# Docker与Kubernetes的研究与实践

第四组： dezhiliu gerryyang kevintzhang willqin xixiong

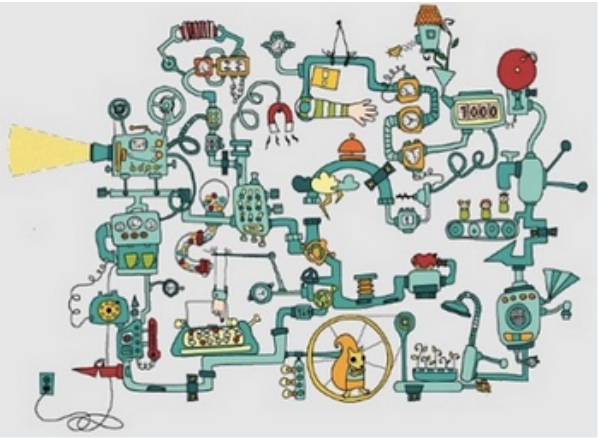
**目录**

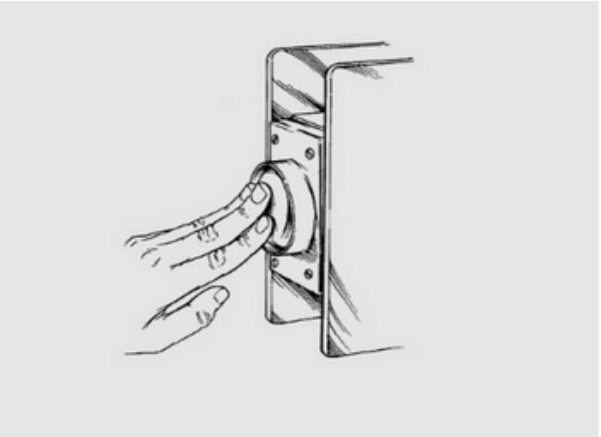


---

# 1 项目背景

你是否遇到过这样的困境：DTO环境迥异，可能导致最终交付不一致；机器类型不同，可能出现资源利用不均衡；机器数量众多，可能带来系统发布不高效。开发侧重于关注应用的运行环境，版本质量，开发效率，而运维更加关注机器资源，监控，容量变更 ，自动化运维。开发和运维之间如何更好地紧密协作，是否存在一套方法论，让开发和运维之间的交互标准化，自动化呢？容器技术成为构建大型分布式系统的基础，而开源的Docker和Kubernetes使得容器的生成，发布，调度等变得非常方便。

