

Docker 企业级架构篇

前言：

随着互联网技术的进步与发展，日益精细化的需求为我们带来挑战也带来了机遇。一个问题提出，同时也有更多对应的解决方法和方案出现，这是社会不断进步的规律。因此，近年来兴起的 Docker 容器技术和基于 Docker 的各平台以及解决方案也逐渐进入我们视野。关于 Docker，已经有很多文章作了相关说明及见解，其入门篇我们已经有了整理，那么今天针对 Docker 在企业级架构中的实战我们也作一次分享。

注：本文部分资料来源于网络，由云舒网络整理发布。



了解最新云计算资讯，关注[云舒网络](#)官方微信

目录

| | |
|---|----|
| 一 . 背景----- | 3 |
| 1. 新 IT 时代的诉求----- | 3 |
| 2. 微服务理念流传----- | 4 |
| 二 . 落地篇----- | 4 |
| 1. Docker Sidekick ----- | 4 |
| 2. DevOps----- | 5 |
| 3. 容器服务企业落地之 Rancher 解决之道----- | 7 |
| 三 . 实战篇----- | 15 |
| 1. Rancher 快速上手指南 (虚拟机篇) ----- | 15 |
| 2. 基于 Docker 的构建流程-持续集成及测试----- | 35 |
| 四 . 番外篇----- | 37 |
| 1. Docker 容器和存储系统之 Root Image 分层文件系统----- | 37 |
| 2. Docker 容器和存储系统之 Volume 存储接口----- | 41 |
| 五 . 结语----- | 44 |

● 背景：

随着 x86 服务器，KVM 虚拟化技术以及 OpenStack 等 IaaS 管理平台的普及和成熟，VM 已经可以在不同 X86 服务器厂商的硬件平台上在线无缝迁移。这就日益形成了硬件差异化不断缩小甚至消失而软件重要性不断提升的一个局面，至此 IT 真正进入“软件定义数据中心”的时代。

1.新 IT 时代的诉求

人们发现，虚拟化后的 IT 在部署简单化，在资源高利用率、高可用性、维护便利性等方面都收获了巨大的收益。但是，随着流程规模化的进一步需求，IT 部门提出了更高的要求：如何将应用在不同操作系统间实现无缝迁移？如何将开发和生产统一实现“一处构建处处运行”？如何能把精力和时间从开发——测试——运维的一系列环境重复搭建和调试之中解放出来？如何使企业从繁杂的基础架构解脱，从而更专注于业务？能否让应用的部署、分发、负载均衡、高可用性、监控运维等方面也与虚拟机一样有更统一和更简单的方法呢？

也因此，一些技术、工具、平台等等应运而生。其中就包括了今天我们重点要讲的 Docker 容器技术。说起容器技术，我们先看看不得不谈的微服务。

2.微服务理念流传

微服务的思路不是开发一个巨大的单体式的应用，而是将应用分解为小的、互相连接的

微服务。微服务带来的好处有：第一软件工程上多部门更好协作，第二软件本身的扩展性更强。一个微服务一般完成某个特定的功能。一些微服务还会发布 API 给其它微服务和应用客户端使用。运行时，一个微服务实例就可以是一个 **Docker 容器**，所以微服务天然需要 Docker，所以说微服务与 Docker 是“相得益彰”的关系。具体如何体现呢？不妨先来看看下文。

● 落地篇：

1. Docker Sidekick

Sidekick 是一种容器，它做了主容器不做的杂事，从而可以让主容器专心的做最核心的事。这个其实也是 Docker 容器非常优美好用的地方。比如说，你起了一个服务进程 A 容器，那么这个容器的主要工作就是服务进程 A，但是还有些辅助的工作。我们可以列举一下：

首先，如果服务进程 A 要备份数据，怎么做呢？容器时代之前，那么大家就进入 shell 上，直接的用 rsync，或者起一个定时备份的进程来备份。

但是容器时代不一样了。记住，每个容器最好只干一件事，然后以服务的形式来对外服务。所以最好的做法是：创建一个 Volume 容器，服务进程 A 容器共享 Volume 容器，然后另外一个备份 Job 容器也共享这个 Volume 容器，然后备份 Job 容器专门来做备份。你看，服务进程 A、Volume、备份，三个容器功能清清楚楚的，相互协作但是又各自独立，

架构上也非常容器理解和把握。对于服务进程 A 容器，其他两个容器就是 sidekick。

其次，如果是要重启服务进程 A 呢？也可以用一个 sidekick 容器来搞定。

比如服务进程 A 创建了一个 /var/run/A.sock，然后提供命令行 A-ctl restart 来重启服务。CID=\$(docker run -d -v /var/run Aservice) 启动 A 服务，拿到容器 CID。然后启动一个 Sidekick 容器来重启主容器 CID 的服务 A。docker run --volumes-from \$CID Aservice A-ctl restart。

所以请记住，容器只把一件事干好，不要把太多东西都放到一个容器里。正确的方式应该是一个主容器带一帮 Sidekick 来提供服务，就像黑帮老大带着马仔出入，这样的效率和架构是最优的。

2. DevOps

在讲述容器技术如何在企业落地之前，请允许我向大家介绍下 DevOps。DevOps (Development Operations) 兴起于互联网公司，基本理念是开发和运维合为一体，把开源工具拿过来再根据自己的业务特点稍加改动，测试通过后就上线支撑公司的产品和服务，并一边运维一边改进。这样做的好处是整个过程将更加流程化和更加方便快捷。因为 DevOps 本身就是一组过程、方法与系统的统称，它的价值在于为了按时交付软件产品和服务、开发和运营工作，部门之间必须紧密合作。也就是说，如果把 Docker 看着是为你提供了“集装箱”，那么 DevOps 的加入使得这些集装箱运作起来更加有序。

在 DevOps 出现之前，开发的工作流程是：市场调研，需求分析，系统设计，软件编码，单元测试，集成测试，系统测试，软件发布。运维人员的工作流程是：安装服务器，安装软件，配置软件，系统运维。很明显在老的模型中，运维人员地位低下，整天做着重复且枯燥的工作，并且开发和运维之间相互等待资源环境、软件版本、以及 Buglist。

所以，能不能使运维专注于提供、维护、监控平台，提供工具，让开发利用运维的工具发布、维护自己的产品或服务呢？互联网的神仙/屌丝们很快发现，Docker 的“一处构建，处处运行”正是他们需要的。那么有了 Docker 又如何呢？

首先，他们可以快速构建，整体打包 (Build)。在这个过程中主要是运维和研发需要更多的协助，编写 Dockerfile 来构建镜像。

其次，快速交付 (Ship)。研发以镜像作为交付，整体交付给测试人员，缩短研发和测试之间处理问题的周期。

最后，快速部署 (Run)。Docker 镜像保证了环境的一致性，让问题在部署前得到解决，更重要的是可以基于镜像，快速部署应用。

好了，到这里，我们已经基本确定，我们的企业需要 Docker。我们可以不关心服务商自己写的还是基于开源平台改的，但是我们要关心他的服务是否可靠、是否稳定、是否便宜、是否安全、是否在我的企业可以落地。那么当下市场上的服务商们是怎么落地的呢？

3. 容器服务如何在企业客户落地—Rancher 解决之道

对企业来说，尤其是对企业的 CIO 来说，首先要考虑的问题就是“容器技术如何在我的企业平稳落地”？

对于非互联网公司的传统企业客户，以及大量围绕企业客户做集成、交付解决方案的服务提供商而言，我们需要考虑的一个问题就是怎么样把容器技术以高质量、低成本、易维护的方式落地到企业的生产环境中来。换句话说，如果把容器技术比做 KVM 和 Xen，我们需要一个容器界的 OpenStack 或是 CloudStack。

Rancher 就是定位在提供“企业私有容器服务”这一核心业务需求上，并提出构建“企业私有、混合容器云”；“像 AppStore 一样的企业应用商店实现一键式应用部署”；“CI/CD 部署流水线优化践行 DevOps”以及“轻量级 PaaS 平台”等多个被企业客户所关注的一揽子解决方案。

Rancher 公司是一家位于硅谷的美国公司，创建人梁胜博士和他的团队一直是专注于计算技术在企业落地工作的，梁博士创建的 CloudStack 项目是很多大的公有、私有 IaaS 云的支撑平台，他在早期时还是 SUN 公司 JVM 和 JNI 的开发带头人，所以“云计算”、“企业客户”是 Rancher 公司基因当中的两大关注点。

纯粹的 Docker 和可以落地到企业生产环境的容器平台还是有很大距离的，需要做的工作至少有以下这些方面：

- 容器主机管理
- 容器SDN网络的实现
- 服务注册和发现
- 容器负载均衡
- 容器和服务的健康检查
- 服务升级和灰度发布
- 容器持续存储和数据管理

基础架构服务管理

- 容器生命周期管理
- 容器运行状态监控
- 访问容器的日志
- 访问容器的Shell
- 鉴权系统和账户管理
- 容器环境和多租户

生命周期管理和团队协作

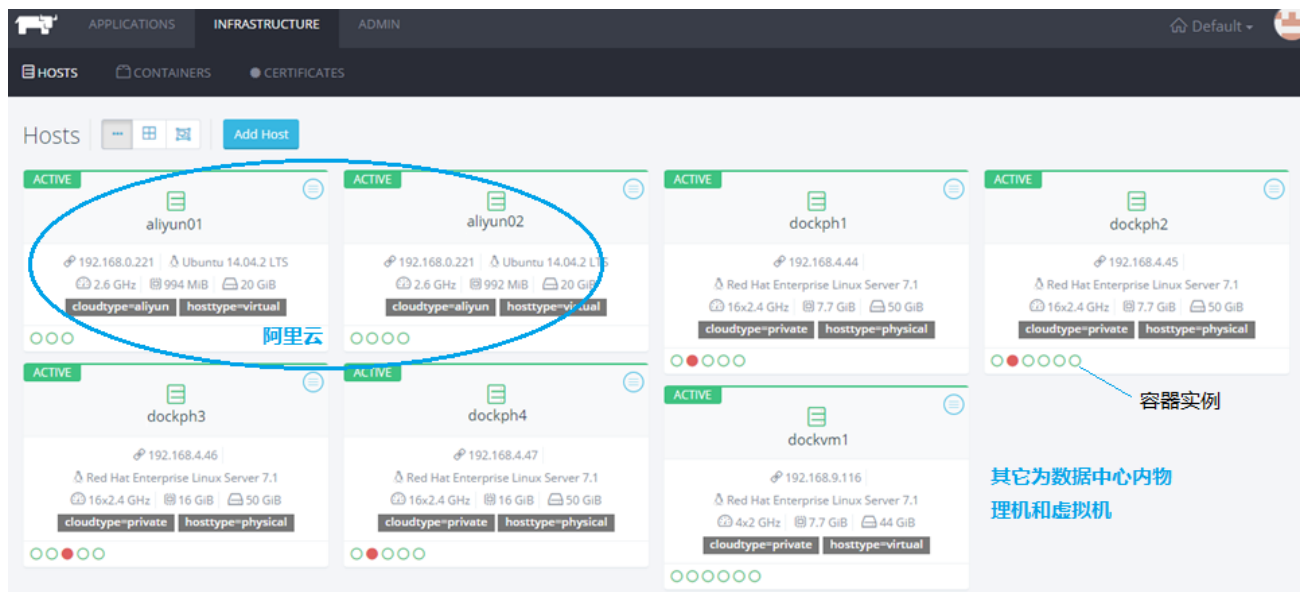
- 应用的配置管理
- 丰富的Compose文件格式
- 企业镜像库管理
- 功能完备的API接口
- 兼容Docker CLI命令行

配置管理和部署支持

© 2015 Rancher Labs, Inc.



