Docker

简介

为什么出现?

Docker的出现使得Docker得以打破过去「程序即应用」的观念。透过镜像(images)将作业系统核心除外，运作应用程式所需要的系统环境，由下而上打包，达到应用程式跨平台间的无缝接轨运作。

Docker的理念

Docker是基于Go语言实现的云开源项目。

Docker的主要目标是“Build，Ship and Run Any App,Anywhere”，也就是通过对应用组件的封装、分发、部署、运行等生命周期的管理，使用户的APP（可以是一个WEB应用或数据库应用等等）及其运行环境能够做到“一次镜像，处处运行”。

Linux容器技术的出现就解决了这样一个问题，而 Docker 就是在它的基础上发展过来的。将应用打成镜像，通过镜像成为运行在Docker容器上面的实例，而 Docker容器在任何操作系统上都是一致的，这就实现了跨平台、跨服务器。只需要一次配置好环境，换到别的机子上就可以一键部署好，大大简化了操作。

解决了运行环境和配置问题的软件容器， 方便做持续集成并有助于整体发布的容器虚拟化技术

Docker与传统虚拟化方式的不同之处

- 传统虚拟机技术是虚拟出一套硬件后，在其上运行一个完整操作系统，在该系统上再运行所需应用进程；
- 容器内的应用进程直接运行于宿主的内核，容器内没有自己的内核且也没有进行硬件虚拟。因此容器要比传统虚拟机更为轻便。
- 每个容器之间互相隔离，每个容器有自己的文件系统 ，容器之间进程不会相互影响，能区分计算资源。

Docker能做什么

- 技术职级的变化
    - coder
    - programmer
    - software engineer
    - DevOps engineer
- 开发/运维（DevOps）新一代开发工程师
    - 一次构建、随处运行
        - 更快速的应用交付和部署
        - 更便捷的升级和扩缩容
        - 更简单的系统运维
        - 更高效的计算资源利用

Docker的基本组成

1. 镜像

   Docker 镜像（Image）就是一个**只读**的模板。镜像可以用来创建 Docker 容器，一个镜像可以创建很多容器。它也相当于是一个root文件系统。比如官方镜像 centos:7 就包含了完整的一套 centos:7 最小系统的 root 文件系统。相当于容器的"源代码"，docker镜像文件类似于Java的类模板，而docker容器实例类似于java中new出来的实例对象。

2. 容器

   - 从面向对象的角度

     Docker 利用容器（Container）独立运行的一个或一组应用，应用程序或服务运行在容器里面，容器就类似于一个虚拟化的运行环境，容器是用镜像创建的运行实例。就像是Java中的类和实例对象一样，镜像是静态的定义，容器是镜像运行时的实体。容器为镜像提供了一个标准的和隔离的运行环境，它可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台。

   - 从镜像容器的角度

     可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。

3. 仓库

   仓库（Repository）是集中存放镜像文件的场所。类似于Maven仓库，存放各种jar包的地方；github仓库，存放各种git项目的地方；Docker公司提供的官方registry被称为Docker Hub，存放各种镜像模板的地方。

   仓库分为公开仓库（Public）和私有仓库（Private）两种形式。

   最大的公开仓库是 Docker Hub(https://hub.docker.com/)，存放了数量庞大的镜像供用户下载。

   国内的公开仓库包括阿里云 、网易云等...

4. 总结：

   Docker 本身是一个容器运行载体或称之为管理引擎。我们把应用程序和配置依赖打包好形成一个可交付的运行环境，这个打包好的运行环境就是image镜像文件。只有通过这个镜像文件才能生成Docker容器实例(类似Java中new出来一个对象)。 image文件可以看作是容器的模板。Docker 根据 image 文件生成容器的实例。同一个 image 文件，可以生成多个同时运行的容器实例。

   镜像文件：

   image 文件生成的容器实例，本身也是一个文件，称为镜像文件。

   容器实例：

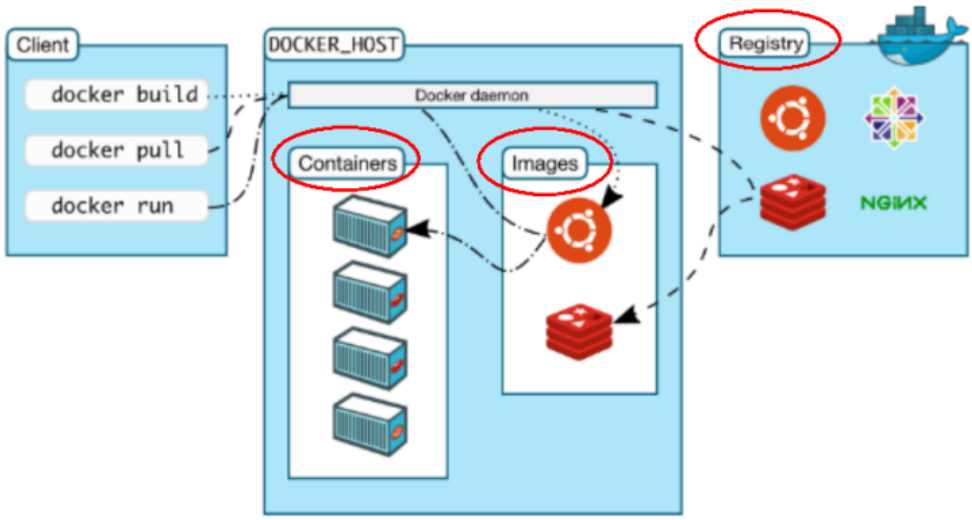
   一个容器运行一种服务，当我们需要的时候，就可以通过docker客户端创建一个对应的运行实例，也就是我们的容器