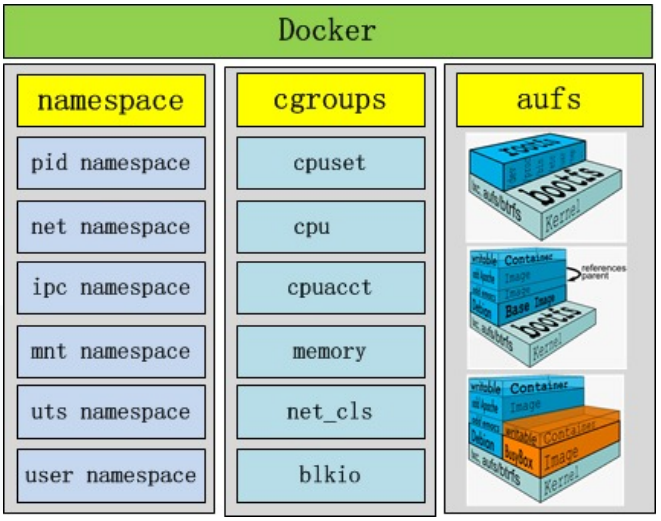




# 2 项目目标

利用Docker容器化技术，将 系统容器化 = **Runtime** + **Image分发机制** + **Orchestration**

- 解决环境复杂依赖
- 更多粒度资源隔离
- 容器编排秒级发布




通过对Docker的调研，我们认为目前Docker最适合的使用的场景有：

> 1. **无状态的服务**
> 2. **需要经常部署的系统或模块**
> 3. **测试环境**

# 3 项目挑战与难点

在系统容器化的过程中，我们可能遇到 哪些问题 ？

## 3.1 基础

- **宿主机的选择**

> 物理机/虚拟机（Xen, VMWare, KVM, etc.）+ Docker？对于高I/O要求的业务，例如数据库服务，建议使用物理机；而很多企业先使用OpenStack构建IaaS(Infrastructure as a Service)，然后在IaaS的基础上构建容器服务，以及在多租户场景，虚拟机的多租户强隔离特性，保证租户在拥有虚机root权限的同时，其他租户和主机的安全，此类场景下建议使用虚拟机。例如，Gaia + 游戏云基于腾讯云，阿里容器服务基于阿里云ECS。

- **操作系统的要求**

> Docker官方建议，Linux操作系统，`内核版本大于3.8`。因此，在公司内Docker的宿主机需要使用TLinux 2.0。

- **基础镜像的选择**

> 基础镜像建议选择架平提供的，目前主要提供TLinux版本的基础镜像，因此容器化的系统需要`支持TLinux环境编译`。

- **镜像如何存放**

> Docker镜像默认存储目录在`/var/lib/docker/`，CentOS下修改/etc/sysconfig/docker配置文件通过`-g`将其挂载到`/data`分区下，避免占用root分区空间。

- **镜像升级问题**

> Docker升级1.10+迁移镜像问题，如何避免升级过程中服务不可用。如果镜像数据比较大，那么迁移会占用一些时间(100MB/s)，并且在升级这段时间内，Docker daemon是不会对外服务的。如果想缩短升级时间，可以使用v1.10-migrator工具，或者使用包含此工具的容器。

- **Docker daemon的缺陷**

> - Docker创建的容器里没有init进程，会出现僵尸进程，导致daemon异常退出。
> - 垃圾文件不清理，导致磁盘inode被耗尽。
> - atomic issues，可能产生死锁。
> - daemon重启，container要重启，对于在线业务不可接受。

- **Dockfile的问题**

> - 不会写，学习成本。
> - 三行命令 = FROM + ADD + CMD，建议尽量简化。

- **单/多租户**

Docker本身安全隔离是基于Linux内核的 `Namespace` 和 `CGroups` 提供的隔离和资源调度机制。所有进程运行在同一个内核中，这对于多租户环境而言，目前会存在安全性不足的问题。例如：

- 由于Docker容器中/proc下数据是通过mount bind宿主机中proc得到的，而内核中proc文件系统大部分没有实现Namespace功能，仅有pid和net实现了。故容器中/proc/meminfo显示的是宿主机的信息，而非容器实际使用的信息。此问题会导致公司网管系统TNM2无法使用。可查看宿主机（172.27.208.238）和容器（172.27.208.135）的资源使用对比。
- 有些操作必须在宿主机上完成。
- 特殊系统对内核的要求。