举几个例子：Rancher 可以统一管理企业内部多个数据中心的虚拟机、物理机容器环境、以及公有云（阿里、AWS 等）内的容器主机，允许我们通过标签把业务灵活的分配到不同属性的“云”上。



以下调度策略为：把容器运行在阿里云上，并且容器尽量分散在多台阿里云主机，以提高可用性。

ADVANCED OPTIONS ^

Command Volumes Networking Health Check Security/Host Labels Scheduling

☐ Run all containers on a specific host

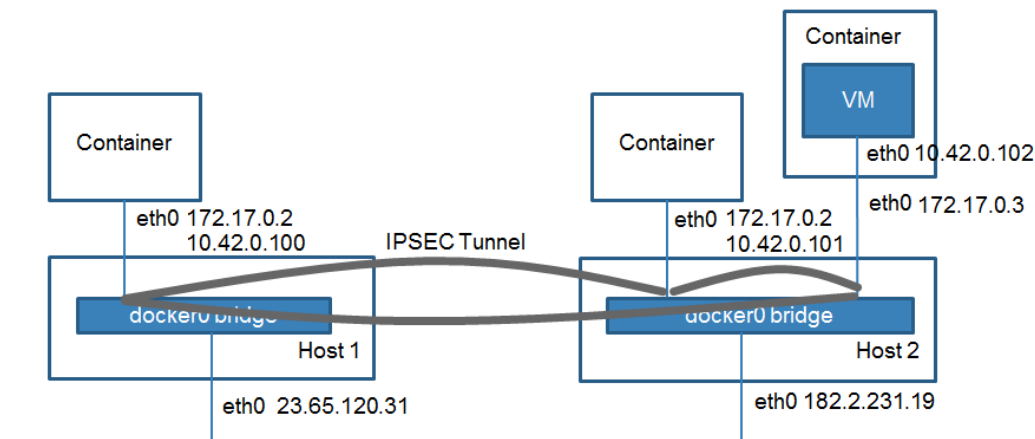
☒ Automatically pick hosts for each container matching scheduling rules:

+ Add Scheduling Rule

| | CONDITION | FIELD | KEY | VALUE |
|----------|-----------|-----------------------------|----------------------------------|-----------------------------------|
| The host | must | have a host label | of cloudtype | = aliyun |
| The host | should n | have a container with label | of io.rancher.stack_service.name | = \${stack_name}/\${service_name} |

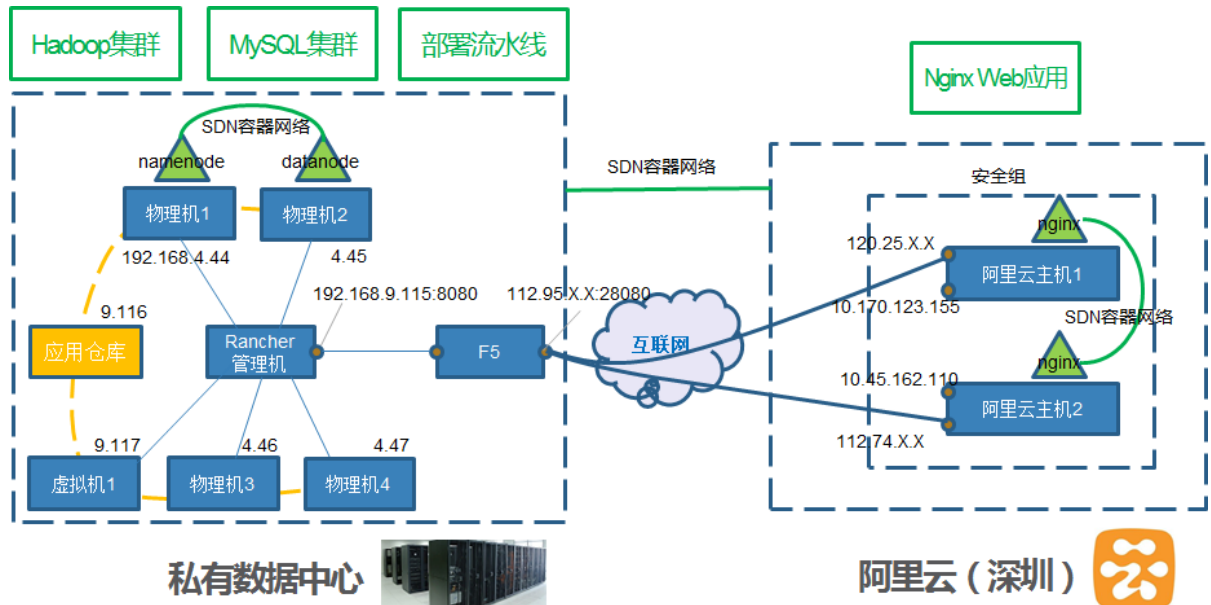
为了实现公有云和私有云间以及同一片云的主机间的容器通讯，Rancher 基于 SDN 技术创建了 overlay 容器网络：

Rancher SDN



9

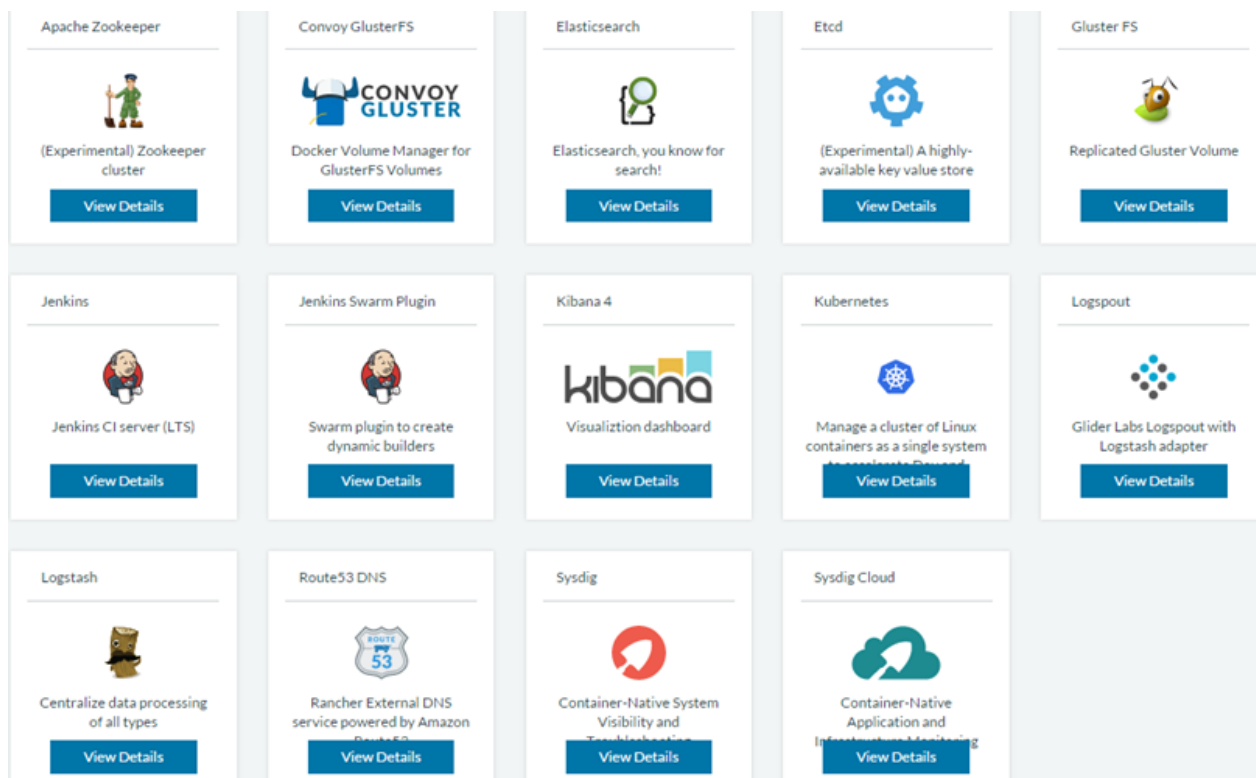
当不同云和不同主机上的容器可以通过容器网络通讯后，再配合 Rancher 实现的负载均衡、服务发现、健康检查机制就可以帮助企业实现快速业务搭建和扩展，手动或是自动的实现容器甚至是容器主机的跨云动态扩容，这一点对“双 11”这样的场景特别有用。



27

企业应用商店和一键部署是另外一个非常强大的功能，这引申出 Rancher 对容器云未来发展方向的一个预见：单纯提供容器编排能力是不够的，提供容器应用的配置管理更能让“以应用为中心”这一容器技术特点发挥得淋漓尽致。因此我们提供了一个开放式的框架，在兼容 docker-compose.yml 的基础上把与应用配置相关的信息记录在 rancher-compose.yml 中，并且允许用户以灵活的方式实现对任何应用的配置管理：你只要提供 docker-compose.yml 和 rancher-compose.yml，Rancher 会自动在应用商店中探测到你上架的应用并支持管理你定义的配置项。

上架应用示例：



应用的配置管理：

Add Hadoop + Yarn Stack

EXPERIMENTAL - Hadoop + Yarn Big Data Tools. (Requires 2+ nodes)

Template Version

2.7.1-rancher1

Select a version of the template to deploy

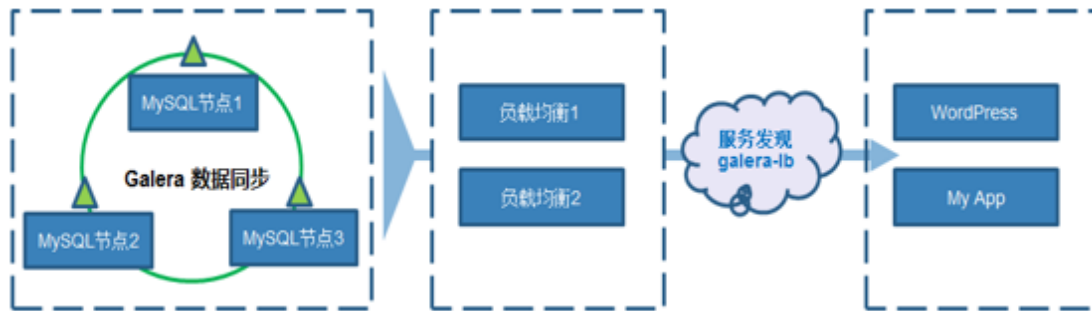
New Stack

| | |
|-------------|--------|
| Name* | hadoop |
| Description | |


Configuration Options

| | | | |
|--|--------|---|----------|
| Cluster Name* | hadoop | Default DFS Replica Count* | 3 |
| Name for the stack volumes | | Default number of HDFS replicas | |
| Yarn Nodemanager CPU vcores* | 8 | Yarn Nodemanager Memory Value* | 8192 |
| yarn.nodemanager.resource.cpu-vcores value | | yarn.nodemanager.resource.memory-mb value | |
| Yarn Minimum Memory allocation* | 1024 | Mapreduce Map Memory* | 1024 |
| yarn.scheduler.minimum-allocation-mb value | | mapreduce.map.memory.mb | |
| Mapreduce Reduce Memory* | 2048 | Mapreduce Map Java Opts* | -Xmx768m |
| mapreduce.reduce.memory.mb | | mapreduce.map.java.opts | |
| Mapreduce Reduce Java Opts* | | | |

