实验地址：

<https://developer.aliyun.com/adc/scenario/813d5d6997b542ed8cbdf040b91aeb66?spm=a2c6h.13858378.0.0.271524adu4nC82>

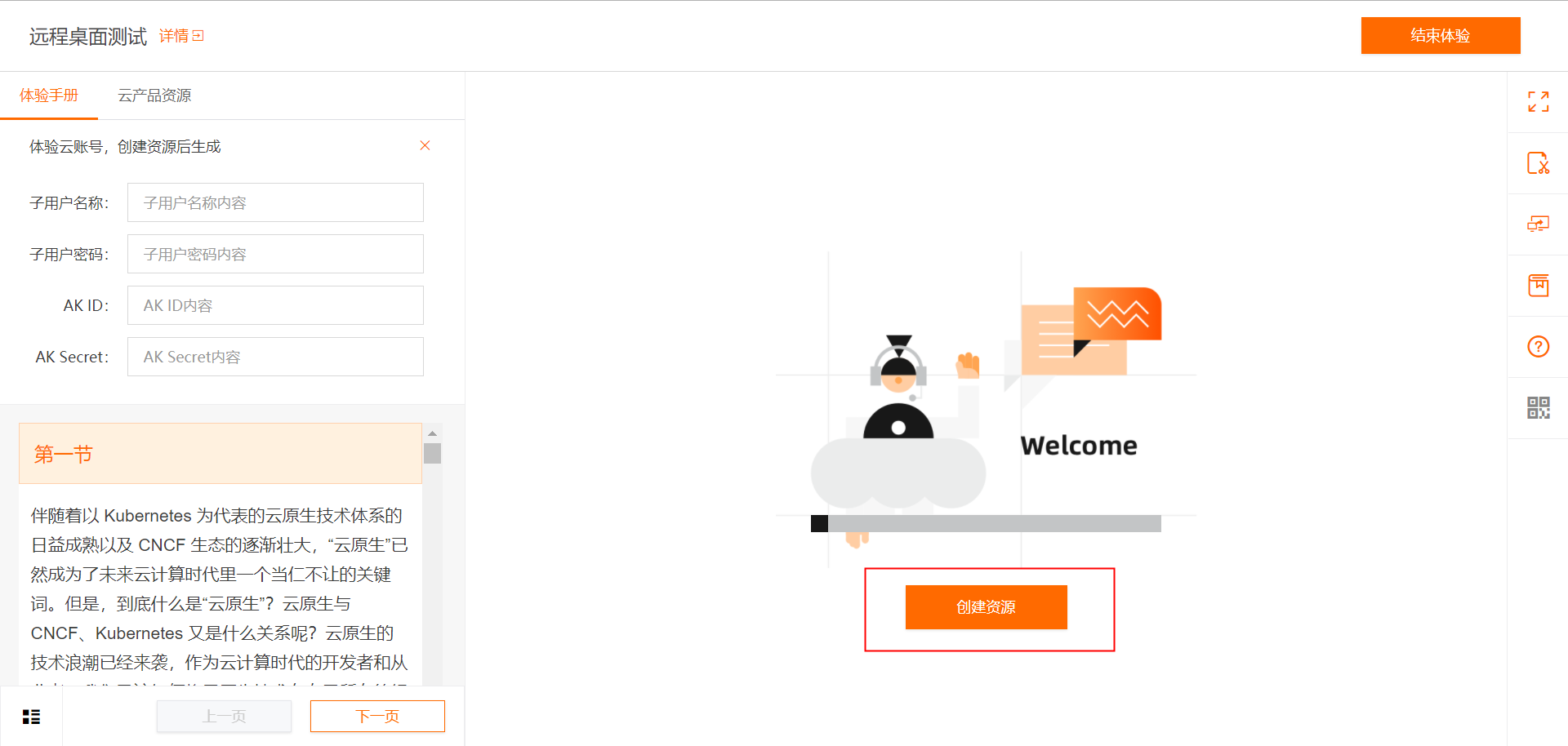
1. 创建资源

开始实验之前，您需要先创建实验相关资源。

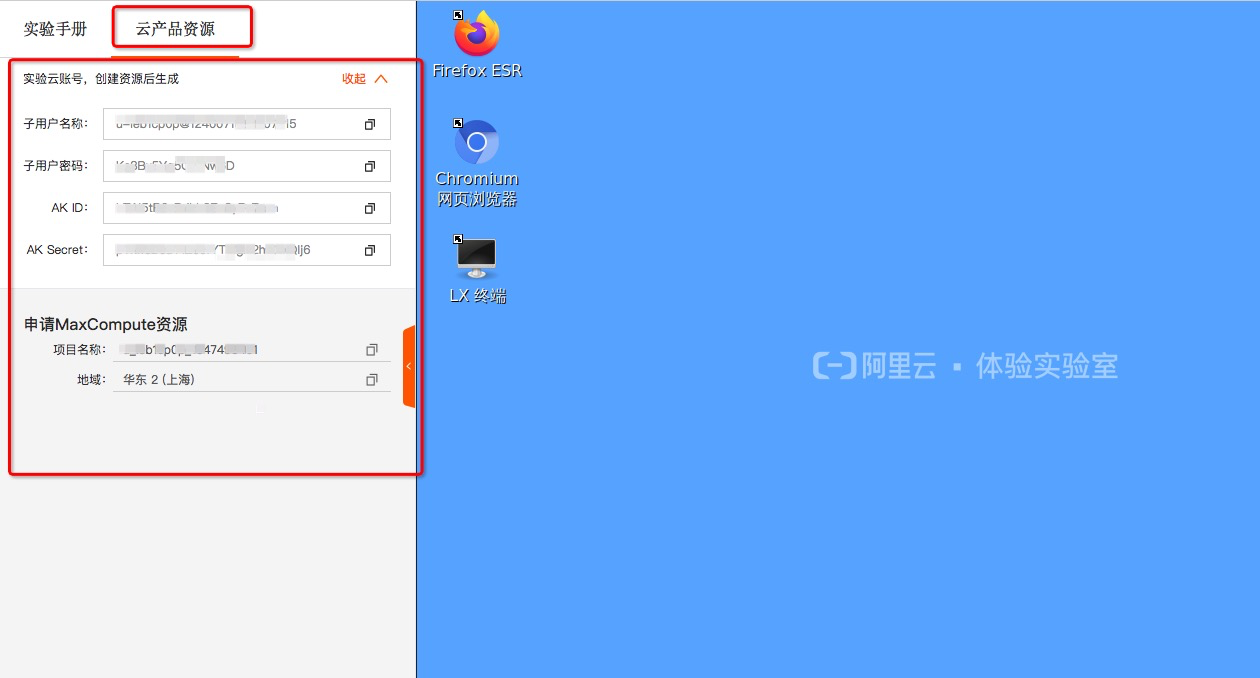
1. 在实验室页面，单击**创建资源**。
2. （可选）在实验室页面左侧导航栏中，单击**云产品资源**列表，可查看本次实验资源相关信息（例如IP地址、子用户信息等）。

**说明：**资源创建过程需要3～5分钟（视资源不同开通时间有所差异，ACK等资源开通时间较长）。完成实验资源的创建后，您可以在**云产品资源**列表查看已创建的资源信息，例如：子用户名称、子用户密码、AK ID、AK Secret、资源中的项目名称等。