## 3.2 网络

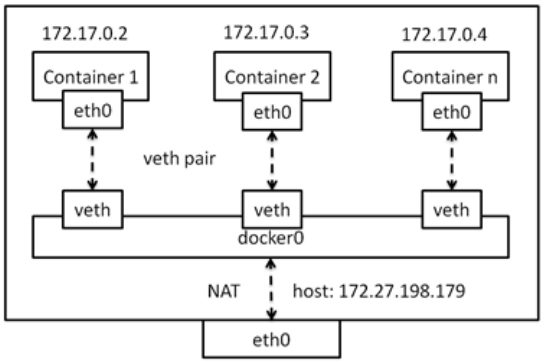
- **NAT**

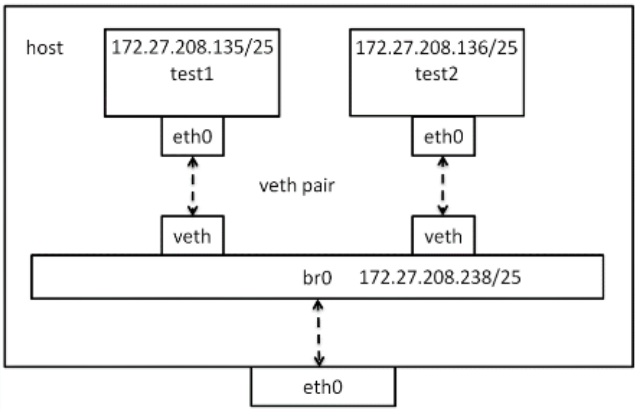
Docker提供了四种网络通信方式，默认使用 `NAT模式` 。但是鉴于安全的考虑（入侵追溯难度大），以及对网络性能方面的影响（相差10%左右），生产环境不建议使用此方式。

- **Bridge + VLAN**

根据公司网络环境，建议采用 `NONE模式` （固定IP网络模式）。

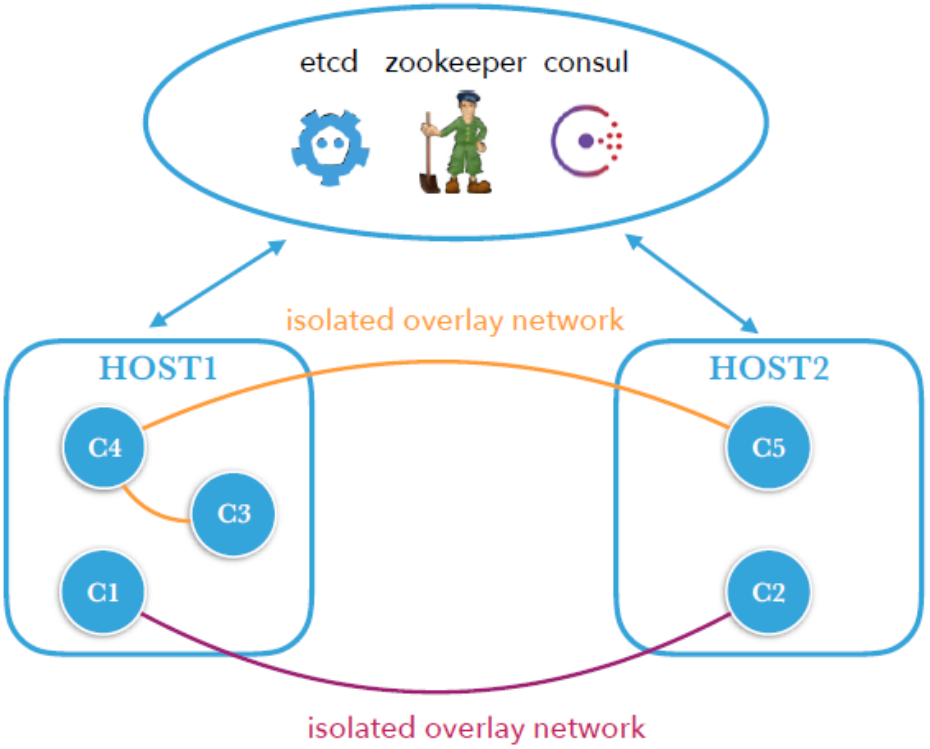
- 使用桥接方式。将宿主机的网卡桥接到虚拟网桥中，再给Docker容器分配一个本地局域网IP。即，将Docker容器网络配置到本地主机网络的网段中，以实现节点之间、各节点与宿主机以及跨主机之间的通信（二层通信）。
- 通过网平申请内网IP，并为需要创建的每个容器绑定一个独立IP。
- 在出现单机故障时，可以通过上层调度系统（比如TSM）实现IP漂移功能。




