镜像介绍

镜像种类

1. 系统级镜像:如Ubuntu镜像，CentOS镜像以及Debian镜像等；
2. 工具栈镜像:如Golang镜像，Flask镜像，Tomcat镜像等；
3. 服务级镜像:如MySQL镜像，MongoDB镜像，RabbitMQ镜像等；
4. 应用级镜像:如WordPress镜像，DockerRegistry镜像等。

镜像与Dockerfile

Dockerfile中的每一条指令都会对应于Docker镜像中的一层。

镜像与文件系统

Docker镜像的大小并不等于容器中文件系统内容的大小（不包括挂载文件，/proc、/sys等虚拟文件），即镜像大小和容器大小有着本质的区别

镜像共享关系

多个不同的Docker镜像可以共享相同的镜像层，举例说明：

如果平均每个镜像500MB，岂不是100个镜像就需要准备50GB的存储空间？

结果往往不是我们想象的那样，假设docker build构建出来的镜像名分别为image1和image2，由于两个Dockerfile均基于ubuntu:14.04，因此，image1和image2这两个镜像均复用了镜像ubuntu:14.04，本地三个镜像的大小关系应该如下：

ubuntu:14.04: 200MB

image1:200MB(ubuntu:14.04)+20MB=220MB

image2:200MB(ubuntu:14.04)+100MB=300MB

如果仅仅是单纯的累加三个镜像的大小，那结果应该是：200+220+300=720MB，但是由于镜像复用的存在，实际占用的磁盘空间大小是：200＋20+100=320MB，足足节省了400MB的磁盘空间。

镜像的制作

镜像包的制作有两种方法，一种是在容器内部修改后，直接通过 docker commit 命令来生成新的镜像包，另一种是通过编写 Dockerfile 来描述镜像包的构建过程，并通过docker build 命令来生成，这里对这两种方法做一个介绍

用commit命令制作镜像

1. 运行ubuntu容器，并执行bash，-i 参数保持输入打开， -t 分配一个伪终端

```
[root@localhost ~]# docker run -i -t ubuntu bash
root@3b102b069e4a:/#
```

2. 宿主机上运行拷贝宿主机jdk目录到容器。3b102b069e4a 为容器ID

```
[root@localhost jvm]# docker cp java-1.8.0-openjdk-1.8.0.141-
1.b16.el7_3.x86_64/ 3b102b069e4a:/usr/local/
```

3. 配置环境变量

```
root@3b102b069e4a:/usr/local# vi /etc/profile

export JAVA_HOME=/usr/local/java-1.8.0-openjdk-1.8.0.141-
1.b16.el7_3.x86_64
export PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

4. 退出docker容器

```
root@3b102b069e4a:/usr/local# exit
```

5. 查看刚才运行的容器

```
[root@localhost ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS                          PORTS
NAMES
3b102b069e4a        ubuntu              "bash"                     24
minutes ago      Exited (130) 5 seconds ago
confident_lumiere
```

6. 查看不同，3b102b069e4a为容器ID。

```
[root@localhost ~]# docker diff 3b102b069e4a
C /root
A /root/.bash_history
C /var
C /var/lib
```

```
C /var/lib/apt
C /var/lib/apt/lists
A /var/lib/apt/lists/lock
A /var/lib/apt/lists/partial
C /usr
C /usr/local
A /usr/local/java-1.8.0-openjdk-1.8.0.141-1.b16.el7_3.x86_64
A /usr/local/java-1.8.0-openjdk-1.8.0.141-1.b16.el7_3.x86_64/jre
A /usr/local/java-1.8.0-openjdk-1.8.0.141-1.b16.el7_3.x86_64/jre/bin
...
```

7. 创建镜像

```
[root@localhost hello]# docker commit -m "add jdk8" 3b102b069e4a
sha256:4f9bab60d59d134bd9b7d0b19f527b07d4f81428c221c2e623ebfbafde80fce0
```

注：仓库名和TAG没有填写

8. 查看是否生成成功

```
    [root@localhost hello]# docker images
    REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
    <none>              <none>              4f9bab60d59d        9 seconds ago
250MB
    ubuntu              latest              14f60031763d        2 weeks ago
120MB
```

用Dockerfile制作镜像

Dockfile是一种被Docker程序解释的脚本，Dockerfile由一条一条的指令组成，每条指令对应Linux下面的一条命令。Docker程序将这些Dockerfile指令翻译真正的Linux命令。

Dockerfile有自己书写格式和支持的命令，Docker程序解决这些命令间的依赖关系，类似于Makefile。Docker程序将读取Dockerfile，根据指令生成定制的image。

相比image这种黑盒子，Dockerfile这种显而易见的脚本更容易被使用者接受，它明确的表明image是怎么产生的。有了Dockerfile，当我们需要定制自己额外的需求时，只需在Dockerfile上添加或者修改指令，重新生成image即可，省去了敲命令的麻烦。

1. 创建Dockerfile文件

此处仅为示例，指令的详细用法请参考官方文档。

```
    # 指定基于的基础镜像
```

```
    FROM ubuntu:13.10

    # 维护者信息
    MAINTAINER sample "sample@sample.com"

    # 镜像的指令操作
    # 获取APT更新的资源列表
    RUN echo "deb http://archive.ubuntu.com/ubuntu precise main universe">
/etc/apt/sources.list
    # 更新软件
    RUN apt-get update

