# 仓库

1. 仓库（Repository)是**集中存放镜像**的地方。
2. 实际上注册服务器是管理仓库的的具体服务器，每个服务器可以由多个仓库，每个仓库下面有多个镜像
3. 例如：dl.dockerpool.com/ubuntu
   1. dl.dockerpool.com是注册服务器地址
   2. ubuntu是仓库名
4. 大部分时候并不严格区分注册服务器和仓库的概念

# Docker Hub

1. 目前Dokcer官方维护了一个公共仓库Docker Hub，大部分需求都可以通过在Docker Hub中直接下载镜像来实现

**登录**

1. 可以通过执行docker login命令来输入用户名、密码和邮箱来完成登录。
2. 如果没有账号，去指定网上注册。
3. 登录成功后，本地用户目录的.dockercfg将保存用户的认证信息

**基本操作**

1. 用户无需登录即可通过**docker search命令**来查找官方仓库中的镜像，并利用**docker pull命令**来将它下载到本地
2. 比如sudo docker search centos：
   1. 返回了很多镜像
   2. 关键字：镜像名字、描述、星级（受欢迎程度）、是否官方创建、是否自动创建
   3. 官方的镜像说明是官方项目组创建和维护的，automated资源允许用户验证镜像的来源和内容
3. 根据是否是官方提供，可以将镜像资源分为两类：
   1. 一种是项centos这样的基础镜像，被称为基础或根镜像。
      1. 这些镜像是由Docker公司创建、验证、支持、提供。
      2. 这样的镜像往往使用单个单词作为名字
   2. 另一种比如tianon/centos镜像，是由Docker的用户创建并维护的
      1. 往往带有用户名称前缀
      2. 可以通过前缀 user\_name/ 来指定使用某个用户提供的镜像
4. 查找的时候可以通过 **-s N 参数**指定仅显示评价为N星以上的镜像
5. 用户登录后可以通过**docker push命令**将镜像推送到Docker Hub

**自动创建**

1. 自动创建功能对于需要经常升级镜像中程序的情况来说，十分方便。有时候，用户创建了镜像，安装了某个软件，如果软件发布新版本则需要手动更新镜像。
2. 自动创建允许用户通过Docker Hub指定跟踪一个目标网站（目前支持GitHub等）上的项目，一旦项目发生新的提交，则自动执行创建。
3. 配置自动创建的步骤：
   1. 创建并登录Docker Hub以及目标网站
   2. 在目标网站连接账户到Docker Hub
   3. 在Docker Hub中配置一个自动创建
   4. 选取一个目标网站中的项目（需要Dockerfile）和分支
   5. 指定Dockerfile的位置，并提交创建
4. 之后，可以在Docker Hub的自动创建页面中跟踪每次创建的状态

# 私有仓库

1. 有时候使用Docker Hub这样的公共仓库可能不方便，用户可以创建一个本地仓库供私人使用，docker-registry是官方提供的工具，可以用于构建私有的镜像仓库

**容器运行registry**

1. 安装Docker后，可以通过获取官方registry镜像来运行：
   1. sudo docker run -d -p 5000:5000 registry
   2. 这将使用官方的registry镜像来启动本地的私有仓库。用户可以通过参数来配置私有仓库的位置
   3. 还可以指定本地路径下(如/home/user/registry-conf)下的配置文件
   4. sudo docker run -d -p 5000:5000 -v /home/user/registry-conf:/registry-conf -e DOCKER\_REGISTRY\_CONFIG=/registry-conf/config.yml registry
2. 默认情况下，仓库会被创建在容器的/tmp/registry下。可以通过 **-v 参数**来将镜像文件存放在本地的指定路径，例如将上传的镜像放到/opt/data/registry目录：
   1. sudo docker run -d -p 5000:5000 -v /opt/data/registry:/tmp/registry registry