- **Overlay Network**

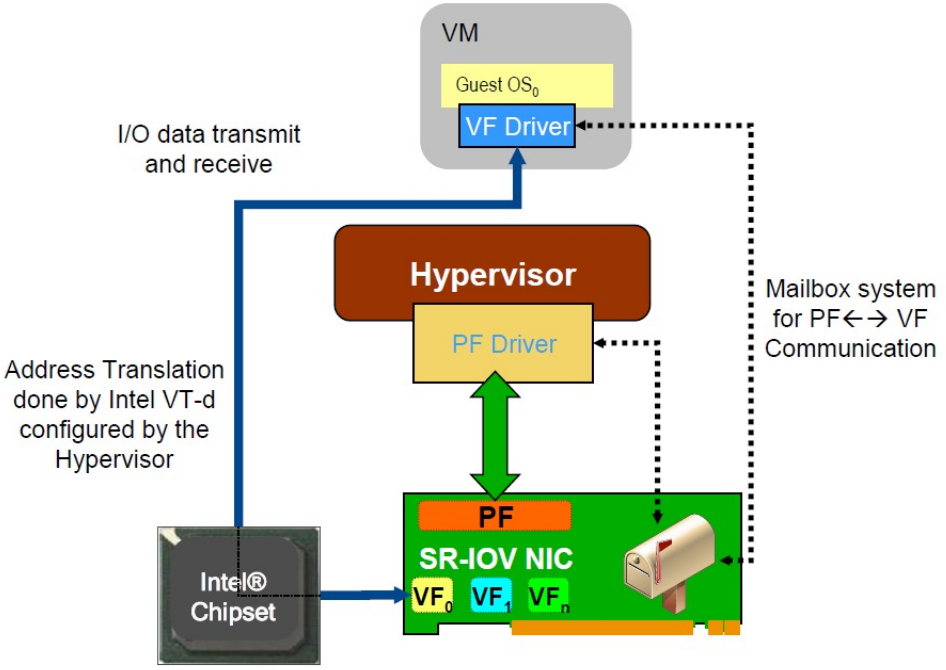
`Docker1.9` 推出了 `Overlay Netowrk` ，底层使用VxLan（Virtual eXtensible Local Area Network），通过MAC in UDP的方式对以太网数据帧进行封装实现大二层网络通信，通过创建一个Overlay网络，实现跨宿主机的容器之间通信。在VxLan的模式下，所有的宿主机或者容器在逻辑上连接到同一个具体的交换机上的，在这个交换机上有VxLan的划分，在同一个VxLan内IP是可以随主机或者容器飘移的。




- **SR-IOV**

硬件虚拟化技术。

- 一个网卡硬件虚拟出多个功能接口，每个功能接口可以作为一个网卡使用。硬件取代内核的虚拟网络设备，极大地提高网络性能，并且减少了物理机的CPU消耗。优点是，既保留了直接I/O 访问的高性能，又保证了虚拟化技术的优势，即设备的共享性；缺点是，需要特殊硬件的支持，对于给定的物理设备，其所能虚拟出来的虚拟设备数量是一定的。
- Docker 1.9.0+版本支持网络插件方式，可以实现SR-IOV的网络插件。
- 具体性能数据可参考Docker容器网络性能(SR-IOV)测试。
- 支持SR-IOV的设备结构。




Docker中使用SR-IOV：

```
激活VF
# echo "options igb max_vfs=7" >> /etc/modprobe.d/igb.conf
# reboot

设置VF的VLAN
# ip link set eth1 vf 0 vlan 12

将VF移到container network namespace
# ip link set eth4 netns $pid
# ip netns exec $pid ip link set dev eth4 name eth1
# ip netns exec $pid ip link set dev eth1 up

In container:
设置IP
#ip addr add 10.217.121.107/21 dev eth1

网关
#ip route add default via 10.217.120.1
```