基于上述技术，可以做的事情有很多，比如通过一个高可用的 MySQL 服务实现一个轻量级的 PaaS 平台：



接入 SysDig 监控



Add Sysdig Cloud Stack

Container-Native Application and Infrastructure Monitoring

Template Version

v0

Select a version of the template to deploy

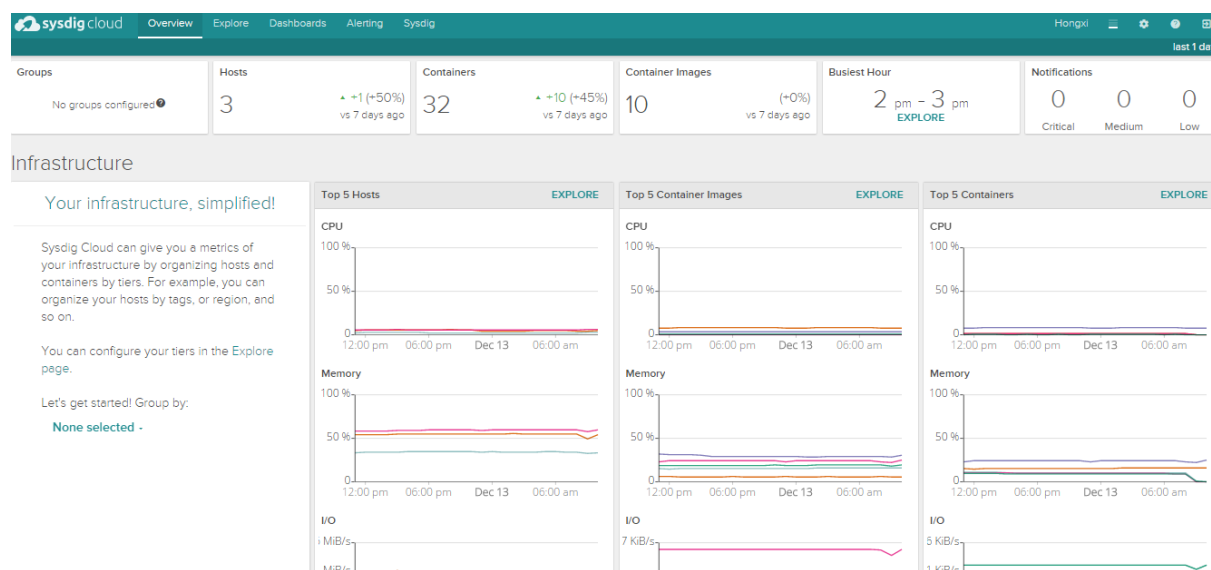
New Stack

| | |
|--------------|-------------|
| Name* | Description |
| sysdig-cloud | |

Configuration Options

| | |
|--|---|
| Sysdig Cloud access key* <div></div> Your unique Sysdig Cloud access key - register for a Sysdig Cloud account at www.sysdig.com to receive a key. | Sysdig Cloud tags <div></div> Tags to be applied to all hosts on which the Sysdig Cloud container is deployed - these will surface in the Sysdig Cloud app. Should be a comma-separated list in the form of TAG_NAME:TAG_VALUE. For example: role:webserver,location:europe. |
| Sysdig Cloud version* <div>latest</div> Specify a version of the Sysdig Cloud container to pull (default will pull latest stable version). | Host exclude label* <div>sysdig.exclude_sysdigcloud=true</div> Specify a Rancher host label here that can be used to exclude deployment of the Sysdig Cloud container on any given host. Eg: sysdig.exclude_sysdigcloud=true (you could then add the label 'sysdig.exclude_sysdigcloud=true' to any Rancher host to exclude Sysdig Cloud from |

云的对接：



Hadoop 动态扩容的支持等：http://www.iqiyi.com/w_19rt9qkn7d.html

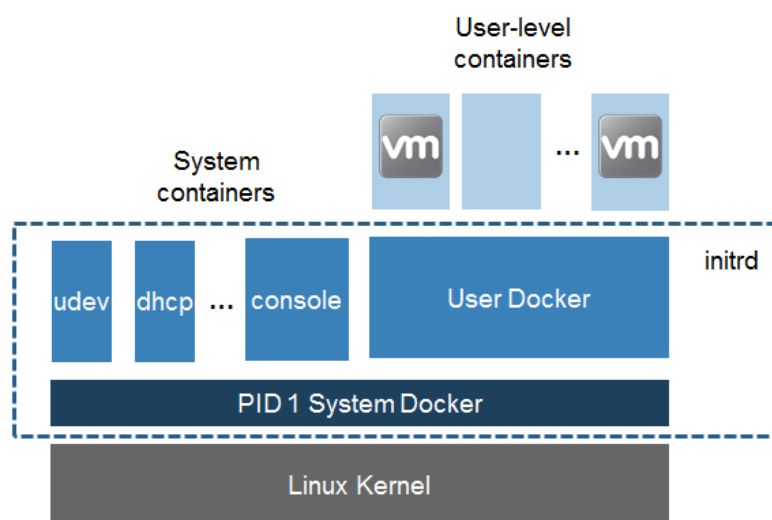
再次强调，上述能力并非是 Hadoop 产品里的，而是通过任何人都可以创建的配置文件完成的。比如：大家可以通过这种技术实现 WebLogic 应用部署或是 Zabbix 监控方案等。

CI/CD 优化部署流水线是 Docker 的拿手项目，Rancher 通过上述一键部署能力提供快速构建的支持。

上面说的主要还是容器管理平台 Rancher，我们还有一款产品是 RancherOS。它是一个只有二十几 M 的操作系统，专门运行容器的。可以看到它的所有系统进程都是在容器里运行，性能好，升级维护特别方便。更酷的是我们还支持把虚拟机（Windows 或

Linux) 跑在容器里, 这样对于还没有上 IaaS 云的企业来说, 直接上容器云也是一个不错的选择。

RancherOS 是“容器化”的 Linux



22

Rancher 还有很多其它超酷的功能, 比如用户和权限管理, 多租户管理, 界面上集成日志和 shell 访问, API 调用器等, 由于时间关系这里不多说了。有兴趣的网友请关注我们的 Blog: <http://rancher.com/blog/>

说得再多都不如大家自己上手亲自感受一下, 一条命令安装好 Rancher 的容器管理平台:

```
sudo docker run -d --restart=always -p 8080:8080 rancher/server
```

注：【Rancher 中国实战群】现已开通，对 Docker 和 Rancher 感兴趣的朋友欢迎加入讨论。加微信群方法：1.关注【云舒网络】公众号，2.留言“我要加群”；QQ 群号：216521218

● 实战篇：

1. Rancher 快速上手指南（虚拟机篇）

通过一个您已经熟悉的任何一种主流的发行版 Linux 虚拟机，就可以开始一个快速简单的 Rancher 测试体验。

建议虚拟机的规格：1vcpu，不小于 4GB 内存，一块能够连通互联网的网卡。本文编写的测试机是 AWS 虚拟机上的 Amazon Linux AMI，对 CentOS/RHEL 有直接参考意义。

1.1 Linux 主机准备

安装和运行 Docker 命令和服务，这基本是 Rancher 对于操作系统的最小化需求了。

如果您还不太熟悉 Linux 或者 Docker 可以参考以下文档：

Ubuntu 用户参考文档：<https://docs.docker.com/engine/installation/ubuntu/linux/>

CentOS/RHEL 用户参考文档：<http://www.dedoimedo.com/computers/docker-guide.html>

Docker 命令和服务安装好之后需要确认：

#确认 Docker 的版本，下面是 CentOS 的输出

```
[ec2-user@ip-172-31-30-38 ~]$ sudo docker version
```

Client:

Version: 1.9.1

API version: 1.21

Go version: go1.4.2

Git commit: a34a1d5/1.9.1

Built:

OS/Arch: linux/amd64

Server:

Version: 1.9.1

API version: 1.21

Go version: go1.4.2

Git commit: a34a1d5/1.9.1

Built:

OS/Arch: linux/amd64

#确认 Docker 服务已经启动并且运行，下面是以 CentOS 为例：

```
[ec2-user@ip-172-31-30-38 ~]$ sudo service docker status
```

```
docker (pid 7652) is running...
```

1.2 启动 Rancher 服务器

Rancher 服务器是一个 Docker Image , 所以其软件本身不需要安装 , 只需执行 Docker 命令下载并且运行 Rancher 服务器的镜像即可。

```
[ec2-user@ip-172-31-30-38 ~]$ sudo docker run -d --restart=always -p 8080:8080  
rancher/server
```

```
Unable to find image 'rancher/server:latest' locally
```

```
latest: Pulling from rancher/server
```

```
0bf056161913: Pull complete
```

```
1796d1c62d0c: Pull complete
```

```
e24428725dd6: Pull complete
```

```
89d5d8e8bafb: Pull complete
```

```
a31a85515ea3: Pull complete
```

```
c2fd2bef635f: Pull complete
```

```
cb545eb6ebd1: Pull complete
```

```
7beaeed203e7: Pull complete
```

```
f483a41462cd: Pull complete
```

```
2fd8dc138841: Pull complete
```

```
a4e1df2cafae: Pull complete
```

```
5f632b46feff: Pull complete
```

```
a4ff409fd1b0: Pull complete
```

```
8713e0a3f956: Pull complete
```

```
7f6c235d968a: Verifying Checksum
```

```
c074ec496974: Download complete
```

```
390a2453f500: Download complete
```

```
c7f9c84ef74a: Download complete
```

```
Status: Downloaded newer image for rancher/server:latest
```

```
docker.io/rancher/server: this image was pulled from a legacy registry. Important:
```

```
This registry version will not be supported in future versions of docker.
```

```
7c41a0a1a9c79842bca53c19e4ec106b0c2dc6469baec6077a40405f80b26963
```

```
[ec2-user@ip-172-31-30-38 ~]$
```

1.3 命令行参数解释

- `docker run [OPTIONS] IMAGE [COMMAND] [ARG...]` 运行一个 Docker 容器
- `-d` 在后台运行 Docker 容器，并且打印出它的容器 ID（Run container in background and print container ID）
- `--restart=always` 当容器存在时所应用的重启策略，总是重启。
- `-p 8080:65432` 容器端口在虚拟机本机上使用 8080 端口，Rancher 服务器的 UI 对外服务的端口是 8080，如果您的服务器是远程的服务器，还需要考虑到你的测试客户机和虚拟机之间的防火墙策略，确保所使用的 Rancher 服务器 UI 对外服务端口不是防火墙阻止的端口。
- `rancher/server` 这里声明让 Docker 去 Docker Hub 下载并且运行名称为 `rancher/server` 的 Docker 镜像到本地。

#检查 Docker 已经正确下载了 rancher/server 镜像到本地

```
[ec2-user@ip-172-31-30-38 ~]$ sudo docker images
```

| REPOSITORY | TAG | IMAGE ID | CREATED |
|----------------|--------|--------------|------------|
| VIRTUAL SIZE | | | |
| rancher/server | latest | 25c20134881a | 3 days ago |
| 845 MB | | | |
| <none> | <none> | 8713e0a3f956 | 4 days ago |
| 473.9 MB | | | |

```
[ec2-user@ip-172-31-30-38 ~]$
```

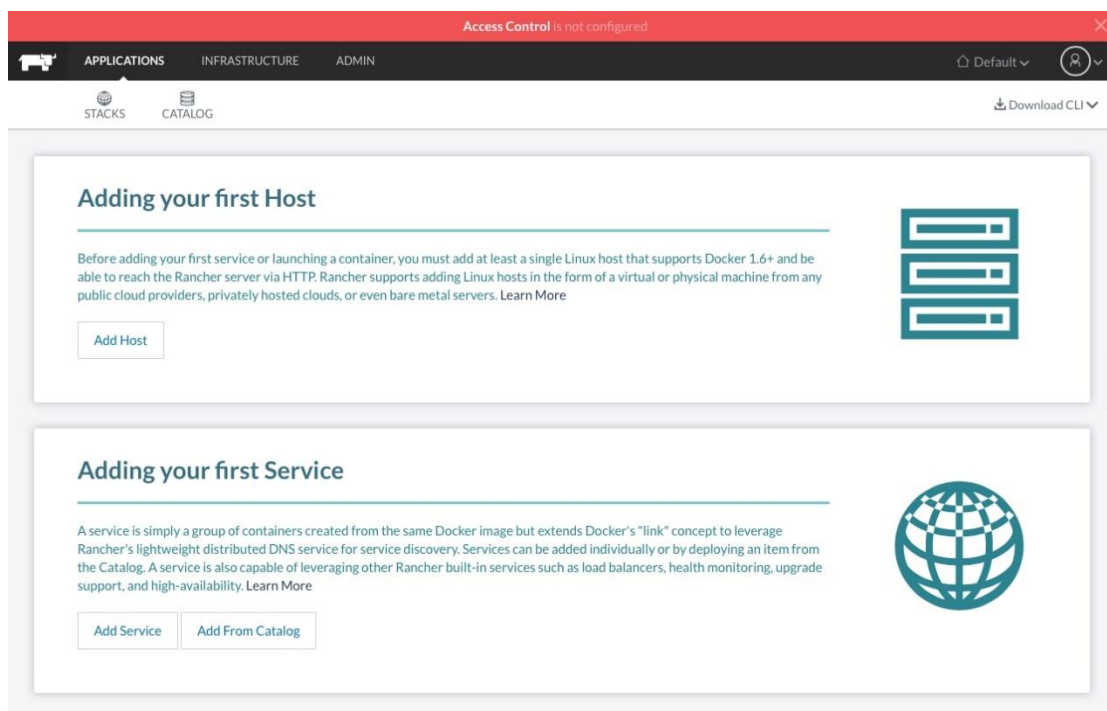
#检查 Rancher 服务器容器已经正常运行

```
[ec2-user@ip-172-31-30-38 ~]$ sudo docker ps
```