实验环境一旦开始创建则进入计时阶段，建议学员先基本了解实验具体的步骤、目的，真正开始做实验时再进行创建。



资源创建成功，可在左侧的资源卡片中查看相关资源信息以及RAM子账号信息



2. docker build快速上手

前面的实验中我们学习了如何通过docker commit命令将一个编辑好的容器，生成一个新的镜像。但是这种方法生成的容器可以查看的只有文件层面的变更内容，容器的使用者往往会搞不清楚在容器制作过程中执行了什么命令，按什么顺序执行了这些命令或者操作。因此除了这种方法之外，Docker还提供了另一种docker build的方式来构建容器。

1. 制作Dockerfile文件

要使用docker build的方式制作容器，我们需要先制作Dockerfile文件。Dockerfile是纯文本文件，我们可以使用vi等纯文本文件编辑工具进行编写。接下来我们使用vi命令生成Dockerfile。注意使用vi编辑器时：

1. 需要先按i键进入编辑模式。
2. 编辑完成之后按esc退出编辑模式。
3. 然后按大写的ZZ保存并退出vim。

vi Dockerfile

然后将下面的内容复制到Dockerfile中：

FROM ubuntu:latest

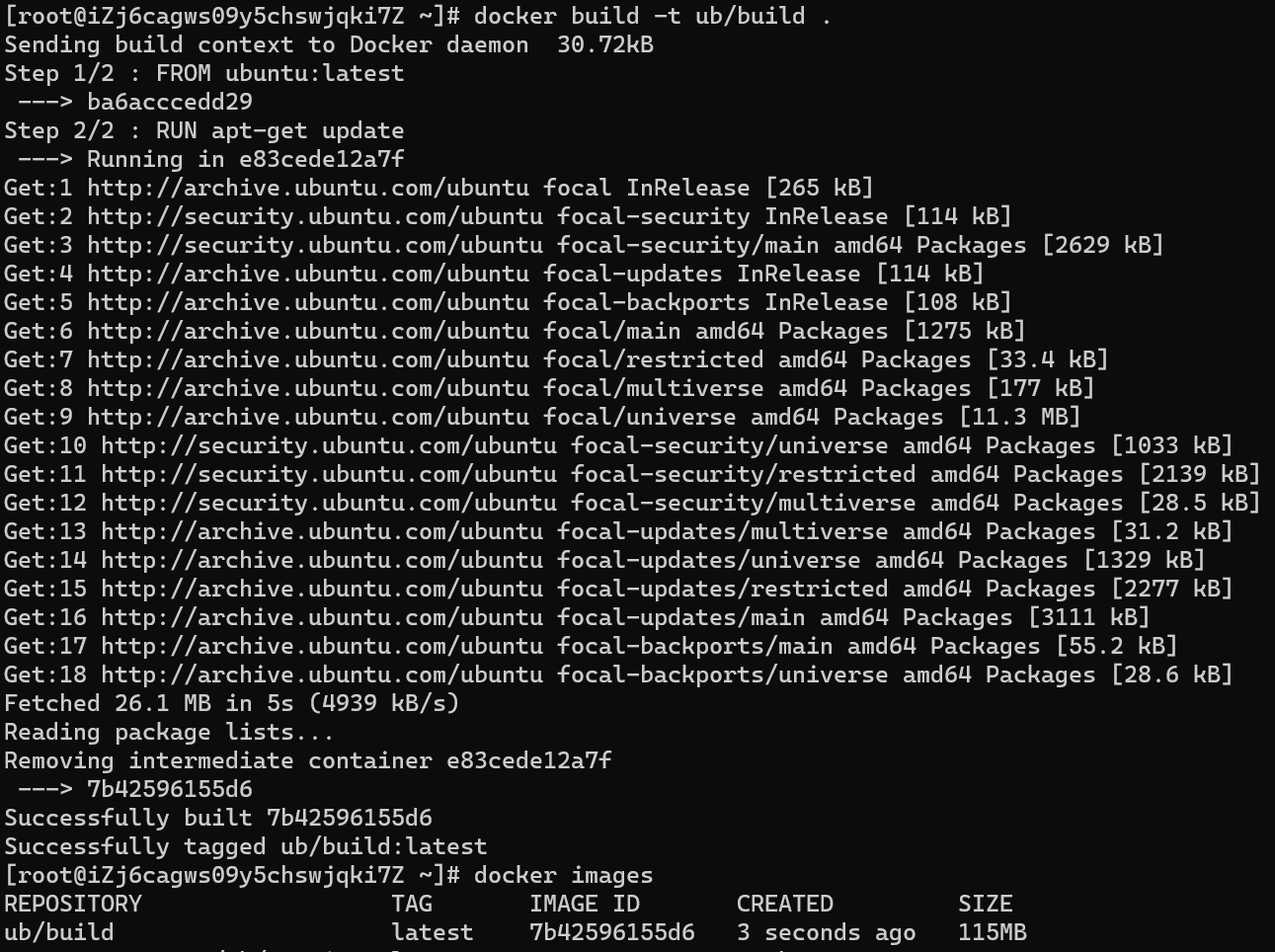
RUN apt-get update

1. 制作容器镜像

Dockerfile文件编写完毕后，我们就就可以根据该文件，使用docker build命令来制作容器镜像，该命令的格式为docker build -t 容器镜像名 Dockerfile所在路径。命令的第三个参数用于指定Dockerfile文明的位置，如果Dockerfile文件就在控制台的当前目录下，一般使用.来设置。

docker build -t ub/build .

docker images



1. Dockerfile命令列表

Dockerfile文件的格式于脚本文件，每一行是一条命令，每行的格式为命令 参数。Dockerfile主要支持的命令如下表，因篇幅的原因，本实验中我们无法具体演示每一条指令的用法，只会讲解一些重要的命令，。

|  |  |