- **HOST**

> 性能不受影响，不趟SDN浑水。

# 3.3 存储

- **存储驱动哪种合适**

> 目前Docker Graphdrivers支持的类型有六种，但是每种都类型都存在一些问题。在启动Docker daemon时使用`docker -d -s some_driver_name`来指定使用的存储驱动。
>
> 下面是`存储驱动的创建过程`：
>
> - 依次检查环境变量DOCKER_DRIVER和DefaultDriver是否存在，若存在则根据驱动名称调用对应的初始化方法创建一个对应的Driver对象。
> - 若第一步没查到，则Graphdriver会从驱动的优先列表中查找一个可用的驱动。
> - 若仍没找到可用的，则Graphdriver会查找所有注册过的驱动，找到第一个注册过可用的并返回。

```
// driver_linux.go:51
// Slice of drivers that should be used in an order
priority = []string{
    "aufs",
    "btrfs",
    "zfs",
    "devicemapper",
    "overlay",
    "vfs",
}
```

- **AUFS**

> Docker最先使用aufs，但在公司内并不完美。根据最新tkernel2 changelogsTLinux内核更新和BUG日志，aufs目前还存在很多问题，因此，在生产环境暂时不建议使用。
>
> - 容器中AUFS的挂载点是：`/var/lib/docker/aufs/mnt/$CONTAINER_ID/`
> - AUFS的分支(只读和读写)位置
>   在：`/var/lib/docker/aufs/diff/$CONTAINER_OR_IMAGE_ID/`

```
[root@TENCENT64 /data/home/gerryyang/mini_docker/bin]# docker info
Containers: 13
Images: 25
Storage Driver: aufs
 Root Dir: /data/home/gerryyang/root_docker/aufs
 Backing Filesystem: extfs
 Dirs: 51
Execution Driver: native-0.2
Kernel Version: 3.10.83-1-tlinux2-0021.tl2
Operating System: Tencent tlinux 2.0 (Final)
CPUs: 4
Total Memory: 7.665 GiB
Name: TENCENT64.site
ID: J2W3:5VDW:KED5:4TSX:C2KL:J3NU:4SU7:MN7Y:KDTL:ST23:FHG6:FDSF
```

- **DeviceMapper**

> DM相对比较稳定，但是也存在`一些问题`。
>
> - 默认一个容器最大存储空间`不超过10G`，数平基于1.9版本修改了Docker，通过docker run启动参数扩展指定的存储空间。官方支持要到正式版本1.12发布。
> - DM默认使用`loop-lvm`，虽然配置简单，但是性能不好。建议使用`direct-lvm`，在高负载环境下会有更好的性能，具体可以参考Comprehensive Overview of Storage Scalability in Docker。官方也是不建议生产环境系统使用默认的`LVM thin pool/sparse file`。
> - docker -d –storage-opt dm.datadev=/dev/sdb1 –storage-opt dm.metadatadev=/dev/sdc1。

```
[root@TENCENT64 /var/lib/docker]# docker info
Containers: 0
Images: 0
Storage Driver: devicemapper
 Pool Name: docker-8:4-21661799-pool
 Pool Blocksize: 65.54 kB
 Backing Filesystem: extfs
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 307.2 MB
 Data Space Total: 107.4 GB
 Data Space Available: 107.1 GB
 Metadata Space Used: 733.2 kB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.147 GB
 Udev Sync Supported: true
 Data loop file: /data/home/gerryyang/root_docker/devicemapper/devi
cemapper/data
 Metadata loop file: /data/home/gerryyang/root_docker/devicemapper/
devicemapper/metadata
 Library Version: 1.02.84-RHEL7 (2014-03-26)
Execution Driver: native-0.2
Kernel Version: 3.10.83-1-tlinux2-0021.tl2
Operating System: Tencent tlinux 2.0 (Final)
CPUs: 4
Total Memory: 7.665 GiB
Name: TENCENT64.site
ID: J2W3:5VDW:KED5:4TSX:C2KL:J3NU:4SU7:MN7Y:KDTL:ST23:FHG6:FDSF
[root@TENCENT64 /data/home/gerryyang/root_docker/devicemapper/devic
emapper]# ls -lsh
total 294M
293M -rw------- 1 root root 100G Apr 22 16:18 data
752K -rw------- 1 root root 2.0G Apr 22 16:18 metadata
```