    # Install curl
    RUN apt-get -y install curl

    # Install JDK 7
    RUN cd /tmp &&  curl -L 'http://download.oracle.com/otn-pub/java/jdk/7u65-
b17/jdk-7u65-linux-x64.tar.gz' -H 'Cookie: oraclelicense=accept-securebackup-
cookie; gpw_e24=Dockerfile' | tar -xz
    RUN mkdir -p /usr/lib/jvm
    RUN mv /tmp/jdk1.7.0_65/ /usr/lib/jvm/java-7-oracle/

    # Set Oracle JDK 7 as default Java
    RUN update-alternatives --install /usr/bin/java java /usr/lib/jvm/java-7-
oracle/bin/java 300
    RUN update-alternatives --install /usr/bin/javac javac /usr/lib/jvm/java-
7-oracle/bin/javac 300

    # 设置系统环境
    ENV JAVA_HOME /usr/lib/jvm/java-7-oracle/

    # Install tomcat7
    RUN cd /tmp && curl -L 'http://archive.apache.org/dist/tomcat/tomcat-
7/v7.0.8/bin/apache-tomcat-7.0.8.tar.gz' | tar -xz
    RUN mv /tmp/apache-tomcat-7.0.8/ /opt/tomcat7/

    ENV CATALINA_HOME /opt/tomcat7
    ENV PATH $PATH:$CATALINA_HOME/bin

    # 复件tomcat7.sh到容器中的目录
    ADD tomcat7.sh /etc/init.d/tomcat7
    RUN chmod 755 /etc/init.d/tomcat7

    # Expose ports.  指定暴露的端口
    EXPOSE 8080

    # Define default command.
    ENTRYPOINT service tomcat7 start && tail -f /opt/tomcat7/logs/catalina.out
```

示例说明如下：

1. 新建一个名为 Dockerfile 的文件。
2. 在 Docker 文件里描述镜像的构建过程，创建一个ubuntu+tomcat的镜像
3. 主要命令说明

   FROM： 必不可少的命令，从某个镜像作为基。如 FROM <image_name> ，或者 FROM <image_name>:. 如果不加tag，默认为latest。先从本地镜像仓库去搜索基镜像，如过本地没有，在去网上docker registry去寻找。

   MAINTAINER：标明该Dockerfile作者及联系方式，可忽略不写

   RUN：建立新的镜像时，可以执行在系统里的命令，如安装特定的软件以及设置环境变量。

   ENV：设置系统环境变量（注意：写在/etc/profile里的命令在dockerfile这里会不生效，所以为改成ENV的方式）

   EXPOSE：开放容器内的端口，但不和宿主机进行映射。方便在宿主机上进行开发测试。（如需映射到宿主机端口，可在运行容器时使用 -p host_port:container_port）

   CMD：设置执行的命令，经常用于容器启动时指定的某个操作。如执行自定义脚本服务，或者是执行系统命令。CMD 只能存在一条，如在Dockerfile中有多条CMD的话，只有最后一条CMD生效！

   （上述 FROM， RUN， VOLUME， EXPOSE， CMD 等命令都是 Dockerfile 文件的基本语法元素，可以查看具体官方文档了解使用方式。）

3. 构建镜像文件

通过 docker build 命令来制作镜像文件。 -t 选项为镜像取名， .则表示 Docker 文件在当前目录中。

```
[root@localhost dockerfile]# docker build -t ubuntu_tomcat_1 .
```

4. 注意事项

1. 使用.dockerignore文件

   为了在docker build过程中更快上传和更加高效，应该使用一个.dockerignore文件用来排除构建镜像时不需要的文件或目录。例如,除非.git在构建过程中需要用到，否则你应该将它添加到.dockerignore文件中，这样可以节省很多时间。

2. 避免安装不必要的软件包

   为了降低复杂性、依赖性、文件大小以及构建时间，应该避免安装额外的或不必要的包。例如，不需要在一个数据库镜像中安装一个文本编辑器。

3. 每个容器都跑一个进程

   在大多数情况下，一个容器应该只单独跑一个程序。解耦应用到多个容器使其更容易横向扩展和重用。如果一个服务依赖另外一个服务，可以参考 Linking Containers Together。

4. 最小化层

   我们知道每执行一个指令，都会有一次镜像的提交，镜像是分层的结构，对于 Dockerfile，应该找到可读性和最小化层之间的平衡。

5. 多行参数排序

   如果可能，通过字母顺序来排序，这样可以避免安装包的重复并且更容易更新列表，另外可读性也会更强，添加一个空行使用 \ 换行:

   ```
   RUN apt-get update && apt-get install -y \
   bzr \
   cvs \
   git \
   mercurial \
   subversion
   ```

6. 创建缓存

   镜像构建过程中会按照 Dockerfile 的顺序依次执行，每执行一次指令 Docker 会寻找是否有存在的镜像缓存可复用，如果没有则创建新的镜像。如果不想使用缓存，则可以在docker build 时添加--no-cache=true选项。

   从基础镜像开始就已经在缓存中了，下一个指令会对比所有的子镜像寻找是否执行相同的指令，如果没有则缓存失效。在大多数情况下只对比 Dockerfile 指令和子镜像就足够了。ADD 和 COPY 指令除外，执行 ADD 和 COPY 时存放到镜像的文件也是需要检查的，完成一个文件的校验之后再利用这个校验在缓存中查找，如果检测的文件改变则缓存失效。RUN apt-get -y update命令只检查命令是否匹配，如果匹配就不会再执行更新了。

   为了有效地利用缓存，你需要保持你的Dockerfile一致，并且尽量在末尾修改。

小结

镜像的制作，特别是怎么通过Dockerfile制作出一个最小化的镜像文件，细节要求比较高，后续会通过结合我们的版本具体实施。