重要概念

✓ 仓库

registry 汉语意为*登记处或注册处*。在 docker 中，它登记了众多容器的镜像。默认是 *hub.docker.com*，由于它是国外的登记处，因此当进行 *docker pull...*等操作时，下载的速度非常缓慢。因此，我们通常会将其修改为国内的注册处，例如阿里云。在 Linux 系统上，通常在*/etc/docker/*目录下，创建 *daemon.json* 文件来修改 registry。

```
{
    "registry-mirrors": ["https://可在阿里云申请.mirror.aliyuncs.com"]
}
```

✓ 镜像

就像是 Java 中的类，像模板一样。镜像的绝对定位就像 maven 的 gav 一样，由*命名空间*、*名称*和*版本*三者决定。就算名称和版本完全一样，但命名空间不一样，那么这样的镜像也不是相同的镜像。例如：*aliyun/tomcat:1.3* 和 *tencent/tomcat:1.3*，名称都是 tomcat，版本都是 1.3，但是命名空间不相同，一个是 aliyun 一个是 tencent，那么这两份镜像也是不相同的。当执行 *docker pull tomcat* 时，相当于省略了版本号，那么 docker 引擎会默认设置版本号为 latest。即 *docker pull library/tomcat:latest*。(library 是默认的用户名)

✓ 容器

就像是 Java 中的实例，像对象一样。就好比是由镜像实例化的对象，在 docker 中称之为一个容器实例。

镜像操作

✓ 获取镜像

```
docker pull [选项] [registry]<命名空间/软件名>:<标签>
```

registry 就是前面提到的仓库概念，默认值是 hub.docker.com。

*命名空间*就像 Java 中的包名一样，默认是 library。

*软件名*就是真正使用的软件功能名，比如 tomcat、ubuntu 等等。

*标签*就相当于软件的版本标识，默认是 latest。

```
docker pull tomcat
```

相当于执行了，*docker pull library/tomcat:latest*。如果本地仓库中没有下载过，那么就会从 *registry* 远程仓库中下载并保存到本地仓库，当再次执行 *docker pull tomcat* 时，就不会再次下载了。

```
docker run -it --rm ubuntu:14.04 bash
```

docker 引擎先在本地仓库查找 ubuntu 指定版本镜像，未发现，就会按照 docker pull 的方式下载该镜像。然后，-it 意思是通过交互式终端，--rm 意思是当退出容器时，将容器删除了。最后 bash 时，进入容器后执行的 shell。（*exit* 退出容器）

✓ 查看镜像

查看镜像列表

```
docker images
```

查看镜像、容器、数据卷占用空间详情

```
docker system df
```

查看虚悬镜像(dangling image)

```
docker images -f dangling=true
```

虚悬镜像(dangling image)就是当官方维护镜像，发布了新版本时，镜像被转移了。再次 docker pull 时，旧的镜像就失效了变成了<none>。因此，通常虚悬镜像(dangling image)就没有实际意义了，可删除。

删除虚悬镜像(dangling image)

```
docker rmi $(docker images -q -f dangling=true);
```

-q：只返回镜像的 id。

-f：其中 f 意为 filter，过滤的意思。

在 docker 1.13+版本中，可以使用 *docker image prune* 删除虚悬镜像。

显示中间层镜像

```
docker images -a
```

默认，只显示顶层镜像。通过-a 参数，将显示出中间层镜像。

列出指定镜像

```
docker images 镜像名
```

例如：*docker images ubuntu*。只显示 Ubuntu 的镜像。也可使用*-f* 来过滤。

Dockerfile 定制镜像

✓ 命令格式

```
docker build [选项] <上下文路径/URL/->
```

✓ 简单示例

```
vim Dockerfile
```

Dockerfile 内容

```
FROM nginx
```

```
RUN echo ‘<h1>Hello, dockers.</h1>’ > /usr/share/nginx/html/index.html
```

build 完成

```
docker build -t nginx:4.0 .
```

运行镜像

```
docker run --name mynginx -it -d -p 80:80 nginx
```

✓ 镜像构建上下文

在构建时，命令行中最后一个字符有个点字符（.）。它代表上下文路径。例如在执行 *COPY ./package.json /app/* 命令时，*./package.json* 代表在上下文路径中找到 *package.json* 文件。可以在上下文路径中定义一个 *.dockerignore* 的文件，与 git 的 *.gitignore* 文件类似。

默认构建时，将会在构建上下文中查找 Dockerfile 文件，所以不用显示指定。当然，也可以使用 *-f* 参数指定。例如：*-f ../nginx-docker.file*。

容器操作

✓ 启动

启动容器有两种情形

1．基于镜像创建并启动容器

```
docker run ...
```

2．启动已停止的容器

```
docker start ...
```

✓ 后台运行

```
docker run --name nginx -d -p 80:80 nginx
```

-d：代表在后台运行。

✓ 终止容器

```
docker stop ...
```

当执行 *docker restart* 时，先停止，再启动。

✓ 进入容器

```
docker attach container-name
```

✓ 导入和导出容器

导出容器

```
docker export containerId > customName.tar
```

导入容器

```
cat customName.tar | docker import - test/ubuntu:v1.0
```

✓ 删除容器

```
docker rm ...
```

可以用 *docker rm $(docker ps -a -q)*，来删除所有的容器。