将本地镜像发布到阿里云

有时候需要共享镜像或者习惯使用自己定义的镜像，可以注册私有仓库，国内推荐使用阿里云

步骤：

1.登录阿里云容器镜像服务：https://cr.console.aliyun.com/cn-hangzhou/repositories

2.将镜像推送到阿里云

```
# 登录阿里云的docker仓库
$ sudo docker login --username=[用户名] registry.cn-hangzhou.aliyuncs.com
# 创建指定镜像的tag，归入某个仓库
$ sudo docker tag [镜像ID] registry.cn-hangzhou.aliyuncs.com/huaan/huaan:[镜像版本号]
# 讲镜像推送到仓库
$ sudo docker push registry.cn-hangzhou.aliyuncs.com/huaan/huaan:[镜像版本号]
```

3.拉取镜像

```
docker pull registry.cn-hangzhou.aliyuncs.com/coldest7/mytom:v1
```

Docker 网络

Docker允许通过外部访问容器或容器互联的方式来提供网络服务。

安装Docker时，会自动安装一块Docker网卡称为docker0，用于Docker各容器及宿主机的网络通信，网段为172.0.0.1。

Docker网络中有三个核心概念：沙盒（Sandbox）、网络（Network）、端点（Endpoint）。

- 沙盒，提供了容器的虚拟网络栈，也即端口套接字、IP路由表、防火墙等内容。隔离容器网络与宿主机网络，形成了完全独立的容器网络环境。
- 网络，可以理解为Docker内部的虚拟子网，网络内的参与者相互可见并能够进行通讯。Docker的虚拟网络和宿主机网络是存在隔离关系的，其目的主要是形成容器间的安全通讯环境。
- 端点，位于容器或网络隔离墙之上的洞，主要目的是形成一个可以控制的突破封闭的网络环境的出入口。当容器的端点与网络的端点形成配对后，就如同在这两者之间搭建了桥梁，便能够进行数据传输了。

Docker的四种网络模式

Docker服务在启动的时候会创建三种网络，bridge、host和none，还有一种共享容器的模式container

Bridge

桥接模式，主要用来对外通信的，docker容器默认的网络使用的就是bridge。

使用bridge模式配置容器自定的网络配置

```
# 配置容器的主机名
docker run --name t1 --network bridge -h [自定义主机名] -it --rm busybox
# 自定义DNS
docker run --name t1 --network bridge --dns 114.114 -it --rm busybox
# 给host文件添加一条
docker run --name t1 --network bridge --add-host [hostname]:[ip] -it --rm busybox
```

Host

host类型的网络就是主机网络的意思，绑定到这种网络上面的容器，内部使用的端口直接绑定在主机上对应的端口，而如果容器服务没有使用端口，则无影响。

None

从某种意义上来说，none应该算不上网络了，因为它不使用任何网络，会形成一个封闭网络的容器

container

共享另外一个容器的network namespace，和host模式差不多，只是这里不是使用宿主机网络，而是使用的容器网络

开放端口

Docker0为NAT桥，所以容器一般获得的是私有网络地址

给docker run命令使用-p选项即可实现端口映射，无需手动添加规则

- -p 选项的使用
  - `-p <containerPort>`
    - 将指定的容器端口映射到主机所有地址的一个动态端口
  - `-p <hostPort>:<containerPort>`
    - 将容器端口`<containerPort>`映射到指定的主机端口`<hostPort>`
  - `-p <ip>::<containerPort>`
    - 将指定的容器端口`<containerPort>`映射到主机指定`<ip>`的动态端口
  - `-p <ip>:<hostPort>:<containerPort>`
    - 将指定的容器端口`<containerPort>`映射至主机指定`<ip>`的端口`<hostPort>`
- 动态端口指随机端口，可以使用docker port命令查看具体映射结果
- -P 暴露所有端口（所有端口指构建镜像时EXPOSE的端口）

自定义docker0桥的网络属性信息：/etc/docker/daemon.json文件

```
{
"bip": "192.168.1.5/24",
"fixed-cidr": "10.20.0.0/16",
"fixed-cidr-v6": "2001:db8::/64",
"mtu": 1500,
"default-gateway": "10.20.1.1",
"default-gateway-v6": "2001:db8:abcd::89",
"dns": ["10.20.1.2","10.20.1.3"]
}
```