**本地安装**

1. 对于Ubuntu等发行版，可以直接通过源安装
2. Ubuntu的安装指令：

sudo apt-get install -y build-essential python-dev libevent-dev python-pip liblzma-dev

sudo pip install docker-registry

1. 亦可以从docker-registry项目下载源码进行安装

sudo apt-get install build-essential python-dev libevent-dev python-pip libssl-dev liblzma-dev libffi-dev

git clone https://github.com/docker/docker-registry.git

cd docker-registry

sudo python setup.py install

1. 然后修改配置文件，主要修改dev模板段的storage\_path到本地的存储仓库的路径
   * 1. cp config/config\_sample.yml config/config.yml
2. 之后启动Web服务
   * 1. sudo gunicorn -c contrib/gunicorn.py docker\_registry.wsgi:application
   1. 或者
      1. sudo gunicorn --access-logfile - --error-logfile - -k gevent -b 0.0.0.0:5000 -w 4 --max-requests 100 docker\_registry.wsgi:application
3. 此时使用curl访问本地的5000端口，看到输出docker-registry的版本信息说明成功
4. **\*注：config/config\_sample.yml文件时实例配置文件**

**在私有仓库上传、下载、搜索镜像**

1. 创建好私有仓库之后，就可以使用**docker tag命令**来标记一个镜像，然后推送到仓库，别的机器上就可以下载下来了。
2. 格式为：Docker tag image[:tag] [registryhost/][username/]name[:tag]
3. 现在本机查看已有镜像：
   1. sudo docker\_images
4. 例如私有仓库地址为192.168.7.26:5000，将ba58这个镜像标记为192.168.7.26:5000/test:
   1. sudo docker tag ba58 192.168.7.26:5000/test
5. 利用docker push上传标记的镜像：
   1. sudo docker push 192.168.7.26:5000/test
6. 用curl查看仓库中的镜像，可以看到镜像成功上传：
   1. curl <http://192.168.7.26:5000/v1/search>
7. 到另外一台机器下载这个镜像：
   1. sudo docker pull 192.168.7.26:5000/test

# 仓库配置文件

1. Docke人的Registry利用配置文件提供了一些仓库的模板，用户可以直接使用它们来进行开发或生产部署

**模板**

1. 在config\_sample.yml文件中，可以看到一些现成的模板段：
   1. Common：基础配置
   2. Local：存储数据到本地文件系统
   3. S3：存储数据到AWS S3中
   4. Dev：使用local模板的基本配置
   5. Test：单元测试使用
   6. Prod：生产环境配置（基本恩s3配置类似）
   7. Gcs：存储数据到Google的云存储
   8. Swift：存储数据到OpenStack Swift服务
   9. Glance：翻出数据到OpenStack Glance服务，本地系统为后备
   10. glance-swift：存储数据到 OpenStack Glance 服务，Swift 为后备
   11. elliptics：存储数据到 Elliptics key/value 存储
2. 用户也可以添加自定义的模版段。
3. **默认情况下使用的模板是 dev**，可以添加 SETTINGS\_FLAVOR 到环境变量中以使用某个模板作为默认值，例如
   1. export SETTINGS\_FLAVOR=dev
4. 另外，配置文件中设置\_env:VARIABLENAME[:DEFAULT]可以从环境变量中加载值。
5. 示例配置

common:

    loglevel: info

    search\_backend: "\_env:SEARCH\_BACKEND:"

    sqlalchemy\_index\_database:

        "\_env:SQLALCHEMY\_INDEX\_DATABASE:sqlite:////tmp/docker-registry.db"

prod:

    loglevel: warn

    storage: s3

    s3\_access\_key: \_env:AWS\_S3\_ACCESS\_KEY

    s3\_secret\_key: \_env:AWS\_S3\_SECRET\_KEY

    s3\_bucket: \_env:AWS\_S3\_BUCKET

    boto\_bucket: \_env:AWS\_S3\_BUCKET

    storage\_path: /srv/docker

    smtp\_host: localhost

    from\_addr: docker@myself.com

    to\_addr: my@myself.com

dev:

    loglevel: debug

    storage: local

    storage\_path: /home/myself/docker

test:

    storage: local

    storage\_path: /tmp/tmpdockertmp