   仓库：

   就是放一堆镜像的地方，我们可以把镜像发布到仓库中，需要的时候再从仓库中拉下来就可以了。

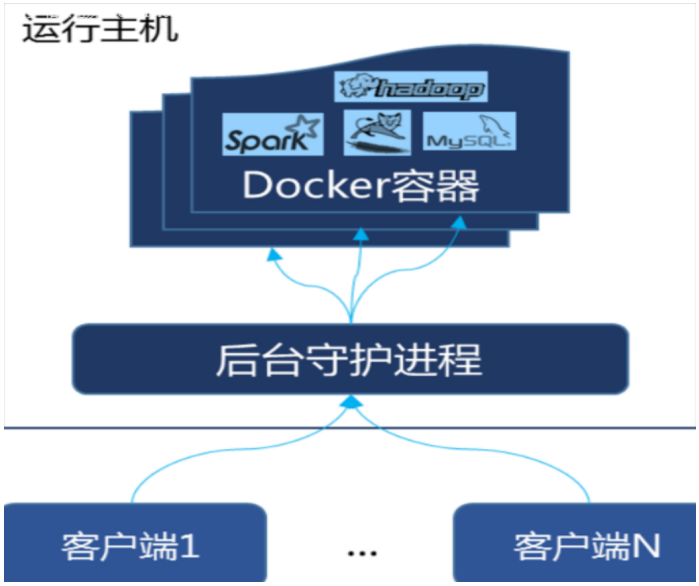
Docker平台架构

1.1

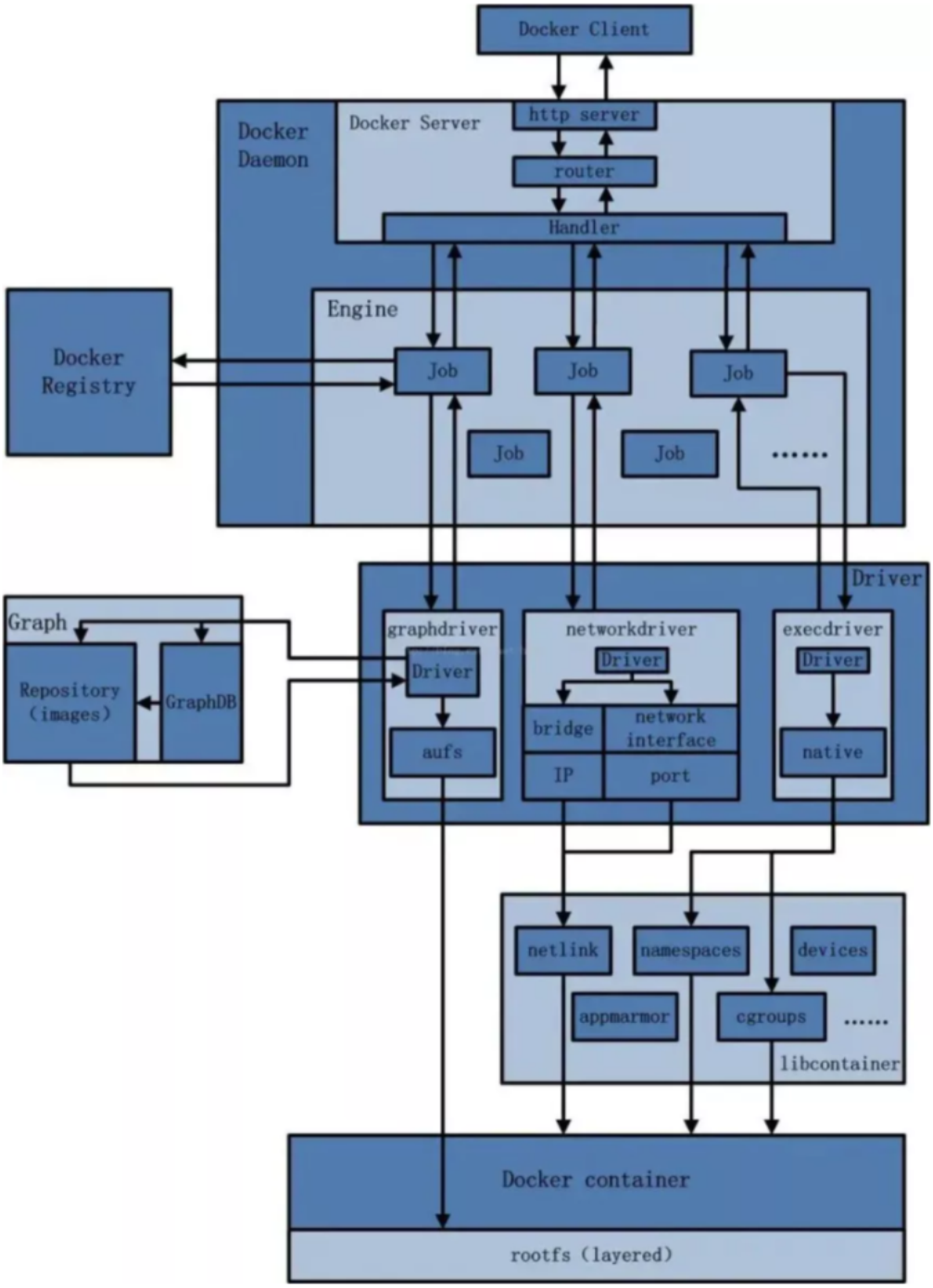


1.2 Docker工作原理

Docker是一个Client-Server结构的系统，Docker守护进程运行在主机_上，然后通过Socket连接从客户端访问，守护进程从客户端接受命令并管理运行在主机上的容器。