- **Docker Volume**

> 通过Docker Volume，把宿主机上的目录挂载到容器内部，可以实现持久存储。但是存在一些问题：
>
> - 容器迁移，数据无法迁移。
> - 多个容器之间不能共享数据。

- **Docker + Ceph**

> IEG的TDocker通过Ceph的RBD递归快照(thin-provisioning snapshot)，支持Docker rootfs数据的共享存储，但是，目前还不支持元数据的共享存储。若实现后，可以实现容器故障无数据迁移。

- **Registry**

> 镜像仓库需要考虑的问题。
>
> - 安全鉴权。例如，`docker pull 172.27.198.179:5000/xxx`
> - 负载容灾。

## 3.4 监控

- **docker stats**

> 在宿主机上通过docker stats可以查看每个容器的`CPU利用率`、`内存的使用量`、`可用内存总量`以及`网络发送和接收数据总量`。可以使用Docker Remote API获取更详细的stats信息。
> **优点**：易于使用和部署。
> **缺点**：没有监测非Docker资源的能力。

```
CONTAINER           CPU %               MEM USAGE/LIMIT     MEM %
             NET I/O
portal_tdf          0.00%               552 KiB/7.665 GiB   0.01%
             0 B/0 B
```

- **cAdvisor + Prometheus**

> Google用来分析运行中Docker容器的资源占用以及性能特性的工具。通过cAdvisor实时监控容器的资源使用情况，并将数据上报监控平台。IEG使用cAdvisor对CPU/MEM/NETWORK的采集算法。
> Prometheus实现告警和过滤。
> **优点**：提供容器监控可视化图表。
> **缺点**：没有监测非Docker资源的能力，只能监控一台宿主机，占用一定资源消耗，上报TNM2网管系统。

- **FUSE**

通过FUSE filesystem(Filesystem in Userspace)解决容器内资源使用显示不正确的问题，从而可以无缝对接TNM2网管系统。
原理：通过FUSE实现用户态的fs，使用cgroups的数据统计container的实际资源使用，然后生成仿真的meminfo,cpuinfo,stats,diskstats文件，bind mount到container中。

- 需要安装 `fuse`。例如，sudo apt-get install fuse。
- 需要Docker升级到1.11+，否则老版本需要打patch。

**优点**：与老的监控系统兼容，对业务使用者透明。
**缺点**：对Docker版本有要求或者修改Docker。

```
# 测试：创建一个内存限制为15MB的容器，并在宿主机和容器内分别查看内存使用情况
container_id=`docker create -v /tmp/cgroupfs/meminfo:/proc/meminfo
-m=15m ubuntu sleep 213133`
root@gerryyang:/tmp# docker start $container_id
e1379cf7e3ba4611201699e77cac28417bf98260b9e0ca07b0f9579b575a6e3c
root@gerryyang:/tmp# cat /tmp/cgroupfs/meminfo
MemTotal:       15360 kB
MemFree:        15144 kB
MemAvailable:   15144 kB
Buffers:        0 kB
Cached:         124 kB
SwapCached:     0 kB
root@gerryyang:/tmp# docker exec -it $container_id bash
root@e1379cf7e3ba:/# free -m
             total       used       free     shared    buffers
cached
Mem:            15          0         14          0          0
     0
-/+ buffers/cache:          0         14
Swap:            0          0          0
```

- **内核支持**

架平目前是通过修改内核来解决Docker容器内资源的显示问题，具体可参考Tkernel2新特性之Docker容器内资源展示隔离。
**优点**：简化业务使用，与老的监控系统兼容。
**缺点**：业务如需在Docker环境下使用该特性，需要自行修改Docker添加相关的mount bind操作。