| --- | --- |
| FROM | 指定基础镜像，必须为第一个命令 |
| LABEL | 为镜像添加标签 |
| RUN | 构建时执行的命令行指令 |
| ADD | 将本地文件添加到容器中，如果为tar类型文件，则会自动解压，可以访问网络资源，类似wget（网络资源不会被解压） |
| COPY | 将本地文件添加到容器中，不会解压，也不可以访问网络资源 |
| CMD | 容器启动时执行的命令，只有最后一条可以生效，可以被docker run的启动命令覆盖。 |
| ENTRYPOINT | 容器启动时执行的入口，只有最后一条可以生效，一定会被执行，不可以被覆盖。 |
| EXPOSE | 设置默认开放的网络端口（后面的实验会涉及到） |
| VOLUME | 设置默认加载的VOLUME卷（后面的实验会涉及到） |
| ENV | 设置环境变量。 |
| WORKDIR | 设置默认的工作目录。 |
| USER | 设置容器运行时的用户。 |
| ONBUILD | 构建触发器，当此镜像被其他镜像用作基础镜像时，触发器会被执行。 |
| ARG | 设置构建参数，可以通过docker build --build-arg将参数从外部传到到Dockerfile构建过程中。 |

3. Dockerfile命令详解1

在上一个小节中我们讲解了Dockerfile文件的格式，以及docker build命令的基本用法。在执行docker build时，docker会先根据Dockerfile文件生成并按顺序操作容器。然后再根据按顺序操作好的容器生成镜像。并且在新的容器镜像中保存了容器的操作顺序。接下来我们来介绍Dockerfile文件中的常用的FROM，RUN，WORKDIR，ADD四个命令命令。

1. 命令讲解和环境准备

这两个命令是最常见的Dockerfile命令。一般来讲我们不会从头创建一个镜像，而是会在已有镜像的基础上添加新的内容。这种情况下我们就需要使用FROM命令来指定基础镜像。

在指定基础镜像之后，我们可以使用RUN命令在基础镜像之上执行一些命令。需要注意的是这些命令都是在基础镜像而不是宿主机中运行，同时每一条命令需要使用一个RUN命令。