2.1



Docker 是一个 C/S 模式的架构，后端是一个松耦合架构，众多模块各司其职。

Docker运行的基本流程

1. 用户是使用Docker Client与Docker Daemon建立通信，并发送请求给后者。
2. Docker Daemon作为Docker架构中的主体部分，首先提供Docker Server的功能使其可以接受Docker Client的请求。
3. Docker Engine执行Docker内部的一系列工作，每一项工作都是以一个Job的形式的存在。
4. Job的运行过程中，当需要容器镜像时，则从Docker Registry中下载镜像，并通过镜像管理驱动Graph driver将下载镜像以Graph的形式存储。
5. 当需要为Docker创建网络环境时，通过网络管理驱动Network driver创建并配置Docker容器网络环境。
6. 当需要限制Docker容器运行资源或执行用户指令等操作时，则通过Exec driver来完成。
7. Libcontainer是一项独 立的容器管理包，Network driver以及Exec driver都是通过L ibcontainer来实现具体对容器进行的操作。

Docker安装

官方安装文档：https://docs.docker.com/engine/install/centos/

安装步骤

1. 确保操作系统版本满足最低版本要求
2. 卸载旧版本
3. 安装gcc相关

   安装gcc：`yum -y install gcc`

   安装gcc-c++：`yum -y install gcc-c++`
4. 安装需要的软件包

   `yum install -y yum-utils`
5. 设置stable镜像仓库

   `yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo`
6. 更新yum软件包索引

   `yum makecache fast`
7. 安装DOCKER CE

   `yum -y install docker-ce docker-ce-cli containerd.io`
8. 启动docker

   `systemctl start docker`
9. 测试版本

   `docker version`

卸载Docker

1. `systemctl stop docker`
2. `yum remove docker-ce docker-ce-cli containerd.io`
3. `rm -rf /var/lib/docker`
4. `rm -rf /var/lib/containerd`

配置阿里云镜像加速

https://promotion.aliyun.com/ntms/act/kubernetes.html

步骤

1. 注册一个属于自己的阿里云账户(可复用淘宝账号)
2. 获得加速器地址连接
   - 登录阿里云开发者平台
   - 点击控制台
   - 选择容器镜像服务
   - 获取加速器地址

3. 粘贴脚本直接执行
4. 重启服务器

   `systemctl daemon-reload`

   `systemctl restart docker`

hello world

`docker run hello-world`

```
[root@wangwei ~]# docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

底层原理

为什么Docker会比VM虚拟机快?

1. docker有着比虚拟机更少的抽象层

   由于docker不需要Hypervisor(虚拟机)实现硬件资源虚拟化,运行在docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上docker将会在效率上有明显优势。

2. docker利用的是宿主机的内核,而不需要加载操作系统OS内核

   当新建一个容器时,docker不需要和虚拟机一样重新加载一个操作系统内核。进而避免引寻、加载操作系统内核返回等比较费时费资源的过程,当新建一个虚拟机时,虚拟机软件需要加载OS,返回新建过程是分钟级别的。而docker由于直接利用宿主机的操作系统,则省略了返回过程,因此新建一个docker容器只需要几秒钟。

	Docker容器	虚拟机（VM）
操作系统	与宿主机共享OS	宿主机OS上运行虚拟机OS
存储大小	镜像小，便于存储与传输	镜像庞大（vmdk、vdi等）
运行性能	几乎无额外性能损失	操作系统额外的CPU、内存消耗
移植性	轻便、灵活，适应于Linux	笨重，与虚拟化技术耦合度高
硬件亲和性	面向软件开发者	面向硬件运维者
部署速度	快速，秒级	较慢，10s以上

Docker常用命令

帮助启动类命令

1. 启动Docker：`systemctl start docker`
2. 停止Docker：`systemctl stop docker`
3. 重启Docker：`systemctl restart docker`
4. 查看Docker状态：`systemctl status docker`
5. 开机启动：`systemctl enable docker`
6. 查看Docker概要信息：`docker info`
7. 查看docker总体帮助文档：`docker --help`
8. 查看docker命令帮助文档：`docker 具体命令 --help`

镜像容器

- 列出本地主机上的镜像：`docker images [OPTIONS]`

OPTIONS	含义
-a	列出本地所有的镜像（含历史映像层）
-q	只显示镜像id