## 3.5 安全

Docker团队已经明确了一些安全问题，正在着手解决。借助docker-bench-security工具可以检查Docker环境下是否存在的一些安全隐患。

```
docker run -it --net host --pid host --cap-add audit_control \
    -v /var/lib:/var/lib \
    -v /var/run/docker.sock:/var/run/docker.sock \
    -v /usr/lib/systemd:/usr/lib/systemd \
    -v /etc:/etc --label docker_bench_security \
    docker/docker-bench-security
# ------------------------------------------------------------------------
-------------
# Docker Bench for Security v1.0.0
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docke
r containers in production.
# Inspired by the CIS Docker 1.11 Benchmark:
# https://benchmarks.cisecurity.org/downloads/show-single/index.cfm
?file=docker16.110
# ------------------------------------------------------------------------
-------------

Initializing Thu May  5 02:12:34 UTC 2016


[INFO] 1 - Host Configuration
[WARN] 1.1  - Create a separate partition for containers
......
```

- **案例**：获取百度机器root权限

  > 通过利用Docker Remote API借助Docker Swarm开放的 2375 端口，获取用户信息，并利用docker命令和volume的方法可进一步获取宿主机的root权限。

```
root@gerryyang:~/Docker# docker -H tcp://104.131.173.242:2375 ps
CONTAINER ID        IMAGE                          COMMAND
   CREATED             STATUS              PORTS
NAMES
34efbcce4513        ubuntu_sshd_gcc_gerry:14.04    "/run.sh"
   47 minutes ago      Up 47 minutes       0.0.0.0:10022->22/tcp
small_payne
```

## 3.6 编排

容器多了，如何管理？资源调度管理需要解决的问题：对机器资源CPU、存储、网络的隔离进行弹性管理，最大化地利用集群的所有资源。业务不用关心机器申请、程序的交付和部署、系统扩缩容等，而只需要构建好镜像，在Web页面填上需要申请的资源、副本数量，其它的事情全部交给调度平台完成。

- **Swarm**

> Docker社区原生，优势是与Docker接口API统一

- **Kubernetes**

> Borg -> Omega -> K8S

- **MM**

> Mesos + Marathon

- **Docker Remote API**

> 通过API实现自定义编排

- **Deis/Flynn**

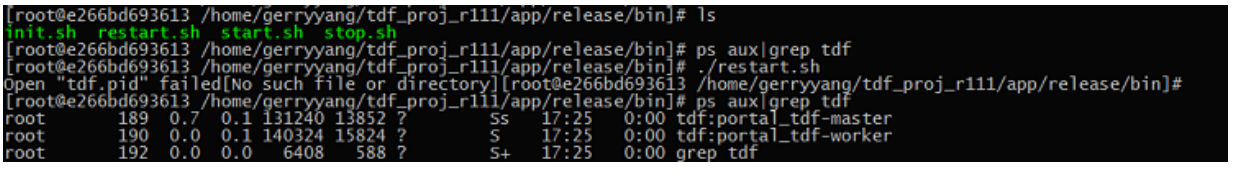
> 其他一些管理容器的开源PaaS平台，Deis是基于Python开发的，而Flynn使用Go

# 4 实践

## 4.1 Portal TDF容器化

通过将Portal容器化，测试系统的基本功能是否符合预期。具体参考Portal TDF Docker容器化实践。

- 测试Portal在容器中的各项功能：



- 将最终测试OK的容器打包成镜像：

```
[root@TENCENT64 ~/gerryyang/bin]# docker images
REPOSITORY                                  TAG                     IMAGE ID            CREATED             VIRTUAL SIZE
portal_tdf                                  tdfr111_portalr230d001  45d0bf89afcc        About an hour ago   1.765 GB
ssh                                         v1                      da5f770f3c0c        5 months ago        1.406 GB
tlinux-sles10sp2-64bit-1.0.13.20150610.sqfs latest                  7ac70b8c3b80        8 months ago        1.384 GB
registry                                    latest                  95099732d4f8        16 months ago       422.9 MB
sles10sp2-32bit-v3.2.20121023.sqfs          latest                  63ed910ab26c        18 months ago       1.344 GB
tlinux-64bit-v1.2.20140902.sqfs             latest                  f8921344038d        18 months ago       1.401 GB
[root@TENCENT64 ~/gerryyang/bin]#
```