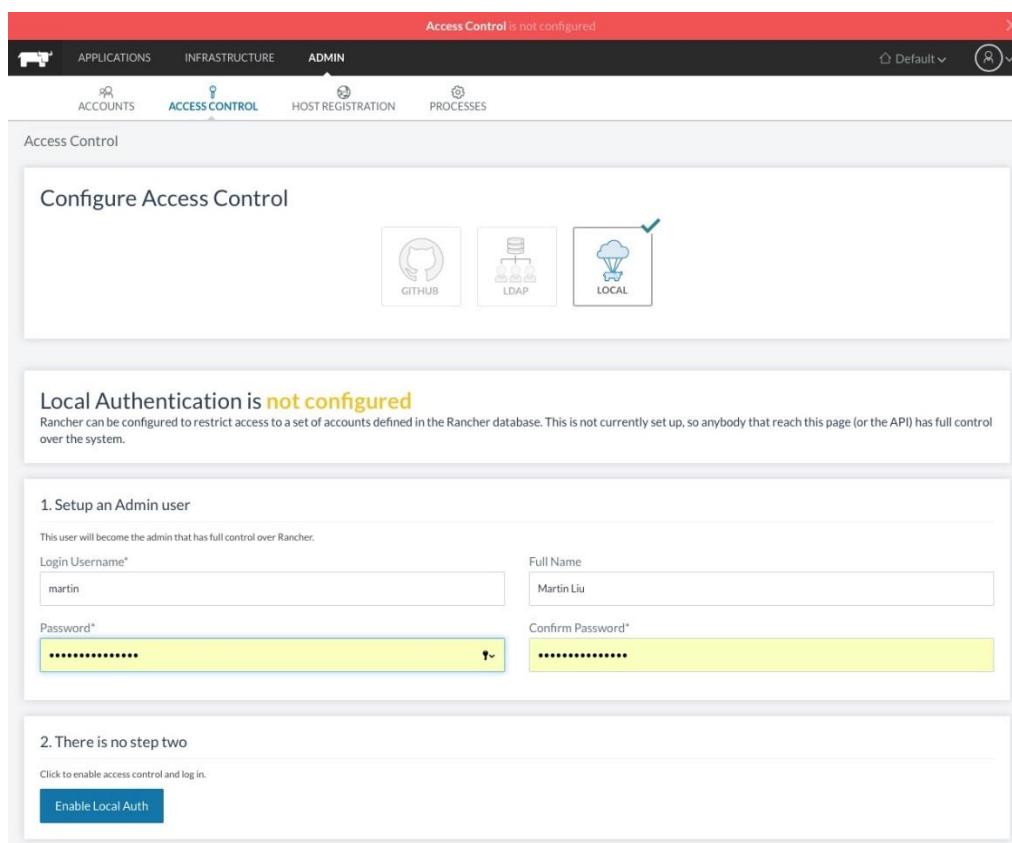
| CONTAINER ID | IMAGE | COMMAND |
|--------------|--|--------------------------------------|
| CREATED | | STATUS |
| NAMES | | PORTS |
| 7c41a0a1a9c7 | rancher/server | "/usr/bin/s6-svscan /" 3 minutes ago |
| Up 3 minutes | 3306/tcp, 8080/tcp, 0.0.0.0:8080->8080/tcp | tiny_kalam |

```
[ec2-user@ip-172-31-30-38 ~]$
```

国内的服务器的下载速度可能会比较慢，需要等待大约 30 分钟左右。用浏览器打开 Rancher 服务器 UI 界面，并且确认是否可以正常登陆。



首次访问还没有配置访问权限的页面如上图所示。为了安全起见，点击上面的 Access Control 来设置一个本地账号和密码。



如上图，使用设置的账号和密码再次登陆确认，配置的信息正确，继续下面的测试。

还可以在命令行开启 Rancher 容器运行日志监控，如下所示：

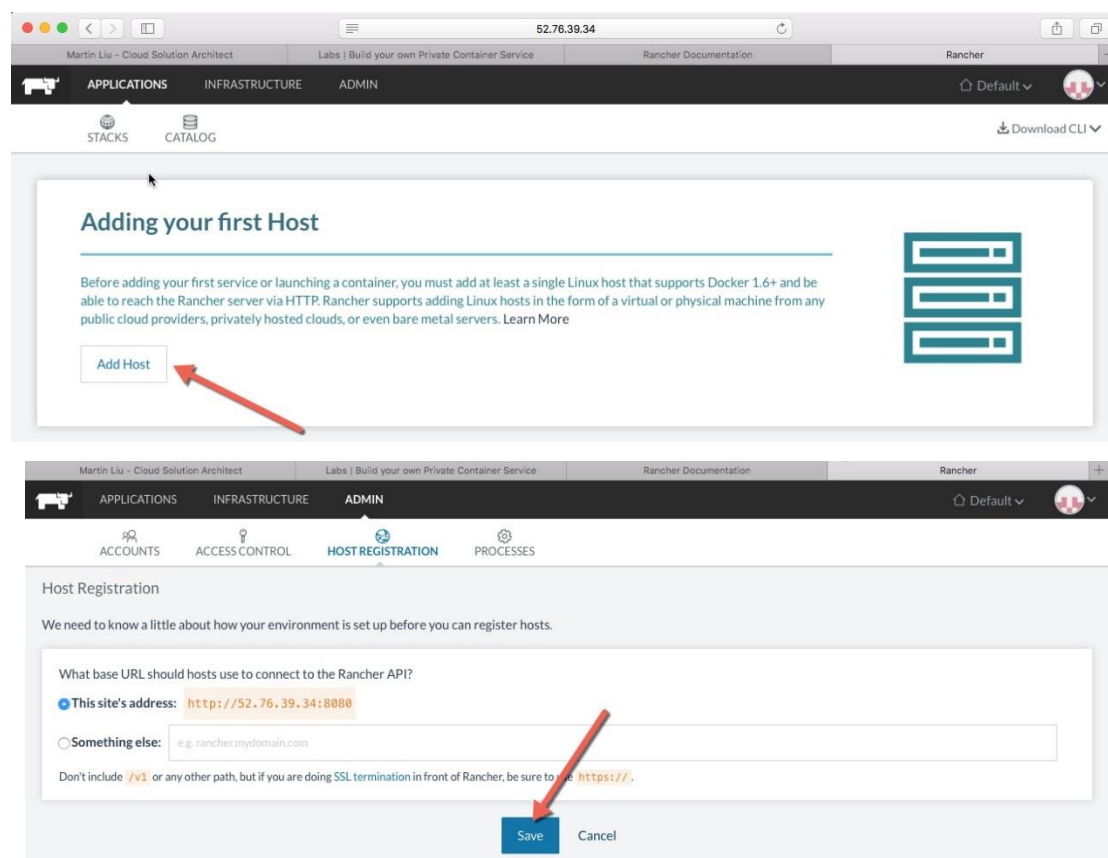
#替换下面红字部分的 docker 容器 id，你的 id 可以从 docker ps 命令查到

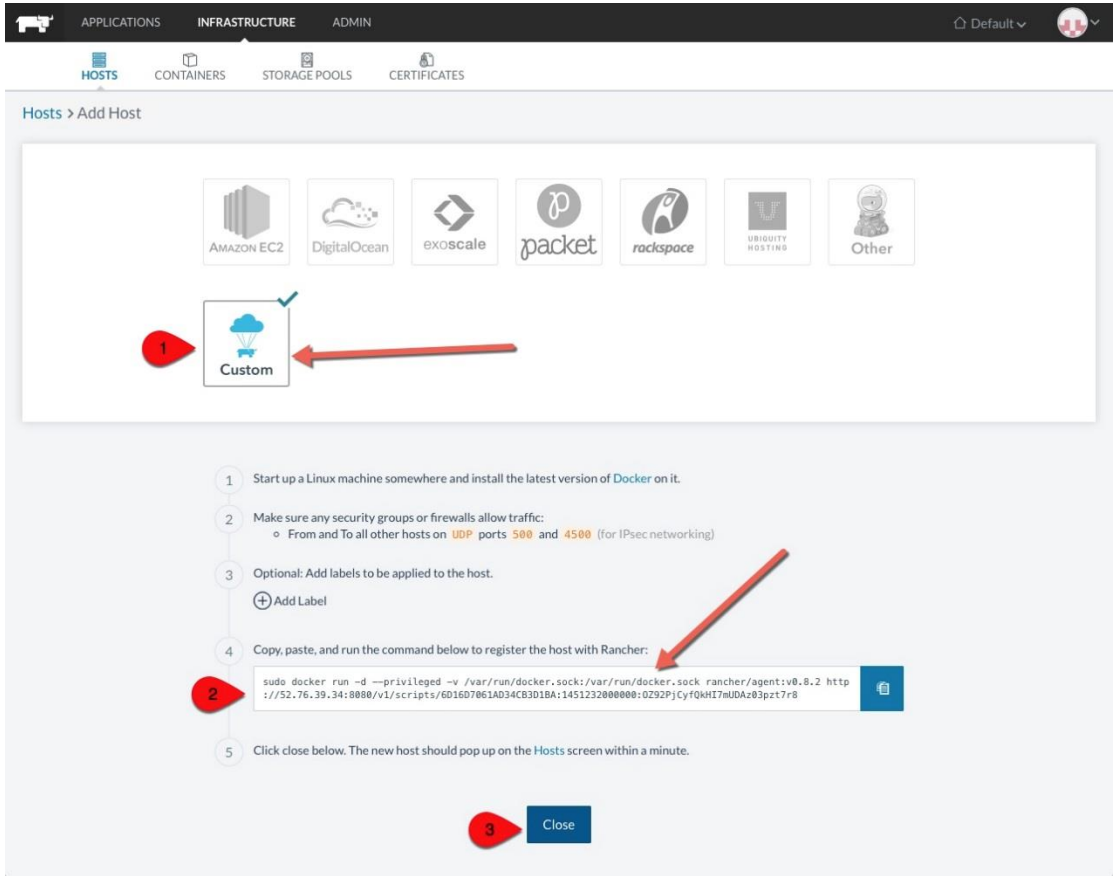
```
[ec2-user@ip-172-31-21-99 ~]$ sudo docker logs -f 988003c02bcd
```

到目前，你已经完成了 Rancher 服务器的部署和基础配置。

1.4 添加主机

主机是 Rancher 的工作节点，类似服务器虚拟化的 Hypervisor；在本实验中我们在运行 Rancher 服务器容器的管理节点上（虚拟机）做 All-in-One 的测试，因此下面把测试所用的虚拟机添加为运行工作负载容器的工作主机。点击登陆以后界面上的 Add Host 按钮。





- 1.) 点击 Customer，默认的选项是 DigitalOcean。
- 2.) 复制文本框中的代码，在虚拟机的命令里面粘贴运行。
- 3.) 点击 Close 按钮。