```
[root@wangwei ~]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
itwang/myubuntu     1.3       5f9f97635b4d   5 hours ago      176MB
wang/ubuntu         3.7       f749e94637fb   21 hours ago     72.8MB
tomcat              latest    fb5657adc892   4 months ago     680MB
ubuntu              latest    ba6acccedd29   6 months ago     72.8MB
hello-world         latest    feb5d9fea6a5   7 months ago     13.3kB
redis               6.0.8     16ecd2772934   18 months ago    104MB
[root@wangwei ~]#
```

表头单词	含义
PEPOSITORY	表示镜像的仓库源
TAG	镜像的标签版本号
IMAGE ID	镜像ID
CREATED	镜像创建时间
SIZE	镜像大小

同一仓库源可以有多个 TAG版本，代表这个仓库源的不同个版本，我们使用 REPOSITORY:TAG 来定义不同的镜像。

如果你不指定一个镜像的版本标签，例如你只使用 ubuntu，docker 将默认使用 ubuntu:latest 镜像。

- 从Docker Hub查找镜像：`docker search [OPTIONS] 某个XXX镜像名字`

  网站：https://hub.docker.com

OPTIONS	含义
--limit	只列出N个镜像，默认25个

- 下载镜像：`docker pull 某个XXX镜像名字[:TAG]`

  说明：没有:tag就是最新版本latest
- 查看镜像/容器/数据卷所占的空间：`docker system df`
- 删除镜像
  - 删除单个镜像：`docker rmi  -f 镜像ID`
  - 删除多个镜像：`docker rmi -f 镜像名1:TAG 镜像名2:TAG`
  - 删除全部镜像：`docker rmi -f $(docker images -qa)` 慎重!!!

容器命令

声明：有镜像才能创建容器，这是根本前提

- 新建+启动容器：`docker run [OPTIONS] IMAGE [COMMAND] [ARG...]`

  OPTIONS说明：有些是一个减号，有些是两个

OPTIONS	含义
--name="容器新名字"	为容器指定一个名称，没有则系统自动指定一个随机名称
-d	后台运行容器并返回容器id，也即启动守护式容器(后台运行)
-i	以交互模式运行容器，通常与-t同时使用
-t	为容器重新分配一个伪输入终端，通常与-i同时使用，也即启动交互式容器(前台有伪终端，等待交互)
-P	随机端口映射，大写P
-p	指定端口映射，小写p

  使用镜像centos:latest以交互模式启动-一个容器,在容器内执行/bin/bash命令。