除了可以在容器中执行命令之外，我们还可以使用ADD命令从宿主机向容器中复制文件。使该命令复制文件时，默认只能复制Dockerfile文件所在目录中或者其子目录中的文件。

如果需要修改需要复制文件的目标路径，我们可以在Dockerfile中使用WORKDIR命令设置容器中的目标路径。

Docker中ADD命令除了可以从宿主机中复制文件，还可以从网络上下载文件到本地。使用该命令远程下载除了支持HTTP/S协议外，也支持GIT等协议。

ADD命令的另一个非常使用的功能是，当复制源文件的扩展名是.tar.gz格式时，该命令会将压缩包解压缩并复制到容器当中。

接下来我们先来进行本小节的实验环境准备。我们创建目录并保存需要上传到镜像中的文件。

mkdir dir6-1

cd dir6-1

echo 本地文件 > info.txt

echo 压缩文件 > tar.txt

tar zcvf info.tar.gz tar.txt

1. 构建容器镜像

接下来我们使用vi命令在dir6-1目录下创建Dockerfile文件，并编辑为如下内容。

vi Dockerfile

FROM ubuntu:latest

WORKDIR data

RUN echo 容器中生成的文件 > img.txt

ADD info.txt info.txt

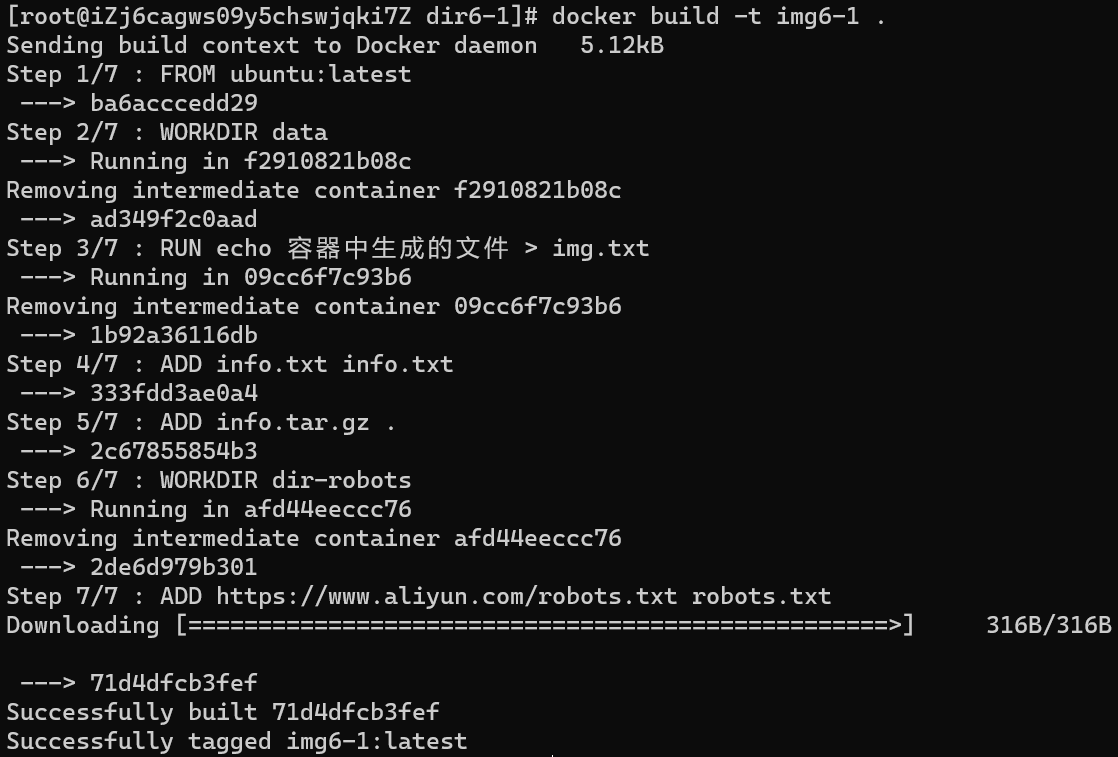
ADD info.tar.gz .