## 4.2 编排工具

Docker集群管理工具K8S部署和测试。

- 部署K8S集群：

```
Found 1 node(s).
NAME              LABELS                                   STATUS    AGE
104.131.173.242   kubernetes.io/hostname=104.131.173.242   Ready     1s
Validate output:
NAME                 STATUS    MESSAGE              ERROR
controller-manager   Healthy   ok                   nil
scheduler            Healthy   ok                   nil
etcd-0               Healthy   {"health": "true"}   nil
Cluster validation succeeded
Done, listing cluster services:

Kubernetes master is running at http://104.131.173.242:8080

root@gerryyang:~/k8s/test/kubernetes/k8s_1.1.3/kubernetes/cluster#
```

- 获取节点状况：

```
root@gerryyang:~/k8s/test/kubernetes/k8s_1.1.3/kubernetes/cluster/ubuntu/binaries# ./kubectl get node
NAME              LABELS                                   STATUS    AGE
104.131.173.242   kubernetes.io/hostname=104.131.173.242   Ready     51s
45.55.79.174      kubernetes.io/hostname=45.55.79.174      Ready     35s
root@gerryyang:~/k8s/test/kubernetes/k8s_1.1.3/kubernetes/cluster/ubuntu/binaries#
```

# 5 其他

**公司内部**：

- 数平的Docker on Gaia，底层资源管理和调度系统，提升公司的整体服务器利用率。基于Docker社区版本的自研定制开发，将自研的新特性尽可能提交社区。
- 基架的CAE over Docker，腾讯内部云的业务托管平台，提升设备利用率，秒级扩容速度，配置环境标准化，并使用自研的调度系统。将互娱官网原有的1500台TVM虚拟机替换为docker虚拟化容器，单机服务能力提升15%。
- 架平的CI自动化测试环境Docker化，解决设备环境冲突问题，将问题环境生产镜像便于开发定位。
- IEG的TDocker云平台 ，为游戏所用，Tencent Docker for Online Service。
- MIG的Sumeru项目支撑业务在Docker平台上运行，对业务接入要求是业务是否能随意放置到任何IP端口上运行，否则需要业务进行改造。
- OMG的视频索引docker容器部署规范和性能测试报告。

**公司外部**：

- 阿里云容器服务，容器服务基于阿里云ECS。
- 平安科技，修改MM框架，mesos_dns，mesos_lb不适用，与内部DNS接口对接实现LB，Mesos+Docker+Marathon+ELK+DockerUI。
- DaoCloud，主打Docker原生工具。
- 数人云，基于Mesos 和 Docker 技术打造下一代DCOS，将应用弹性做到极致，提供容器化整体解决方案。

# 6 结论

| 应用类型 | 例子 | 收益 | 成本 |
| --- | --- | --- | --- |
| 无状态 | Web接入层 | 快速部署、弹性伸缩、自动容错、高利用率 | 制作自己的image，容器化配置分离 |
| 有状态 | DB存储层 | 快速部署 | 更高的失败率 |

在`计平的业务场景`下，我们建议：

- **在线实时核心计费系统，目前`不建议`使用容器化**
    - 系统稳定性是最重要的，资源利用率并不是计费场景下最关键的。
    - 在线系统涉及外部接口较多，权限申请是热点，没有很好的体现容器快速部署的优势。
    - 核心计费系统监控至关重要，目前已有的监控体系已经比较完善，引入容器化会提高监控的复杂度。
    - 计费后台系统很少存在异构的模块，TDF开发框架也逐渐成熟，成为后台开发的标配。
- **离线数据计算系统，`可以考虑`引入容器化**
    - 对资源要求比较高，且不需要长期运行，不同的业务可以共享机器资源。
    - 离线模块需要的计算节点比较多，使用容器方便快速部署。
    - 离线模块可以承受一定的容错性。

# 7 参考

[1] https://github.com/docker
[2] https://github.com/kubernetes/kubernetes
[3] docker-ecosystem