```
[root@wangwei ~]# docker run -it ubuntu /bin/bash
root@1644dcdd3bea:/#
```

  /bin/bash：放在镜像名后的命令，这里我们希望有个交互式Shell，因此用的是/bin/bash。

  退出终端：`exit`
- 列出当前所有正在运行的容器：`docker ps [OPTIONS]`

OPTIONS	含义
-a	列出当前所有正在运行的容器+历史上运行过的
-l	显示最近创建的容器
-n	显示最近n个创建的容器
-q	静默模式，只显示容器编号

- 退出容器
  - `exit`：run进去容器，exit退出，容器停止

    - `Ctrl+p+q`：run进去容器，ctrl+p+q退出，容器不停止
- 启动已停止运行的容器：`docker start 容器ID或者容器名`
- 重启容器：`docker restart 容器ID或者容器名`
- 停止容器：`docker stop 容器ID或者容器名`
- 强制停止容器：`docker kill 容器ID或容器名`
- 删除已停止的容器：`docker rm 容器ID`
    - 一次删除多个容器实例(慎重！！！)

      `docker rm -f $(docker ps -a -q)`

      `docker ps -a -q | xargs docker rm`
- 启动守护式容器

  在大部分的场景下，我们希望 docker 的服务是在后台运行的，我们可以过 -d 指定容器的后台运行模式。

  `docker run -d 容器名`

  说明：Docker容器后台运行就必须有一个前台进程，容器运行的命令如果不是那些一直挂起的命令，就是会自动退出的。
- 查看容器日志：`docker logs 容器ID`
- 查看容器内运行的进程：`docker top 容器ID`
- 查看容器内部细节：`docker inspect 容器ID`
- 进入正在运行的容器并以命令行交互

  ①`docker exec -it 容器ID bashShell`

  ②重新进入：`docker attach 容器ID`

  区别：

  attach 直接进入容器启动命令的终端，不会启动新的进程，用exit退出，会导致容器的停止。

  exec 是在容器中打开新的终端，并且可以启动新的进程，用exit退出，不会导致容器的停止。
- 从容器内拷贝文件到主机上：`docker cp  容器ID:容器内路径 目的主机路径`
- 导入和导出容器

  export：导出容器的内容留作为一个tar归档文件[对应import命令] `docker export 容器ID > 文件名.tar`

  import：从tar包中的内容创建一个新的文件系统再导入为镜像[对应export]

Docker镜像

镜像

是一种轻量级、可执行的独立软件包，它包含运行某个软件所需的所有内容，我们把应用程序和配置依赖打包好形成一个可交付的运行环境(包括代码、运行时需要的库、环境变量和配置文件等)，这个打包好的运行环境就是image镜像文件。只有通过这个镜像文件才能生成Docker容器实例(类似Java中new出来一个对象)。

分层的镜像

以我们的pull为例，在下载的过程中我们可以看到docker的镜像好像是在一层一层的在下载

UnionFS（联合文件系统）

UnionFS（联合文件系统）：Union文件系统（UnionFS）是一种分层、轻量级并且高性能的文件系统，它支持对文件系统的修改作为一次提交来一层层的叠加，同时可以将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)。Union 文件系统是 Docker 镜像的基础。镜像可以通过分层来进行继承，基于基础镜像（没有父镜像），可以制作各种具体的应用镜像。

特性：一次同时加载多个文件系统，但从外面看起来，只能看到一个文件系统，联合加载会把各层文件系统叠加起来，这样最终的文件系统会包含所有底层的文件和目录。

Docker镜像加载原理

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。

bootfs(boot file system)主要包含bootloader和kernel, bootloader主要是引导加载kernel, Linux刚启动时会加载bootfs文件系统，在Docker镜像的最底层是引导文件系统bootfs。这一层与我们典型的Linux/Unix系统是一样的，包含boot加载器和内核。当boot加载完成之后整个内核就都在内存中了，此时内存的使用权已由bootfs转交给内核，此时系统也会卸载bootfs。

rootfs (root file system)，在bootfs之上。包含的就是典型 Linux 系统中的 /dev, /proc, /bin, /etc 等标准目录和文件。rootfs就是各种不同的操作系统发行版，比如Ubuntu，Centos等等。

对于一个精简的OS，rootfs可以很小，只需要包括最基本的命令、工具和程序库就可以了，因为底层直接用Host的kernel，自己只需要提供 rootfs 就行了。由此可见对于不同的linux发行版, bootfs基本是一致的, rootfs会有差别, 因此不同的发行版可以公用bootfs。

为什么 Docker 镜像要采用这种分层结构呢

镜像分层最大的一个好处就是共享资源，方便复制迁移，就是为了复用。比如说有多个镜像都从相同的 base 镜像构建而来，那么 Docker Host 只需在磁盘上保存一份 base 镜像；同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

重点

Docker镜像层都是只读的，容器层是可写的，当容器启动时，一个新的可写层被加载到镜像的顶部。这一层通常被称作“容器层”，“容器层”之下的都叫“镜像层”。

所有对容器的改动 - 无论添加、删除、还是修改文件都只会发生在容器层中。只有容器层是可写的，容器层下面的所有镜像层都是只读的。

Docker镜像commit

docker commit提交容器副本使之成为一个新的镜像

`docker commit -m="提交的描述信息" -a="作者" 容器ID 要创建的目标镜像名:[标签名]`

安装vim并提交使之成为一个新镜像

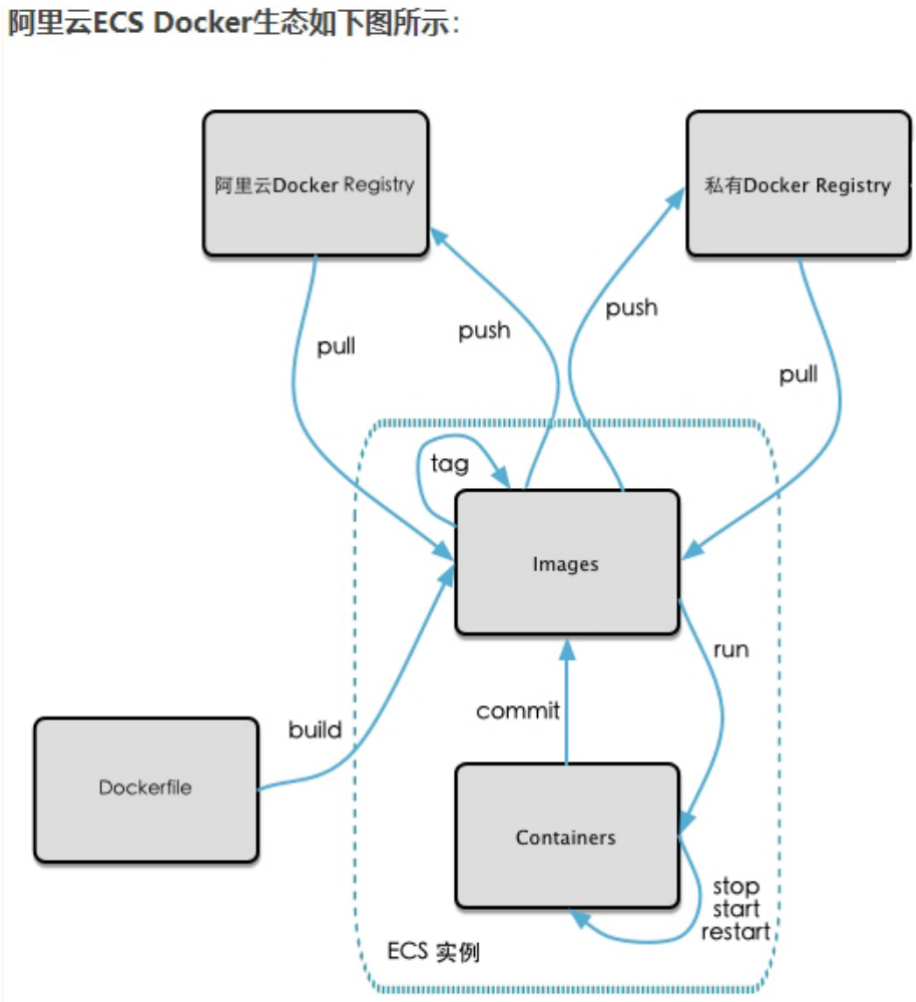
1. 更新包管理工具：`apt-get update`
2. 安装需要的vim：`apt-get -y install vim`
3. 提交