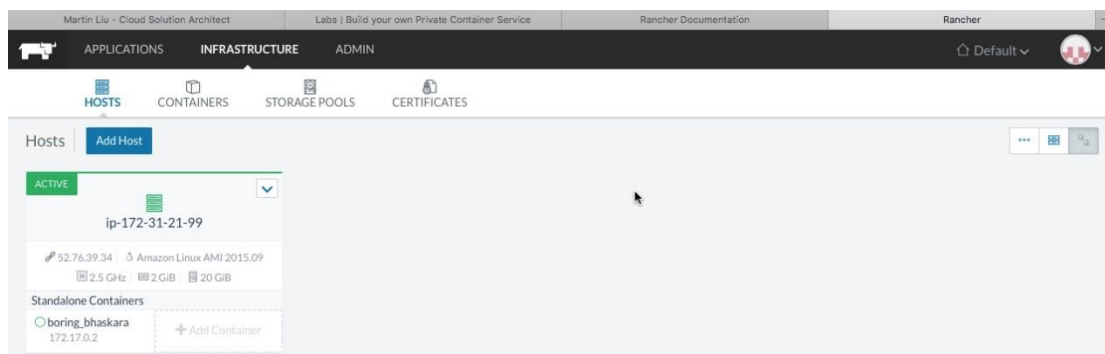
运行以上命令之后，在命令行可以用 docker ps 命令再次查看容器运行的状态，如下所示：

```
[ec2-user@ip-172-31-21-99 ~]$ sudo docker ps
```

| CONTAINER ID | | IMAGE | COMMAND |
|--------------|--------|-------|---------|
| CREATED | STATUS | | PORTS |
| NAMES | | | |

```
6ab206a64a68      rancher/agent:v0.8.2  "/run.sh run"      2 minutes
ago              Up 2 minutes          rancher-
agent
988003c02bcd      rancher/server        "/usr/bin/s6-svscan /"  28 minutes
ago              Up 28 minutes        0.0.0.0:8080->8080/tcp, 3306/tcp
boring_bhaskara
[ec2-user@ip-172-31-21-99 ~]$
```

我们可以看到多了一个名字为 rancher/agent 的容器。过几分钟之后在回到 Web 控制台，查看 Host 添加之后的结果。



如上图所示， 我们看到了一台活动的主机，并且该主机上运行着一个容器，就是 Rancher 服务器自己。

可以继续重用上面的命令，来把其他的测试虚拟机也添加为 Host，如下所示：

```
[ec2-user@ip-172-31-21-99 ~]$ sudo docker run -e CATTLE_AGENT_IP=X.X.X.X -
d --privileged -v /var/run/docker.sock:/var/run/docker.sock rancher/agent:v0.8.2
```

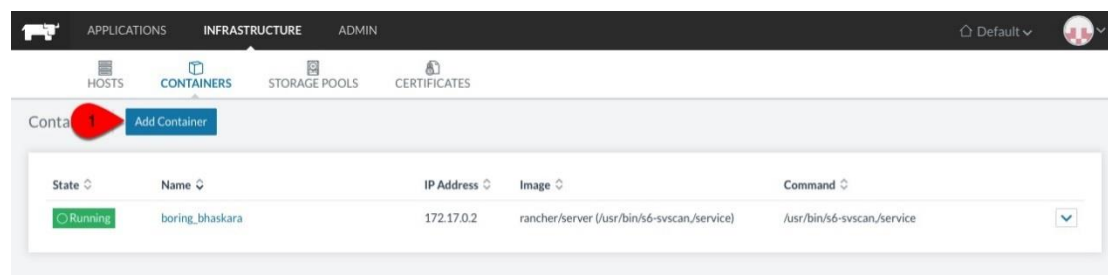
<http://X.X.X.X:8080/v1/scripts/6D16D7061AD34CB3D1BA:1451232000000:OZ92Pj>

[CyfQkHI7mUDAz03pzt7r8](#)

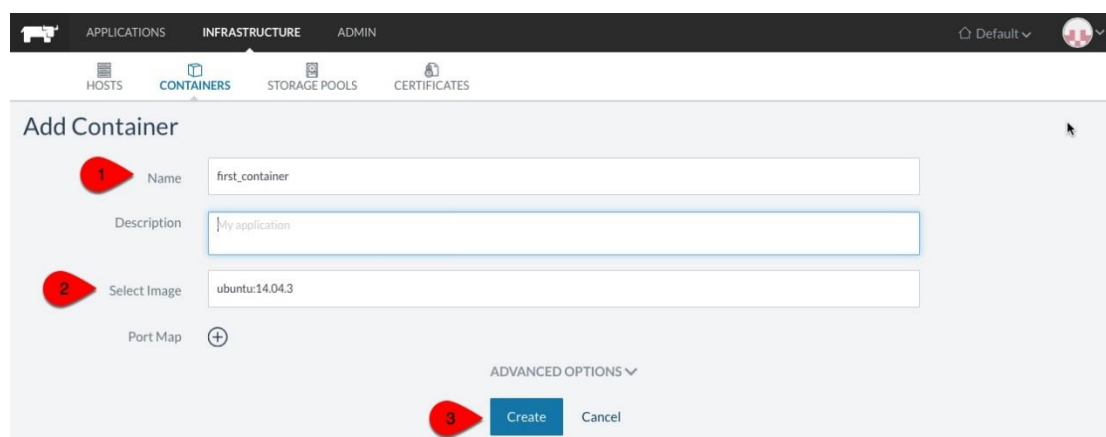
替换以上红色部分，主要是 IP 地址需要修改为运行 Rancher 服务器容器的服务器的 IP，当然，运行此条命令的前提条件如第一节所述，与 Rancher 服务器的准备工作相同。

1.5 用 Web UI 运行容器

现在来通过图形界面来运行第一个容器。通过简单地几下鼠标点击即可为完成一个容器的运行动作。



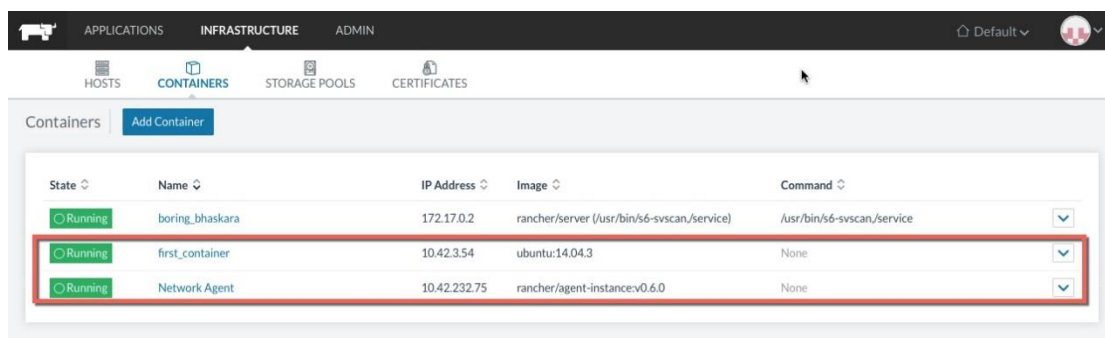
1.) 点击 Add Container 按钮增加新的容器。



1.) 输入 first_container 作为容器的名称。

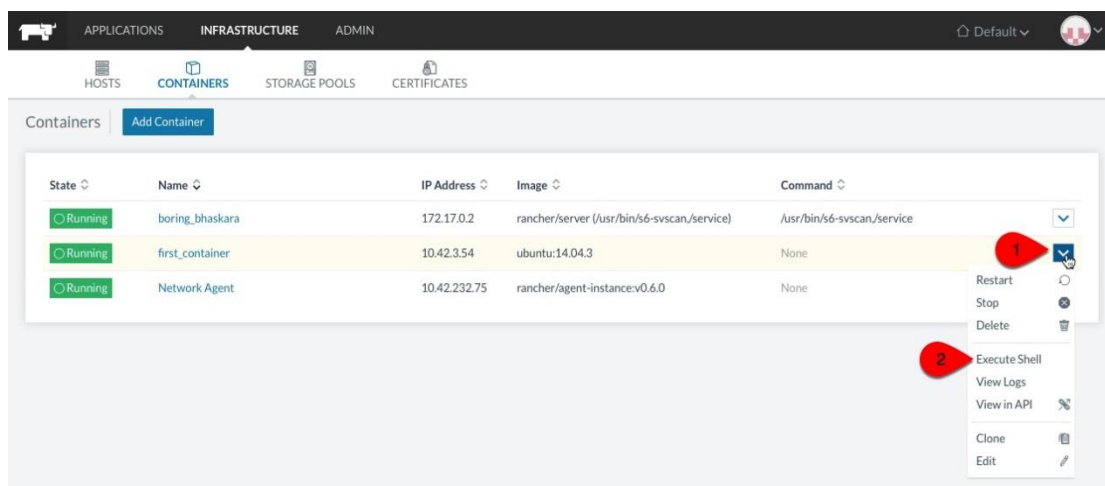
2.) 使用默认的 Ubuntu14 镜像。

3.) 点击 create 创建按钮。



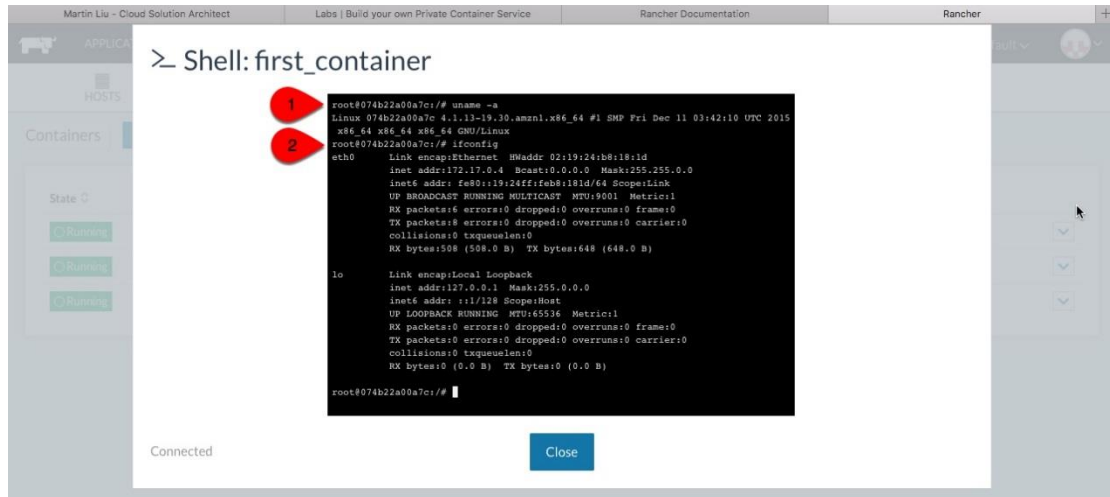
1.) 经过大约几秒钟，出现了两个运行的容器，firs_container 为本测试创建的容器，下面的那个 Network Agent 容器的作用是用户主机之间的网络通信。

2.) 上图可以看出主机间会使用 Rancher 服务器所管理的 10.42.0.0 内网通信。



1.) 在新建的 Ubuntu 容器的菜单上点击向下按钮。

2.) 选择在 Web 页面中运行 Shell 选项。



1.) 经过几秒钟的连接，网页上出现了 Ubuntu 容器的 root 命令行提示符，运行第一个命令 `uname -a`。

2.) 运行第二个命令行 `ifconfig`。

到此为止你已经成功创建了新的容器，在命令行里面，如果你是 developer 的话，你已经可以开工了，或是部署测试代码，或者打造应用服务的 image。

1.6 用命令行运行容器

如果您偏爱 CLI 命令行，可以在 host 上执行容器的创建和管理动作，如下所示：

#下面命令为运行一个新的名为 `second_container` 的 Ubuntu14.04.2 容器，并在容器运行之后，直接进入该容器的命令行

```
[ec2-user@ip-172-31-21-99 ~]$ docker run -it --name=second_container
```

```
ubuntu:14.04.2
```

```
Unable to find image 'ubuntu:14.04.2' locally
```

```
14.04.2: Pulling from library/ubuntu
```

```
Digest:
```

```
sha256:a1cec70421f71f00c8bdb0adf0226dc548ff5ba9699cbd5fa09acdb68df82a02
```

```
Status: Downloaded newer image for ubuntu:14.04.2
```

```
root@be607f589023:/#
```

```
#这里是第二个容器的命令行，下面执行 id 命令
```

```
root@be607f589023:/# id
```

```
uid=0(root) gid=0(root) groups=0(root)
```

```
#在该容器里做网络测试
```

```
root@be607f589023:/# ping rancher.com
```

```
PING rancher.com (104.24.18.49) 56(84) bytes of data.
```

```
64 bytes from 104.24.18.49: icmp_seq=1 ttl=59 time=1.75 ms
```

```
64 bytes from 104.24.18.49: icmp_seq=2 ttl=59 time=1.97 ms
```

```
^C
```

```
--- rancher.com ping statistics ---
```

```
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
```

```
rtt min/avg/max/mdev = 1.758/1.866/1.975/0.116 ms
```

```
#退出该容器命令行
```

```
root@be607f589023:/# exit
```

```
exit
```