WORKDIR dir-robots

ADD https://www.aliyun.com/robots.txt robots.txt

然后使用docker build编译容器镜像

docker build -t img6-1 .



1. 验证容器镜像

接下来我们来验证编译好的容器镜像，我们通过docker run启动镜像，然后验证容器中的内容。我们发现容器在运行之后，默认的目录为/data/dir-robots这说明WORKDIR命令不但会改变镜像生成时的操作目录，也会影像到容器运行时的默认目录。

接下来我们看到了从www.aliyun.com/robots.txt地址下载的同名文件。验证了ADD命令的网络下载功能。

docker run -itd --name container6-1 img6-1

docker exec container6-1 pwd

docker exec container6-1 cat robots.txt

文本

描述已自动生成

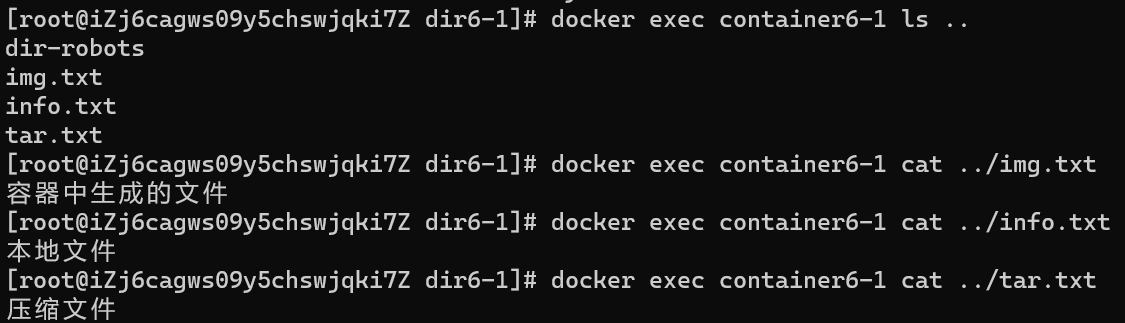
我们继续验证其他目录的文件。这里我们使用了docker exec container6-1 ls和docker exec container6-1 cat命令。来查看文件列表和文件内容。会发现目录中的3个文件和内容符合期望

docker exec container6-1 ls ..

docker exec container6-1 cat ../img.txt

docker exec container6-1 cat ../info.txt

docker exec container6-1 cat ../tar.txt



4. Dockerfile命令详解2

接下来的这个小节我们再来看另外的三个Dockerfile命令，分别是ENV，CMD，ENTRYPOINT。

1. ENV和CMD命令讲解

Dockfile中的ENV命令的功能相对比较简单。通过该命令我们可以为容器镜像设置环境变量。通过环境变量我们可以将一些配置信息固化在镜像中。

接下来再来看CMD命令，这个命令用来设置容器的初始化命令。通过此命令我们可以让容器在启动时候执行一条命令，通过这条命令我们可以实现容器启动后，运行一个服务，或者在控制台输出一段文字等功能。此命令的格式为CMD [”参数1“, ”参数2“...]。

在使用CMD命令是我们需要注意的几点是：a. 这条命令虽然可以设置很多次，但是之后最后一次会生效。b. 当我们使用docker run命令启动容器时，我们可以docker run 容器镜像名后面加入参数1 参数2..的形式 代替容器镜像中原有的CMD命令。

我们创建子目录，并使用vi修改Dockerfile为如下内容。并编辑为如下内容。

cd /

mkdir dir6-2

vi dir6-2/Dockerfile

FROM ubuntu:latest

ENV IMG\_STRING img6-2的环境变量

CMD ["echo", "$IMG\_STRING"]