核心选项为bip，即bridge ip之意，用于指定docker0桥自身的IP地址；其它选项可通过此地址计算得出

远程连接

创建自定义的桥

```
docker network create -d bridge --subnet "172.26.0.0/16" --gateway "172.26.0.1" mybr0
```

Docker Compose

从上一节课我们了解到可以使用一个Dockerfile模板文件来快速构建一个自己的镜像并运行为应用容器。但是在平时工作的时候，我们会碰到多个容器要互相配合来使用的情况，比如数据库加上咱们Web应用等等。这种情况下，每次都要一个一个启动容器设置命令变得麻烦起来，所以Docker Compose诞生了。

简介

Compose的作用是"定义和运行多个Docker容器的应用"。使用Compose，你可以在一个配置文件（yaml格式）中配置你应用的服务，然后使用一个命令，即可创建并启动配置中引用的所有服务。

Compose中两个重要概念：

- 服务 (service)：一个应用的容器，实际上可以包括若干运行相同镜像的容器实例。
- 项目 (project)：由一组关联的应用容器组成的一个完整业务单元，在 docker-compose.yml文件中定义。

安装

Compose支持三平台Windows、Mac、Linux，安装方式各有不同。我这里使用的是Linux系统，其他系统安装方法可以参考官方文档和开源GitHub链接：

Docker Compose官方文档链接：https://docs.docker.com/compose

Docker Compose GitHub链接：https://github.com/docker/compose

Linux上有两种安装方法，Compose项目是用Python写的，可以使用Python-pip安装，也可以通过GitHub下载二进制文件进行安装。

通过Python-pip安装

1.安装Python-pip

```
yum install -y epel-release
yum install -y python-pip
```

2.安装docker-compose

```
pip install docker-compose
```

3.验证是否安装

```
docker-compose version
```

4.卸载

```
pip uninstall docker-compose
```

通过GitHub链接下载安装

非ROOT用户记得加sudo

1.通过GitHub获取下载链接，以往版本地址：https://github.com/docker/compose/releases

```
curl -L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

2.给二进制下载文件可执行的权限

```
chmod +x /usr/local/bin/docker-compose
```

3.可能没有启动程序，设置软连接，比如:

```
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

4.验证是否安装

```
docker-compose version
```

5.卸载

如果是二进制包方式安装的，删除二进制文件即可。

```
rm /usr/local/bin/docker-compose
```

简单实例

Compose的使用非常简单，只需要编写一个docker-compose.yml，然后使用docker-compose 命令操作即可。
docker-compose.yml描述了容器的配置，而docker-compose 命令描述了对容器的操作。

1.我们使用一个微服务项目先来做一个简单的例子，首先创建一个compose的工作目录，然后创建一个eureka文件夹，里面放可执行jar包和编写一个Dockerfile文件，目录结构如下：

```
compose
    eureka
        Dockerfile
        eureka-server-2.0.2.RELEASE.jar
```

2.在compose目录创建模板文件docker-compose.yml文件并写入以下内容：

```
version: '1'
services:
  eureka:
    build: ./eureka
    ports:
     - 3000:3000
    expose:
     - 3000
```

Docker Compose模板文件常用指令

image

指定镜像名称或者镜像id，如果该镜像在本地不存在，Compose会尝试pull下来。

示例：

image: java:8

build

指定Dockerfile文件的路径。可以是一个路径，例如：

build: ./dir

也可以是一个对象，用以指定Dockerfile和参数，例如：

build: context: ./dir dockerfile: Dockerfile-alternate args: buildno: 1

command

覆盖容器启动后默认执行的命令。

示例：

command: bundle exec thin -p 3000

也可以是一个list，类似于Dockerfile总的CMD指令，格式如下：

command: [bundle, exec, thin, -p, 3000]

links

链接到其他服务中的容器。可以指定服务名称和链接的别名使用SERVICE:ALIAS 的形式，或者只指定服务名称，示例：

web: links: - db - db:database - redis

external_links

表示链接到docker-compose.yml外部的容器，甚至并非Compose管理的容器，特别是对于那些提供共享容器或共同服务。格式跟links类似，示例：

external_links: - redis_1 - project_db_1:mysql - project_db_1:postgresql

ports

暴露端口信息。使用宿主端口:容器端口的格式，或者仅仅指定容器的端口（此时宿主机将会随机指定端口），类似于docker run -p ，示例：

ports:

- "3000"
- "3000-3005"
- "8000:8000"
- "9090-9091:8080-8081"
- "49100:22"
- "127.0.0.1:8001:8001"
- "127.0.0.1:5000-5010:5000-5010"

expose

暴露端口，只将端口暴露给连接的服务，而不暴露给宿主机，示例：

expose: - "3000" - "8000"

volumes

卷挂载路径设置。可以设置宿主机路径 （HOST:CONTAINER） 或加上访问模式 （HOST:CONTAINER:ro）。示例：

volumes:

Just specify a path and let the Engine create a volume

- /var/lib/mysql

Specify an absolute path mapping

- /opt/data:/var/lib/mysql

Path on the host, relative to the Compose file

- ./cache:/tmp/cache

User-relative path

- ~/configs:/etc/configs/:ro

Named volume

- datavolume:/var/lib/mysql

volumes_from

从另一个服务或者容器挂载卷。可以指定只读或者可读写，如果访问模式没有指定，则默认是可读写。示例：

volumes_from:

- service_name
- service_name:ro
- container:container_name
- container:container_name:rw

environment

设置环境变量。可以使用数组或者字典两种方式。只有一个key的环境变量可以在运行Compose的机器上找到对应的值，这有助于加密的或者特殊主机的值。示例：

environment: RACK_ENV: development SHOW: 'true' SESSION_SECRET: environment: - RACK_ENV=development - SHOW=true - SESSION_SECRET

env_file

从文件中获取环境变量，可以为单独的文件路径或列表。如果通过 docker-compose -f FILE 指定了模板文件，则 env_file 中路径会基于模板文件路径。如果有变量名称与 environment 指令冲突，则以envirment 为准。示例：

env_file: .env env_file: - ./common.env - ./apps/web.env - /opt/secrets.env

extends

继承另一个服务，基于已有的服务进行扩展。

net

设置网络模式。示例：

net: "bridge" net: "host" net: "none" net: "container:[service name or container name/id]"

dns

配置dns服务器。可以是一个值，也可以是一个列表。示例：

dns: 8.8.8.8 dns: - 8.8.8.8 - 9.9.9.9

dns_search

配置DNS的搜索域，可以是一个值，也可以是一个列表，示例：

dns_search: example.com dns_search: - dc1.example.com - dc2.example.com

其它

docker-compose.yml 还有很多其他命令，可以参考docker-compose.yml文件官方文档：

https://docs.docker.com/compose/compose-file/

使用Docker Compose编排SpringCloud微服务

使用docker-compose一次性来编排三个微服务:eureka服务(eureka-server-2.0.2.RELEASE.jar)、user服务(user-2.0.2.RELEASE.jar)、power服务(power-2.0.2.RELEASE.jar)

1.创建一个工作目录和docker-compose模板文件

2.工作目录下创建三个文件夹eureka、user、power，并分别构建好三个服务的镜像文件

以eureka的Dockerfile为例:

```
# 基础镜像
FROM java:8
# 作者
MAINTAINER huaan
# 把可执行jar包复制到基础镜像的根目录下
ADD eureka-server-2.0.2.RELEASE.jar /eureka-server-2.0.2.RELEASE.jar
# 镜像要暴露的端口，如要使用端口，在执行docker run命令时使用-p生效
EXPOSE 3000
# 在镜像运行为容器后执行的命令
ENTRYPOINT ["java","-jar","/eureka-server-2.0.2.RELEASE.jar"]
```

目录文件结构：

```
compose
        docker-compose.yml
        eureka
                Dockerfile
                eureka-server-2.0.2.RELEASE.jar
        user
                Dockerfile
                user-2.0.2.RELEASE.jar
        power
                Dockerfile
                power-2.0.2.RELEASE.jar
```

3.编写docker-compose模板文件：

```
version: '1'
services:
  eureka:
    image: eureka:v1
    ports:
     - 8080:8080
  user:
    image: user:v1
    ports:
     - 8081:8081
  power:
    image: power:v1
    ports:
     - 8082:8082
```

4.启动微服务，可以加上参数-d后台启动

```
docker-compose up -d
```