#在 host 执行 docker 容器运行状态查看命令

```
[ec2-user@ip-172-31-21-99 ~]$ sudo docker ps
```

| CONTAINER ID | IMAGE | COMMAND |
|--------------------------------------|--------------------------------------|----------------------------------|
| CREATED | STATUS | PORTS |
| NAMES | | |
| 074b22a00a7c | ubuntu:14.04.3 | "/bin/bash" |
| 15 minutes ago | Up 15 minutes | |
| 37695985-73e7-4e54-be90-870c86d4cef3 | | |
| 9132d138e896 | rancher/agent-instance:v0.6.0 | "/etc/init.d/agent-in" 15 |
| minutes ago | Up 15 minutes | 0.0.0.0:500->500/udp, |
| 0.0.0.0:4500->4500/udp | ce65d23e-2f55-4497-9ffe-38ae1bcedf9f | |
| 6ab206a64a68 | rancher/agent:v0.8.2 | "/run.sh run" |
| 39 minutes ago | Up 39 minutes | |
| rancher-agent | | |
| 988003c02bcd | rancher/server | "/usr/bin/s6-svscan /" |
| About an hour ago | Up About an hour | 0.0.0.0:8080->8080/tcp, 3306/tcp |
| boring_bhaskara | | |
| [ec2-user@ip-172-31-21-99 ~]\$ | | |

以上测试结束后，可以在 Web UI 中查看该容器的状态，可以看到它已经处于 stop 的状态，它所使用网络地址与 Rancher 服务器容器同一个网段，这是默认执行 docker run 命令的默认结果；还可以在命令行制定使用 Rancher 服务器所管理的叠加的内部私有网络，

命令行如下：

#注意一下 docker run 命令使用了 `--label io.rancher.container.network=true` 参数

```
[ec2-user@ip-172-31-21-99 ~]$ docker run -it --label  
io.rancher.container.network=true ubuntu:14.04.2
```

#进入新容器的命令之后查看网络地址

```
root@a12455b64a7b:/# ifconfig
```

```
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:05
```

```
          inet addr:172.17.0.5  Bcast:0.0.0.0  Mask:255.255.0.0
```

```
          inet6 addr: fe80::42:acff:fe11:5/64 Scope:Link
```

```
          UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
```

```
          RX packets:5 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
```

```
          collisions:0 txqueuelen:0
```

```
          RX bytes:438 (438.0 B)  TX bytes:508 (508.0 B)
```

```
lo        Link encap:Local Loopback
```

```
          inet addr:127.0.0.1  Mask:255.0.0.0
```

```
          inet6 addr: ::1/128 Scope:Host
```

```
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
```

```
collisions:0 txqueuelen:0
```

```
RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

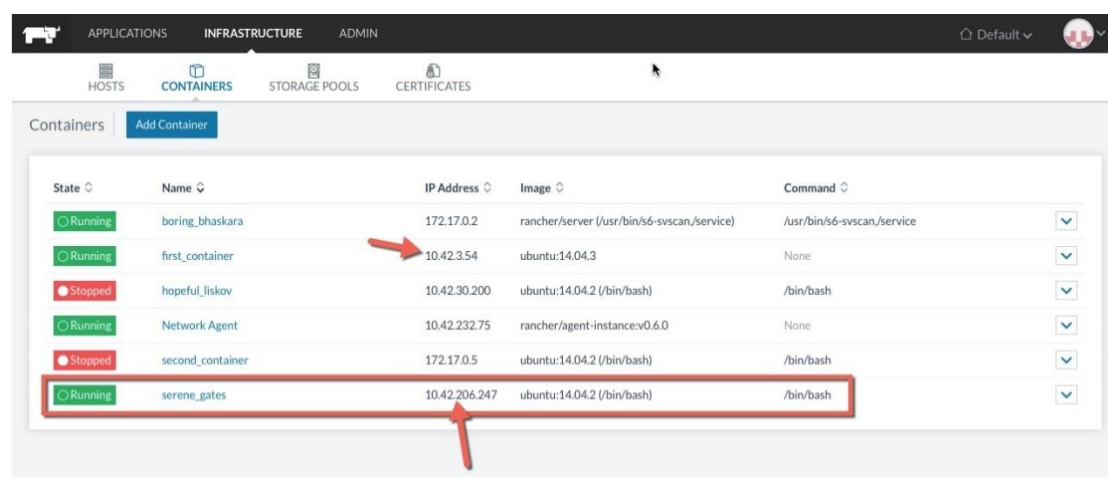
```
root@a12455b64a7b:/# ping rancher.com
```