Dockerfile编写完毕后，使用docker build进行编译。

docker build -t img6-2 dir6-2

1. CMD和ENTRYPOINT命令讲解

除了CMD命令之外我们还可以使用ENTRYPOINT命令来实现类似的功能。该命令和CMD命令的格式和功能基本一致，其区别在于docker run命令只能使用--entrypoint参数替换镜像中的ENTRYPOINT设置。

在编写Dockerfile时，如果ENTRYPOINT和CMD命令同时出现的时候，容器启动时会将两个指令的参数串联起来，以ENTRYPOINT参数1, ENTRYPOINT参数2..., CMD参数1, CMD参数2...的形式执行启动命令。因此在具体使用时，我们一般在ENTRYPOINT中设置初始化命令，在CMD中设置初始化命令的参数。需要注意的是，ENTRYPOINT和CMD命令联合使用的时候，只能使用[”参数1“, ”参数2“...]格式的参数。

我们创建子目录，并使用vi修改Dockerfile为如下内容。并编辑为如下内容。

cd /

mkdir dir6-3

vi dir6-3/Dockerfile

FROM ubuntu:latest

CMD ["-c", "echo CMD作为ENTRYPOINT的参数"]

ENTRYPOINT ["bash"]

Dockerfile编写完毕后，然后使用docker build编译容器镜像

docker build -t img6-3 dir6-3

1. 验证容器镜像

我们来验证编译好的img6-2容器镜像，我们通过docker run启动镜像，然后验证容器中的内容。

docker run img6-2

docker run img6-2 ls -ll

图形用户界面

低可信度描述已自动生成

接下来我们来验证编译好的img6-3容器镜像，我们通过docker run启动镜像，然后验证容器中的内容。

docker run img6-3

docker run --entrypoint echo img6-3 "手动设置ENTRYPOINT和CMD"

docker run img6-3 -c "echo 手动设置CMD参数"

文本

描述已自动生成

5. 容器镜像的层次关系

在上一个实验中我们验证了通过docker commit命令生成镜像时，会在原有的镜像层之上添加一个新的层。接下来我们来看一下通过docker build方式生成的镜像的层次结构。

1. 查看镜像层次

我们也用docker inspect命令来查看一下镜像img6-1的层次信息，结果会发现docker build在基础镜像之上构建多个了新的镜像。

docker inspect -f "{{json .RootFS.Layers}}" img6-1 | jq

文本

描述已自动生成

1. 查看镜像历史

通过docker build生成的镜像。除了docker inspcet之外，还可以通过docker history命令来查看通过Dockerfile定义的镜像的生成方式。我们可以看到docker history命令输出了镜像构建的过程信息，通过这一信息我们能比较清晰的看到镜像作者在制作镜像时的具体操作。

docker history img6-1

文本

描述已自动生成

1. 镜像层次和镜像历史之间的关系

当我们执行上两个步骤的时候，细心的同学可能会发现，docker history中的每一个步骤都会生成一个中间状态的镜像，但是镜像的层数和镜像的步骤并不是严格的一一对应关系。这是因为docker会自动压缩一些没有实际产生数据修改的镜像，将多个临时镜像压缩成一层。比如WORKDIR命令所生成的中间状态。

同样我们可以比较img6-2和img6-3镜像。我们会发现虽然这两个镜像包含了很多的步骤，但是由于没有实际向镜像中写入文件，因此这两个镜像的存储层实际上和基础镜像ubuntu的存储层保持一致，这就以为着这两个镜像并没有在看本地镜像仓库中额外消耗存储控件。

docker history img6-2

docker inspect -f "{{json .RootFS.Layers}}" img6-2 | jq

docker history img6-3

docker inspect -f "{{json .RootFS.Layers}}" img6-3 | jq

docker inspect -f "{{json .RootFS.Layers}}" ubuntu | jq

图形用户界面, 文本, 应用程序

描述已自动生成