```
[root@wangwei ~]# docker commit -m="vim cmd" -a="wang" 1253688a7cbf itwang/myubuntu:1.3
sha256:5f9f97635b4d6b6c23a7f0860ab12f6473785e50b4604bbf0b06f5586cae375f
[root@wangwei ~]# docker images
REPOSITORY          TAG       IMAGE ID       CREATED          SIZE
itwang/myubuntu     1.3       5f9f97635b4d   5 seconds ago    176MB
wang/ubuntu         3.7       f749e94637fb   16 hours ago     72.8MB
tomcat              latest    fb5657adc892   4 months ago     680MB
ubuntu              latest    ba6acccedd29   6 months ago     72.8MB
hello-world         latest    feb5d9fea6a5   7 months ago     13.3kB
redis               6.0.8     16ecd2772934   18 months ago    104MB
```

安装了vim的镜像明显比原镜像大

本地镜像发送到阿里云

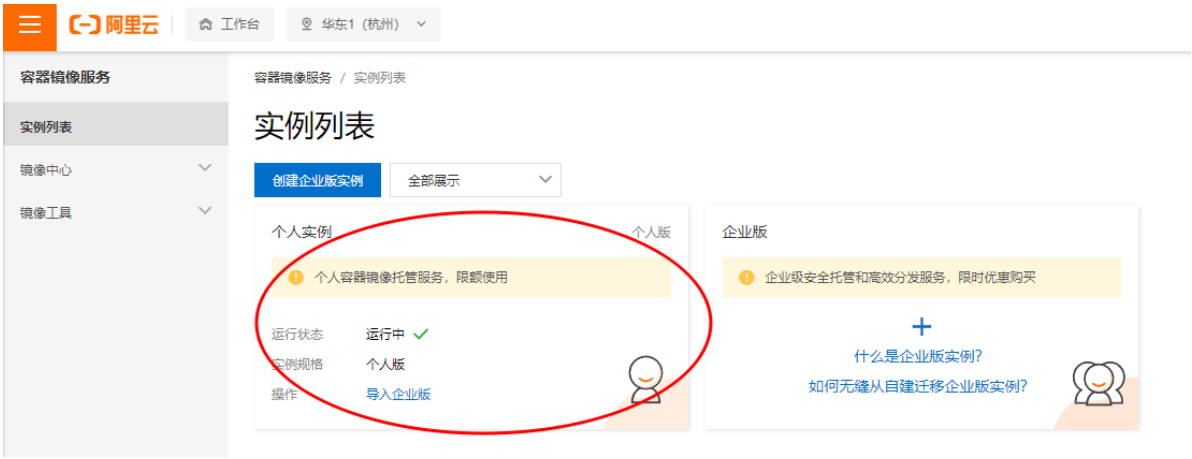
阿里云ECS Docker生态如下图所示：



阿里云设置个人镜像实例

一、进入控制台，选择容器镜像服务

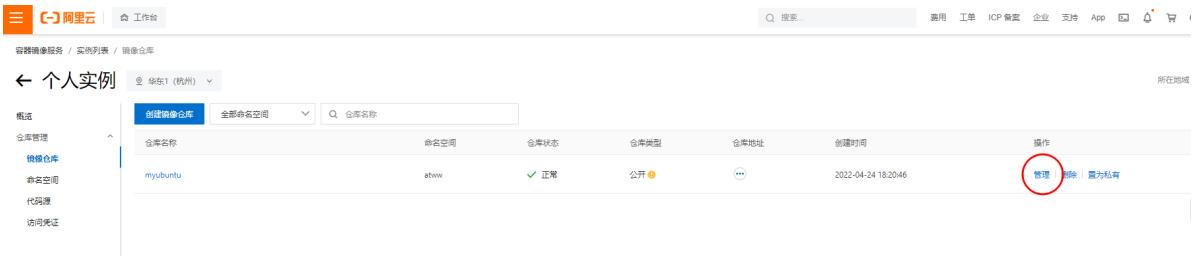
二、选择个人实例



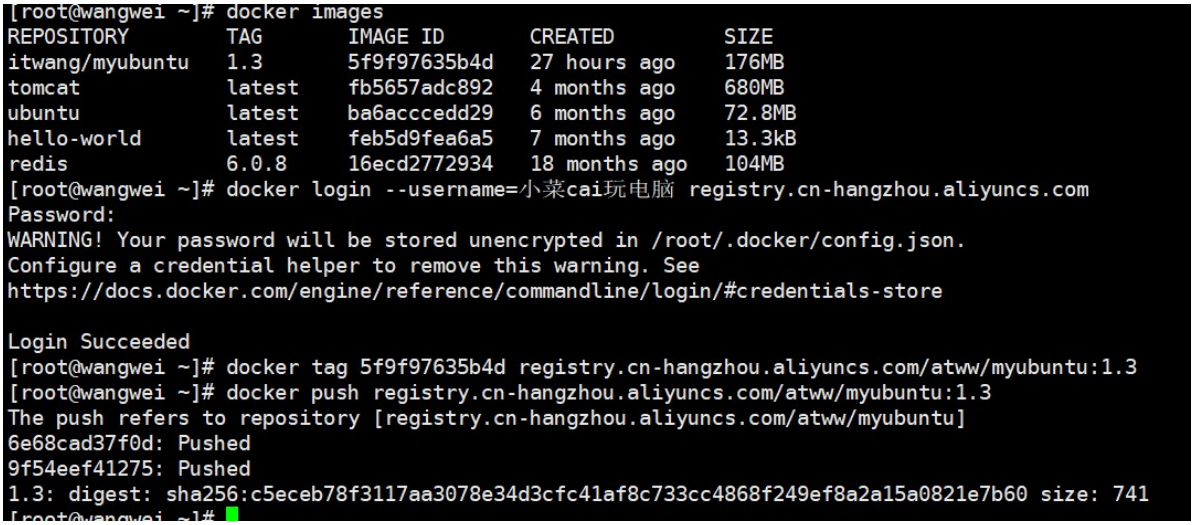
三、选择命名空间，创建命名空间

四、创建镜像仓库

五、进入仓库获得脚本



将镜像推送到阿里云



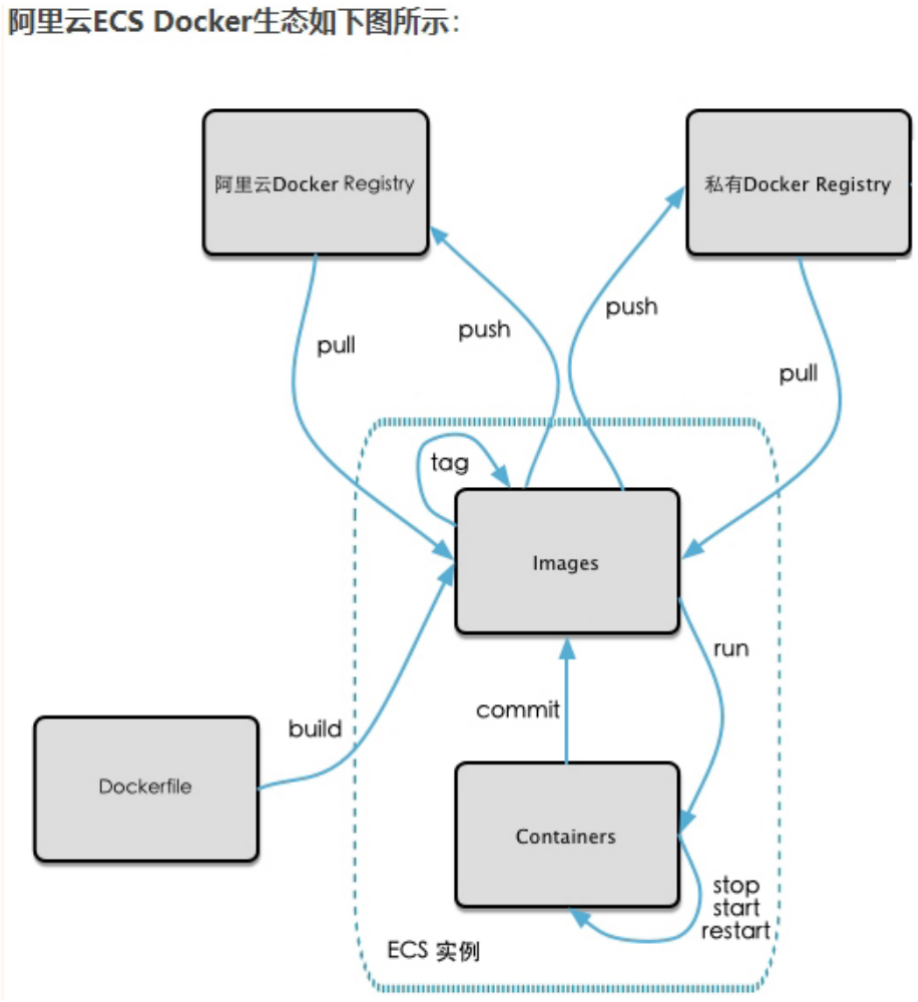

