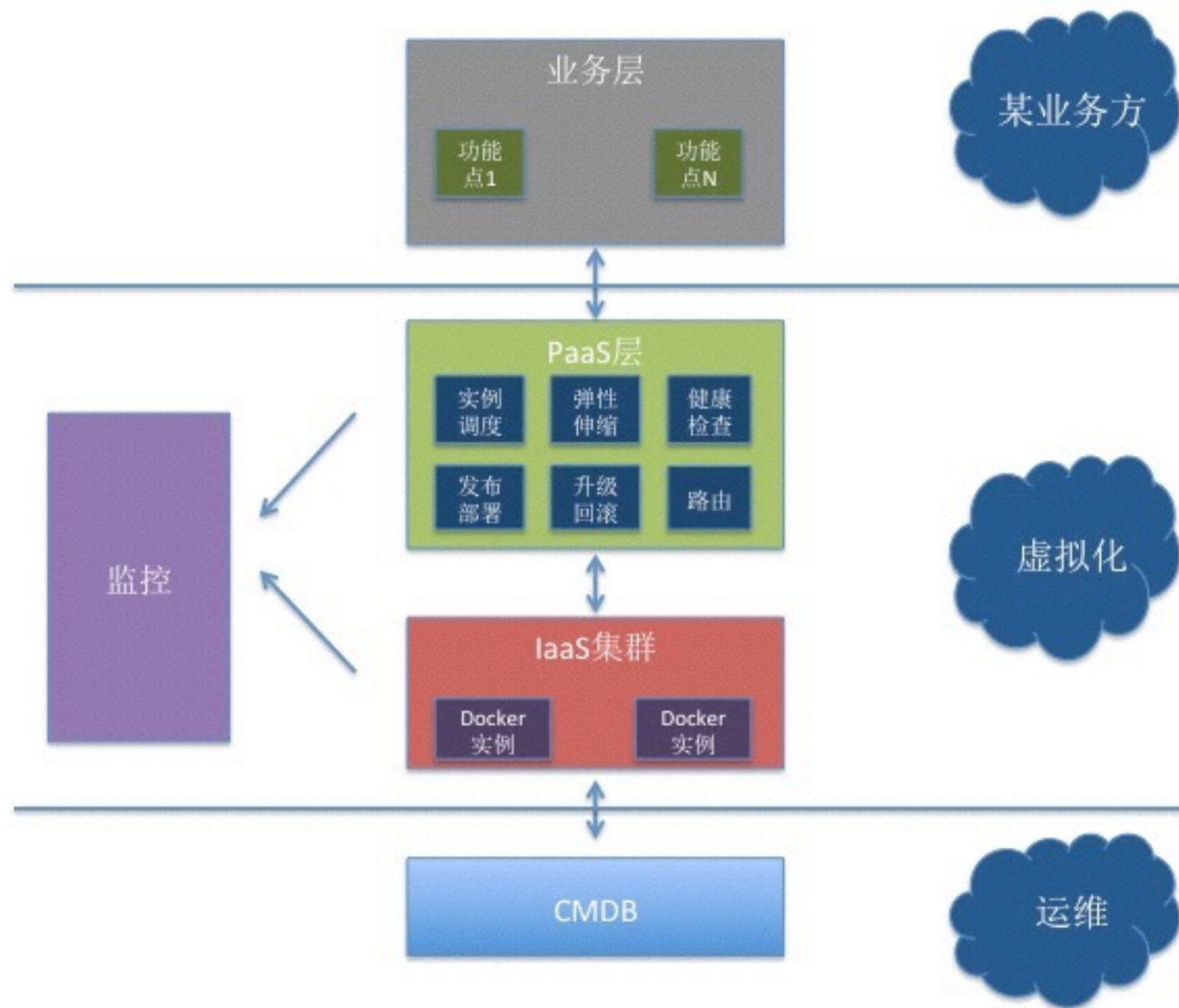
Tag

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

- devicemapper中thin-provisioning的discard功能引起的kernel crash

```
<4>Process dm-thin (pid: 30790, threadinfo ffff880872458000, task ffff8804126a8080)
<4>Stack:
<4> ffff880872459b90 ffff880872459ce0 ffff880a146e2640 ffff880872459c68
<4><d> ffff8802f85d8000 00000000000013a5a ffff880872459ca0 ffffffff801e4876
<4><d> ffff880872459bf0 ffffffff81058d53 ffff88022a09f400 ffff880870d7cd14
<4>Call Trace:
<4> [<ffffffffff801e4876>] remove_raw+0x5f6/0x810 [dm_persistent_data]
<4> [<ffffffffff81058d53>] ? __wake_up+0x53/0x70
<4> [<ffffffffff81266c40>] ? generic_make_request+0x240/0x5a0
<4> [<ffffffffff801e4b40>] dm_btree_remove+0xb0/0x150 [dm_persistent_data]
<4> [<ffffffffff801fb727>] dm_thin_remove_block+0x87/0xb0 [dm_thin_pool]
<4> [<ffffffffff801f7702>] process_prepared_discard+0x22/0x60 [dm_thin_pool]
<4> [<ffffffffff801f6a37>] process_prepared+0x87/0xa0 [dm_thin_pool]
<4> [<ffffffffff801f93e0>] ? do_worker+0x0/0x260 [dm_thin_pool]
<4> [<ffffffffff801f9432>] do_worker+0x52/0x260 [dm_thin_pool]
<4> [<ffffffffff801f93e0>] ? do_worker+0x0/0x260 [dm_thin_pool]
<4> [<ffffffffff81094d10>] worker_thread+0x170/0x2a0
<4> [<ffffffffff8109b290>] ? autoremove_wake_function+0x0/0x40
<4> [<ffffffffff81094ba0>] ? worker_thread+0x0/0x2a0
<4> [<ffffffffff8109aee6>] kthread+0x96/0xa0
<4> [<ffffffffff8100c20a>] child_rip+0xa/0x20
<4> [<ffffffffff8109ae50>] ? kthread+0x0/0xa0
<4> [<ffffffffff8100c200>] ? child_rip+0x0/0x20
```

DOCKER_STORAGE_OPTIONS="--storage-opt dm.mountopt=nodiscard --storage-opt dm.blkdiscard=false"

http://mogu.io/docker_crash-79

遇到的坑 - 隔离

- 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

- 容器内的内存值计算不准确，比实际低一个量级
- `/cgroup/memory/docker/id/
memory.usage_in_bytes`

体会和思考

- 相比KVM，容器技术还有不完善的地方。
- 容器下的运维手段和运维经验的冲击。

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

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

“欢迎您加入蘑菇街!”

简历请发至 guojia@mogujie.com