```
PING rancher.com (104.24.18.49) 56(84) bytes of data.
```

```
64 bytes from 104.24.18.49: icmp_seq=1 ttl=59 time=1.75 ms
```

```
64 bytes from 104.24.18.49: icmp_seq=2 ttl=59 time=1.95 ms
```

在不退出以上容器命令行的情况下，进入 Web UI 查看此容器的状态。



| State | Name | IP Address | Image | Command |
|---------|------------------|---------------|---|----------------------------|
| Running | boring_bhaskara | 172.17.0.2 | rancher/server (/usr/bin/s6-svscan/service) | /usr/bin/s6-svscan/service |
| Running | first_container | 10.42.3.54 | ubuntu:14.04.3 | None |
| Stopped | hopeful_jiskov | 10.42.30.200 | ubuntu:14.04.2 (/bin/bash) | /bin/bash |
| Running | Network Agent | 10.42.232.75 | rancher/agent-instance:v0.6.0 | None |
| Stopped | second_container | 172.17.0.5 | ubuntu:14.04.2 (/bin/bash) | /bin/bash |
| Running | serene_gates | 10.42.206.247 | ubuntu:14.04.2 (/bin/bash) | /bin/bash |

最下面一个是使用了内部叠加网的容器，它的 Ip 地址从 Web UI 中看和第一个 first_container 的 IP 为同一个网段。

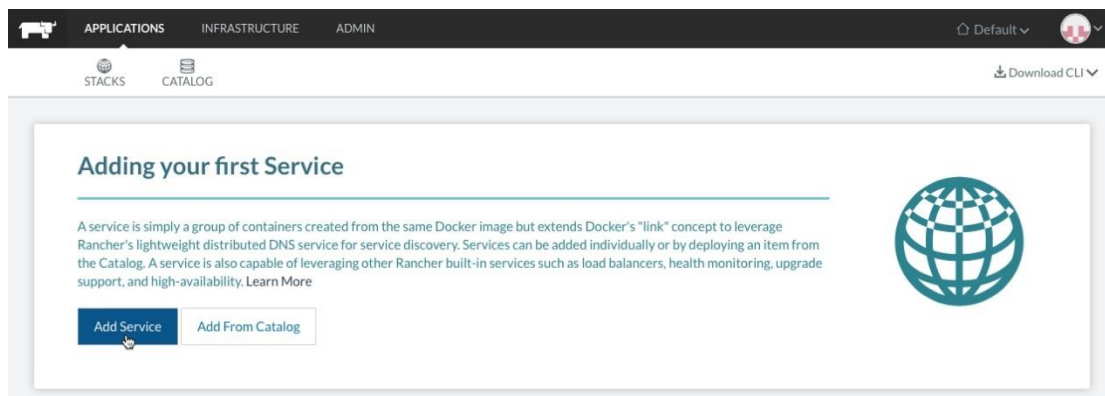
至此搞清楚了多个容器的运行和基本的网络概念之后，下面可以做更为复杂的测试。

1.7 用 Web UI 完成多层应用架构的部署

本测试需要完成 WordPress 应用的多层部署，期望的部署模式如下：

- 1.) 前端负载均衡服务，一个 LB 用来接入来自互联网的流量。
- 2.) WordPress 应用层，有两个运行 WordPress 软件的容器组成。
- 3.) 数据库服务层，有一个 MySQL 服务器容器。

具体的操作步骤如下所示：



在登录后的首页，点击 Add Service 按钮开始创建整个堆栈。

The screenshot shows the 'Add Service' form in the Cloudsoar interface. The form is divided into several sections. At the top, there are tabs for 'APPLICATIONS', 'INFRASTRUCTURE', and 'ADMIN'. Below these are 'STACKS' and 'CATALOG' tabs. The main form area is titled 'Add Service'. It has a 'Scale' section with two radio buttons: 'Run 1 container' (selected) and 'Always run one instance of this container on every host'. Below this is a 'Name' field with the value 'database' (annotated with a red circle 1). There is a 'Description' field with the value 'My application'. A 'Select Image' dropdown menu shows 'mysql' (annotated with a red circle 2). Below this is a 'Port Map' section with a plus icon. A 'Service Links' section says 'There are no other services to link to'. An 'ADVANCED OPTIONS' section is expanded (annotated with a red circle 3), showing tabs for 'Command', 'Volumes', 'Networking', 'Health Check', 'Security/Host', 'Labels', and 'Scheduling'. The 'Command' tab is active, showing a 'Command' field with 'e.g. /usr/bin/httpd -f httpd.conf', an 'Entry Point' field with 'e.g. /bin/sh -c', a 'user' field with 'e.g. apache', and a 'Console' section with three radio buttons: 'Interactive & TTY (-i -t)' (selected), 'TTY (-t)', and 'None'. Below this is an 'Environment Vars' section (annotated with a red circle 4) with a plus icon. It shows a table with 'Name' and 'Value' columns. The first row has 'MYSQL_ROOT_PASSWORD' (annotated with a red circle 5) and 'pass1'. At the bottom, there are 'Create' and 'Cancel' buttons (annotated with a red circle 6).

1.) 选择 Add Service 增加 数据库服务，输入名称 database。

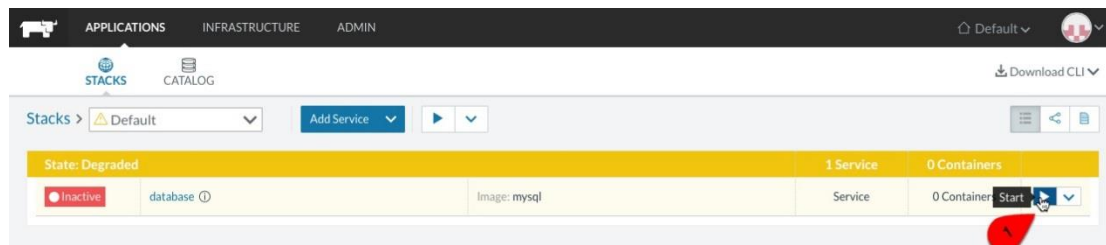
2.) 选择使用 mysql 镜像。

3.) 点击高级选项卡。

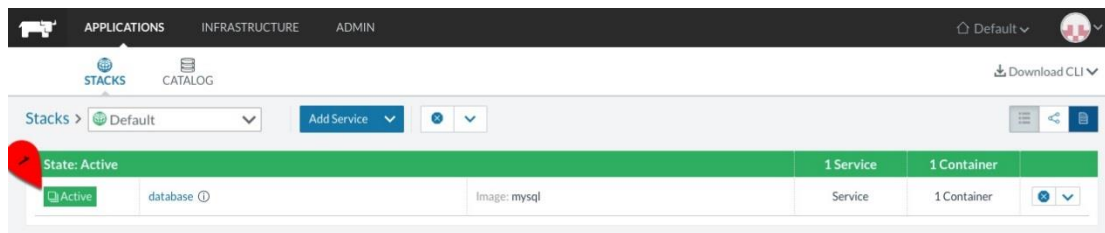
4.) 点击添加环境变量。

5.) 数据环境变量的内容，mysql 数据库的 root 密码为 pass1。

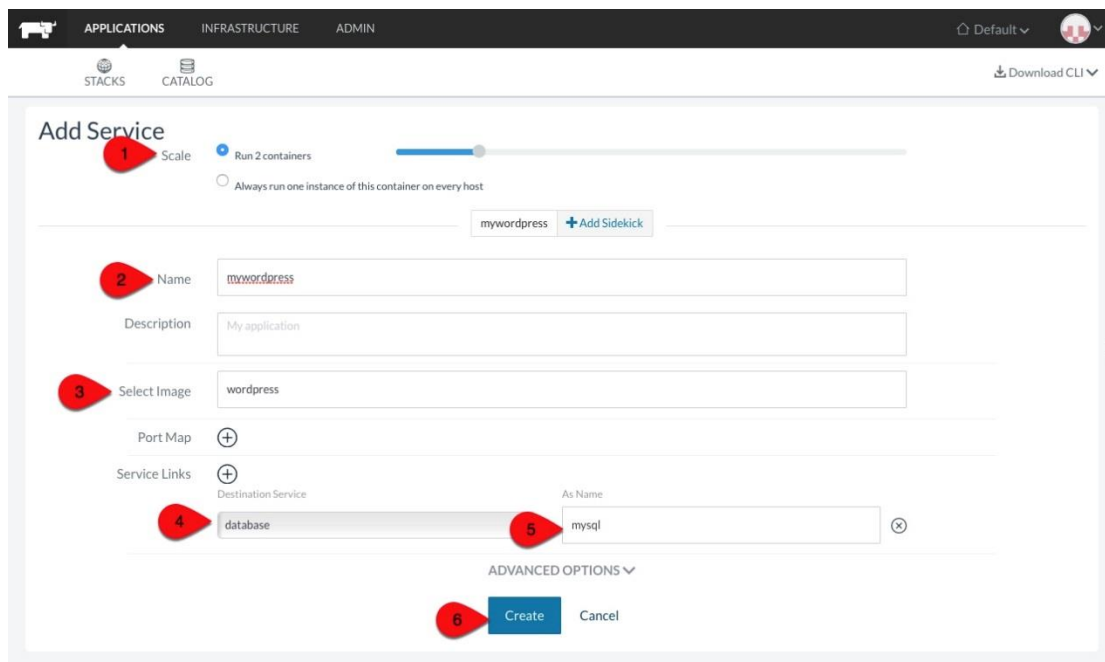
6.) 点击创建此服务。



1.) 创建后的服务默认为非活动状态，点击 Start 按钮，启动数据库服务。



1.) 启动之后的 mysql 数据库服务状态正常，点击上面的 Add Service 按钮添加服务。



1.) 拖动圆点，使本层服务的容器数量为 2。

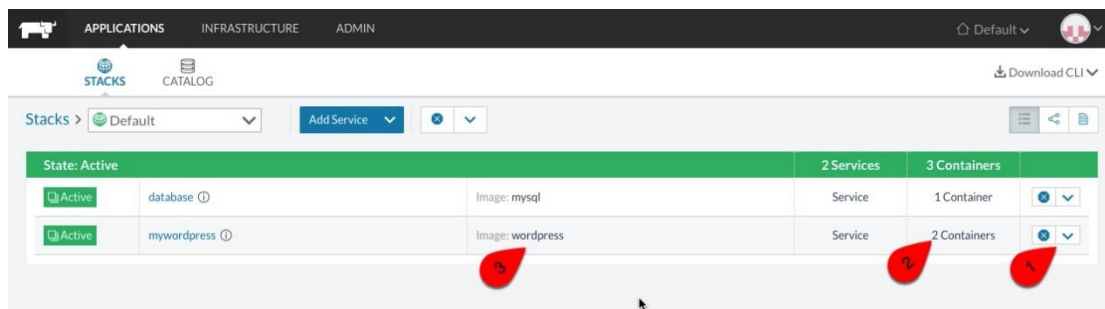
2.) 输入名称为 mywordpress。

3.) 输入所需要使用的 Wordpress 镜像。

4.) 选择它所依赖的数据库服务。

5.) 输入名称 mysql。

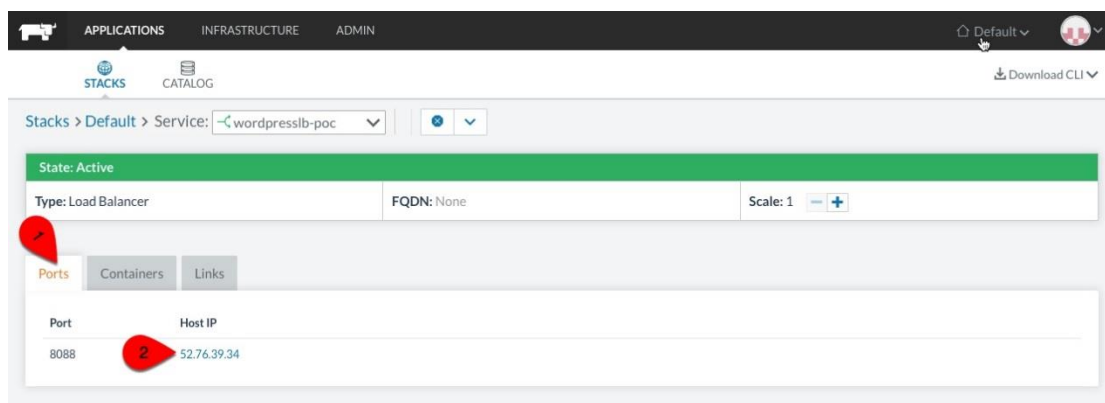
6.) 点击创建按钮。



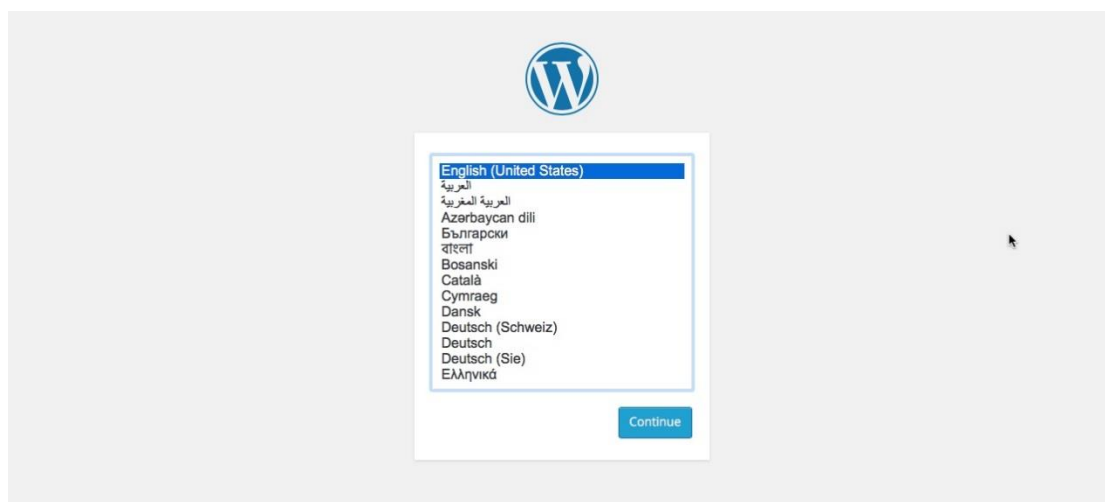
- 1.) 点击菜单中的运行按钮，启动新建容器。
- 2.) 观察容器的数量为 2。
- 3.) 观察容器所使用的镜像名称，点击 Add Service 下拉菜单，选择创建负载均衡器。

The screenshot shows the 'Add Load Balancer' configuration page. It includes a 'Scale' section with a slider set to 1. The 'Name' field is filled with 'wordpresslb-pool'. The 'Listening Ports' section contains a table with one row: Source Port 8088, Protocol http, SSL checked, Default Target Port 80, and Access Public. The 'Targets' section has a dropdown menu with 'mywordpress' selected. At the bottom, there are tabs for 'SSL Termination', 'Stickiness', 'Labels', and 'Scheduling'. A 'Save' button is at the bottom right.

- 1.) 使用默认 1 为 LB 的数量。
- 2.) 输入 LB 的名称。
- 3.) 输入在 Host 上 LB 对外服务的端口为 8088。
- 4.) 输入 Wordpress 容器的服务端口 80。
- 5.) 选择对象服务，为 myWordpress。
- 6.) 点击 Save 创建此容器和相关配置。



- 1.) 运行创建好的 LB 容器，点击该容器，查看它的状态，点击 Ports 选项卡。
- 2.) 点击 Host IP，浏览器就会连接到 http://your_host_ip:port 打开负载均衡的服务网址。



如上图所示 wordpress 的安装页面正常打开，可以继续完成 WordPress 的安装和配置。至此您已经顺利完成了多层应用的部署和搭建。

本篇参考了 Rancher 官方文档：<http://docs.rancher.com/rancher/quick-start-guide/>

但是不包含 docker-compse 命令行工具 + yaml 配置文件的创建方式，建议可以参考该文档完成完整的测试。

本测试把 Rancher 的基础用法做了一个初级的尝试，希望对新手有所帮助。

2. 基于 Docker 的构建流程 - 持续集成及测试

之前我们发布过一篇这个系列的文章，它包括了代码构建，测试，打包，持续集成及部署，以及在生产环境中管理应用 stack 等。我们应该明确的是，一个好的构建系统需要达到如下要求：

- 1.) 可重复性——能在不同的开发机器和自动构建服务器中，生成/创建有一致依赖关系的构建环境。
- 2.) 集中化管理——所有开发机器和构建服务器的构建环境，是来自于同一个代码仓库中心或服务器，且环境设置能及时更新。
- 3.) 隔离性——项目的各个子模块相互隔离，而不是相互依赖。
- 4.) 并行性——并行构建子模块，提高构建效率。

虽然用依赖管理工具 (Maven、 Pip、 Rake)，配置管理工具(Puppet、 Chef) 和虚拟化工具 (Vagrant)，也能建立一个稳定、集中管理的构建系统，但是这也有各种问题。不是所有的项目都需要所有这些工具，但是任何长时间运行的大项目都需要自动化到这个程度。好了，现在该我们 Docker 出场了；无需投入大量时间和资源，Docker 和其生态系统能帮助我们支持上述工具。我们来看看为应用创建集中化构建环境的步骤：

2.1.构建系统扩展带来的挑战

2.2 解决方案和最佳实现

2.3.利用 Docker 进行系统构建

- 集中化构建环境
- 用 Docker 打包应用
- 用 Docker compose 生成构建环境

2.4.通过 Docker 和 Jenkins 进行持续集成 (CI)