```
$ docker login --username=小菜cai玩电脑 registry.cn-hangzhou.aliyuncs.com
$ docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/atww/myubuntu:[镜像版本号]
$ docker push registry.cn-hangzhou.aliyuncs.com/atww/myubuntu:[镜像版本号]
```

从阿里云拉取镜像

```
$ docker pull registry.cn-hangzhou.aliyuncs.com/atww/myubuntu:[镜像版本号]
```

本地镜像发送到私有库

阿里云ECS Docker生态如下图所示：



Docker Registry

Docker Registry是官方提供的工具，可以用于构建私有镜像仓库

推送步骤

一、下载镜像Docker Registry

```
docker pull registry
```

二、运行私有库Registry，相当于本地有个私有Docker hub

```
docker run -d -p 5000:5000 -v /ww/myregistry/:/tmp/registry --privileged=true registry
```

说明：默认情况，仓库被创建在容器的/var/lib/registry目录下，建议自行用容器卷映射，方便于宿主机联调

三、curl验证私服库上有什么镜像

```
curl -XGET http://虚拟ip地址:5000/v2/_catalog
```

四、将新镜像修改符合私服规范的Tag

```
docker tag 镜像:Tag Host:Port/Repository:Tag
```

五、修改配置文件使之支持http

```
vim /etc/docker/daemon.json
{
  "registry-mirrors": ["https://aa25jngu.mirror.aliyuncs.com"],
  "insecure-registries": ["192.168.111.162:5000"]
}
```

上述理由：docker默认不允许http方式推送镜像，通过配置选项来取消这个限制。====> 修改完后如果不生效，建议重启docker

六、push推送到私服库

`docker push 虚拟ip:5000/镜像名:tag`

七、curl验证私服库上有什么镜像

`curl -XGET http://虚拟ip地址:5000/v2/_catalog`

八、pull到本地并运行

`docker pull 虚拟ip:5000/zzyyubuntu:1.2`

数据卷

坑：

容器卷记得加入：`--privileged=true`

Docker挂载主机目录访问如果出现cannot open directory .: Permission denied

解决办法：在挂载目录后多加一个--privileged=true参数即可

什么是数据卷

卷就是目录或文件，存在于一个或多个容器中，由docker挂载到容器，但不属于联合文件系统，因此能够绕过Union File System提供一些用于持续存储或共享数据的特性

卷的设计目的就是数据的持久化，完全独立于容器的生存周期，因此Docker不会在容器删除时删除其挂载的数据卷

一句话：有点类似我们Redis里面的rdb和aof文件，将docker容器内的数据保存进宿主机的磁盘中

作用

将运用与运行的环境打包镜像，run后形成容器实例运行 ，但是我们对数据的要求希望是持久化的。

Docker容器产生的数据，如果不备份，那么当容器实例删除后，容器内的数据自然也就没有了。

为了能保存数据在docker中我们使用卷。

特点：

1：数据卷可在容器之间共享或重用数据

2：卷中的更改可以直接实时生效，爽

3：数据卷中的更改不会包含在镜像的更新中

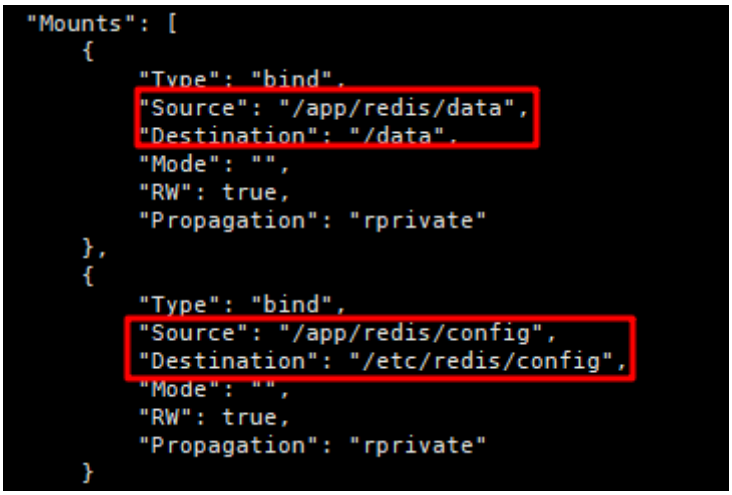
4：数据卷的生命周期一直持续到没有容器使用它为止

运行一个带有容器卷存储功能的容器实例

`docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录 镜像名`

查看数据卷挂载情况

`docker inspect 容器ID`



容器和宿主机之间数据共享

1 docker修改，主机同步获得

2 主机修改，docker同步获得

3 docker容器stop，主机修改，docker容器重启看数据是否同步。

读写规则映射添加说明

读写(默认)

`docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:rw 镜像名`

只读

容器实例内部被限制，只能读取不能写

`docker run -it --privileged=true -v /宿主机绝对路径目录:/容器内目录:ro 镜像名`

卷的继承和共享

容器2继承容器1的卷

`docker run -it --privileged=true --volumes-from 父类 --name name 镜像名`

Docker常规安装

总体步骤

1. 搜索镜像
2. 拉取镜像
3. 查看镜像
4. 启动镜像==》服务端口映射
5. 停止容器
6. 移除容器

安装tomcat

一、docker hub上面查找tomcat镜像

`docker search tomcat`

二、从docker hub上拉取tomcat镜像到本地

`docker pull tomcat`

三、docker images查看是否有拉取到的tomcat

四、使用tomcat镜像创建容器实例(也叫运行镜像)

`docker run -it -p 8080:8080 tomcat`

五、访问tomcat

可能出现404的问题

1.可能没有映射端口或者没有关闭防火墙

2.最新版的tomcat需要把webapps.dist目录换成webapps，然后重启

六、安装免修改版的tomcat

```
docker pull billygoo/tomcat8-jdk8
docker run -d -p 8080:8080 --name mytomcat8 billygoo/tomcat8-jdk8
```

安装mysql

一、docker hub上面查找mysql镜像

`docker search mysql`

二、从docker hub上拉取mysql镜像到本地

`docker pull mysql:tag`

三、新建mysql容器实例

```
docker run -d -p 3306:3306 --privileged=true -v /ww/mysql/log:/var/log/mysql
-v /ww/mysql/data:/var/lib/mysql -v /ww/mysql/conf:/etc/mysql/conf.d -e
MYSQL_ROOT_PASSWORD=密码  --name mysql mysql:5.7
```

四、新建my.cnf，修改默认字符集编码

```
[client]
default_character_set=utf8
[mysqld]
collation_server = utf8_general_ci
character_set_server = utf8
```

五、重启mysql查看字符编码

`show variables like 'character%'`

安装redis

一、docker hub上面查找redis镜像

`docker search redis`

二、从docker hub上拉取redis镜像到本地

`docker pull redis:tag`

三、在宿主机下新建目录/app/redis

`mkdir -p /app/redis`

四、将一个redis.conf文件模板拷贝进/app/redis目录下

五、/app/redis目录下修改redis.conf文件

1.开启redis密码验证

`requirepass password`

2.允许redis外地连接

`注释掉 bind 127.0.0.1`

3.注释或删掉daemonize或设为daemonize no

4.开启redis数据持久化，appendonly yes

六、使用redis6.2.6镜像创建容器(也叫运行镜像)

```
docker run -d -p 6379:6379 --name redis6 --privileged=true -v
/ww/redis/config:/etc/redis/config/ -v /ww/redis/data:/data    redis:6.2.6
redis-server /etc/redis/config/redis.conf
```

七、测试redis-cli连接上来

`redis-cli`