- 分支模型
- 用 Jenkins 创建 CI 流程
 - 构建 go-auth 服务
 - 打包 Go Auth
 - 运行集成测试

原文包含了完整的代码及相关说明,由于篇幅有限这里我们就不再分别展开,对其中细节感兴趣的的朋友可以自行去看:

基于 Docker 的构建流程 - 持续集成及测试:

<http://www.cloudsoar.com/about/blog/1/Rancher/v1.1/>

● 番外篇:

OK, 关于 Docker 企业级架构篇这里我们先告一个段落。下面是番外,关于企业数据迁移的。开篇我们说到过:**企业关心如何将应用在不同操作系统间实现无缝迁移?**我们来看看

这篇关于 Docker 和存储的文章：

Docker 容器采用两种不同的数据存储模式，一种是对 Root Image 采用分层文件系统的方式，一种是对应用数据采用了 Volume 接口。

1. Docker 容器和存储系统之 Root Image 分层文件系统

要想在生产环境中部署 Docker，开发运维人员必须还要问自己关于数据存储的几个问题：

- 1.) Docker 的数据存储组织形式是什么？
- 2.) Docker 目前存储生态圈各个公司的技术特点？
- 3.) Docker 的出现对未来存储后端要求会带来哪些技术变革？

关于 Docker 存储的介绍目前国内比较少，作为一名在存储和云计算领域扎根多年的 IT 人，我希望能够亲自研究这些课题，抛砖引玉，分享讨论。其中的观点不一定都是正确的，但是希望能够引起更多人的思考和共鸣。

1.1 Docker 的整体数据存储方式

要说明这个问题，我们可先看看容器和 VM 的区别，看看他们对存储有哪些不一样的要求：

容器和 VM，几乎所有人都会说容器就是一种比 VM 更轻更优的虚拟化技术。其实，我的理解不太一样，我认为容器和 VM 最大的不同在于容器不是 VM。这个，不是废话么。仔细分析一下，容器的重点其实不在虚拟化技术，它叫 Container，并没有 Virtual 这个词。它的关注核心已经从冷冰冰的 Machine，上升到了如何更好的去承载应用。这个是有本质区别的。VM 关注的是如何让 Machine 更加高效，VM 发展的再厉害，它也是一种更高级的 Machine。它看待后端存储其实是和一个物理 Machine 看待后端存储是一致的。而容器呢，它希望后端存储对它来说是透明的，不用它关心的，它更加关心的是应用数据的组织形式。

下面这个表是我做的一些对比

| | 运行在 | 关注 | 涉及的产品 |
|---------------------|--|--------------------|--|
| VM | Root Image 运行于块设备存储上 Image 存储在对象存储上 文件存储不在 VM 关注的层次上 | 性能 容量 可拓展 | VSAN, Ceph, ScaleIO , 存储接口 rbd, cinder 等 |
| Docker Container | Root Image 运行于分层文件系统上 Image 存储在对象存储上 Volume 是数据存储的接口 | 文件分层 迁移 应用感知 | AUFS, DeviceMapper, ZFS 存储接口 Rancher Convoy, Flocker |

云舒网络

VM 相关的存储技术已经有很多介绍了。这里我们继续把和容器相关的存储知识简单展开的讨论一下。

容器的 Root Image 存储就是实现分层的文件组织和写时复制 CoW，如 AUFS、

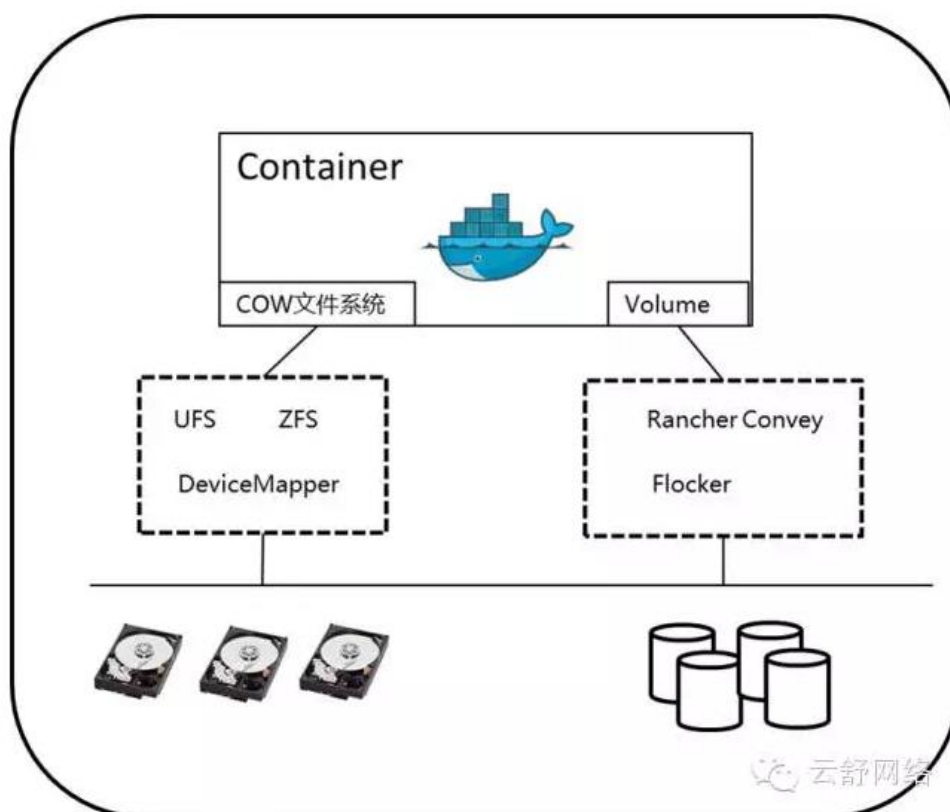
Device Mapper、ZFS。这些技术满足了容器的核心价值，即极快的创建速度，极小的存储资源消耗以及容器迁移的便捷性。他们又分成了三类：

AUFS、Overlay：联合文件系统。

DeviceMapper：CoW 块存储。

ZFS、btrfs: CoW 文件系统。

Docker 并不推荐采用 Root Image 的存储方式来存储应用数据。因为应用数据对安全、可用性、共享、性能等方面的要求和 Root Image 的要求是完全不一样的。后面我们还会专门介绍。Docker 采用了 Volume 这样一个独立的数据访问接口，应用通过 Volume 去访问相关的数据，Volume 的实现和 CoW 的分层文件系统完全独立。Volume 通过 Rancher Convoy 或者 Flocker 这样的存储驱动去管理和访问具体的存储设备。



2. Docker 容器和存储系统之 Volume 存储接口

为什么我们需要 Volume 这样的数据接口？首先我们要深刻理解的是：Docker 容器是承载应用的，是对应用环境的抽象而不是对 OS 运行环境的抽象。

Docker 容器天生设计就是为了应用的运行环境打包、启动、迁移、弹性拓展，所以 Docker 容器一个最重要的特性就是 disposable，是可以被丢弃处理，稍瞬即逝的。而应用访问的重要数据可不是 disposable 的，这些重要数据需要持久化的存储保持。Docker 提出了 Volume 数据卷的概念就是来应对数据持久化的。如果把容器比喻成一个人，那么这个人的重要数据（物质上的）就是他的财产（钱）。容器可以不存在了，但是数据必须还要存在。小沈阳说，人生最大的痛苦就是人没了，钱还在。容器会说，正相反，人生最大的幸福就是我不在了，数据还在。所以，定义好需要持久化的数据，采用 Volume 接口来存储访问是容器应用需要考虑的首要课题，必须引起高度的重视。我们可以想象一下容器和应用之间这样一段对话：

容器：我稍瞬即逝，我稍瞬即逝，重要的事情说三遍，我稍瞬即逝。

应用：哦，这样啊，那我的重要数据不能丢怎么办？

容器：请用 Volume 数据卷，请用 Volume 数据卷，重要的事情说三遍，请用 Volume 数据卷。

应用：&%#!，知道了，真啰嗦，你的前世一定是一台复读机，妈妈再也不用担心你的学习了。

那么，既然容器的分层文件系统是为了容器稍瞬即逝，弹性迁移所设计的，Volume 接口的实现就肯定和它很不一样了。具体从技术上来说，Volume 接口绕过了 Disposable 的分层文件系统，而是采用直接 Mount 挂载的方式。Volume 目录挂载后，对应用来说是透明的，应用不需要任何改变，应用按照原来的方式访问 Volume 目录就能实现重要数据的持久化。这有点像你在 Win7 下创建一个百度网盘目录一样，你向网盘目录拷贝文件的方式和向其他目录拷贝文件没有区别，但是如果你的笔记本丢了，其他目录的文件就丢了，但是网盘上的文件始终存在。这就是所谓的数据持久化。

2.1 数据迁移

数据能够持久化以后，应用容器迁移和数据共享就成为了可能。Volume 接口可以说很大程度上讲是容器迁移和数据共享的基础组件。我们先谈谈迁移：假设一个人需要从一个城市搬到另外一个城市，如果让他把所有的现金都打包带到身上是不太现实的，也非常不安全。怎么办呢？很简单，他去银行开个账户，把钱存进去，到另外一个城市的分行取就可以了。

容器迁移的道理是一样的，因为数据都是存储在 Volume 卷（银行账户）里的，所以容器在集群的另外一个服务器甚至云端重新启动的时候，只要挂载同样的数据卷就可以了。当然，这些都需要数据卷后端有共享存储，或者数据副本的支撑。

容器应用迁移的核心其实是数据卷 Volume 迁移，（注：容器本身的迁移由镜像库 Docker Registry 主导）这一部分涉及数据存储、安全加密、网络传输、性能优化、快照备

份等等的技术点，是容器管理的核心功能。Volume 数据卷更是容器间共享数据的基础，道理很简单，数据是存储在容器之外的，那么容器间共享同样一个数据卷就能共享数据。为此，Docker 容器还专门推出了数据卷容器这种特殊的容器，只要一个数据卷容器来挂载 Volume，其他需要共享 Volume 的容器只需要很简单的指明和这个数据卷容器共享 Volume 就可以了。有兴趣的读者可以自己查找数据卷容器的资料。

为了更好的支持容器迁移和数据共享，Docker 推出了 Volume Plugin 接口机制，让第三方的存储厂商来支持 Docker Volume 并且在此基础上进行功能拓展。下面这个表提供了 Volume plugin 的接口规范：

| | | |
|---------|--------------------|--------------|
| Create | 创建一个 Volume | 创建一个银行账户 |
| Remove | 删除一个 Volume | 注销一个银行账户 |
| Mount | 挂载一个 Volume，应用可以使用 | 开通 ATM 和各种支付 |
| Unmount | 卸载一个 Volume | 暂停 ATM 和各种支付 |
| Path | 返回主机目录信息 | 查询账户信息 |

可以看到 Volume Plugin 的接口规范是相当简洁的。大部分的存储和高级功能由和 Volume plugin driver 驱动的后端存储提供。以下说明就是前一期提到的 Volume plugin driver，我们以后会重点介绍。

2.2 Rancher Convoy

Convoy 是 Rancher Labs 用 Go 开发的支持 DeviceMapper、NFS、EBS、Glusterfs

多种后端存储的 Docker Volume plugin driver。Convoy 还提供了一个存储拓展功能（如快照、备份恢复等）的接口框架。

2.3 Flocker

Flocker Volume Plugin driver 主要用于多主机环境 Docker 数据卷的迁移，从而支持数据库应用等 stateful 有状态应用的主机间迁移。

关于 Volume 今天我就介绍到这里。又到了总结的时间了，前两天做了个梦，梦见自己参加了“迎接一带一路，全国 Docker 知识大赛”18 岁以下组别的决赛。里面一个题目是这样的：

Q：

请以 Volume 的字母为首字母，分别组成一个句子来说明 Volume 的特性。好有挑战啊，不过有了前面对 Volume 概念的理解，我很快就写出了答案。

A：

V -- Very important data. （存储很重要的数据）

O -- Opt to be shared. （多用于容器数据共享）

L -- Like a bank to store money. （像银行存储资金）

U -- User defined plugin driver. （用户定义的插件驱动）

M -- Move around is not easy. （迁移非常有挑战）

E -- Essential to clustered environment. （集群多主机应用的关键）

我梦见了我拿到了第一名，受到了 Docker 创始人 Solomon Hykes 的亲切接见。。。继续做梦吧，不要醒来。

● 结语：

感谢各位前辈导师的观点和经典之作，本文收录了有关 Docker 在企业级架构中从落地到实战的精彩篇章，希望对想把 Docker 这一技术落地到企业中的朋友有所帮助，有关原文阅读可下载可点击相关链接。

本文电子书下载：

网页下载：<http://www.cloudsoar.com/down/ddoc/v1.1/>

百度云盘下载：<http://pan.baidu.com/s/1jHfP9Dc>

温馨提示：

云舒网络携手 Rancher Labs 推出【[Rancher | 实战微信群](#)】，在线为您分享 Docker 技术干货，更有往期回顾精选期刊等你拿！

本群汇集了 Rancher 中国最强技术精英团队及业内技术派高人，宗旨是为了大家拥有更专业的平台交流 Rancher 实战技术，实时与 Rancher 创始团队面对面！同时欢迎各位分享自己的经验、疑难问题，我们将定期邀请分享嘉宾做各类话题分享及回顾，共同实践研究 Docker 容器生态圈。

对 Rancher 和 Docker 技术感兴趣、或对本文中细节需继续探讨的朋友，欢迎加入本群参与讨论！

加微信群方法：

- 1.关注【云舒网络】公众号
- 2.留言“我要加群”

QQ 群号：216521218

