Dockerfile CMD与ENTRYPOINT

nak21 关注 IP属地: 浙江

2022.03.25 18:49:49 字数 2,253 阅读 2,385

简介

在查看Dockerfile可用指令（instructions）时，会发看起来有一些“重复”指令（即不同指令实现的功能几乎相同）。之前我们讲解了COPY和ADD的区别，本章会分析CMD与ENTRTYPOINT的不同。

ENTRYPOINT与CMD都可以对iamge配置启动命令。但两者之间还是有一些细微的区别。多数情况下需要用户在二者中选择其一使用，但也可以共同使用两者。下面将具体分析二者不同的使用场景：

ENTRYPOINT or CMD

最终，ENTRYPOINT与CMD都提供了一个方法，让用户指定容器默认启动命令。事实上，如果你希望你的image是可执行的（启动docker run时不额外指定启动命令就可以运行），那么你必须在Dockerfile中使用ENTRYPOINT或CMD

尝试运行一个没有使用ENTRYPOINT或CMD指令的image，启动时会报错：

```
$ docker run alpine
FATA[0000] Error response from daemon: No command specified
```

你能在Docker Hub上找到的大多数linux版本基础镜像都使用了`/bin/sh`或`/bin/bash`这样的shell命令来作为CMD启动命令。这意味着，任何人运行这些image时，都会默认进入到交互式shell界面中（假设运行时指定了-t/-i参数）

这对通用的基础镜像是十分方便的，但是当你希望运行自己的image时（即非通用基础iamge时），更多时候需要指定一个更具体的可执行文件或命令来作为CMD或ENTRYPOINT参数。

Overrides

在Dockerfile中指定的ENTRYPOINT或CMD为你的image指定默认启动命令。并且，用户可以选择在容器运行时重写（overrides）这些值中的任何一个。

例如，假设我们有以下Dockerfile：

```
FROM ubuntu:trusty
CMD ping localhost
```

如果我们构建该image，在运行是会看到如下输出：

```
$ docker run -t demo
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
^C
```

```
--- localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.026/0.032/0.039/0.008 ms
```

你可以看到，在容器启动时，自动运行了*ping*命令。然而，在启动容器时，我们可以在image名称后面添加一个参数来重写默认CMD指令:

```
$ docker run demo hostname
6c1573c0d4c0
```

在上述例子中，*hostname*命令代替了*ping*命令运行

默认ENTRYPOINT指令也可以被类似的方式重写，不过需要使用--entrypoint参数

```
$ docker run --entrypoint hostname demo
```

CMD的使用场景

考虑到重写CMD指令是非常简单的，所以在希望用户使用该image创建容器时拥有更高的灵活性，可以更方便的指定任何自己想要的启动命令时，更推荐CMD指令。例如，你有一个通用的Ruby Image，默认情况下将启动一个交互式的irb会话(CMD irb)，但你也想给用户一个选项来运行任意的Ruby脚本(docker运行Ruby Ruby -e 'puts "Hello"')。

ENTRYPOINT的使用场景

相反，ENTRYPOINT指令适合用于：希望容器最终运行时所执行的命令与Dockerfile内配置的命令相同的场景下。也就是说，不希望用户重写image启动命令

通常使用Docker作为指定可执行文件的容器是很方便的。假设您有一个Python脚本的实用程序，您需要发布它，但又不想让安装正确的解释器版本和依赖项给最终用户带来负担。你可以配置好解释器版本与依赖后，通过ENTRYPOINT指定可执行文件。用户便可以使用docker运行你的image，它的行为就像直接运行你的脚本，但又不用考虑依赖项、启动命令参数等信息，直接运行即可。

当然，使用CMD指令可以实现同样的功能，但使用了ENTRYPOINT相当于给用户传递了一个**强烈的信息**：这个容器**只为运行这一个程序而存在**，尽量不要修改容器启动命令来另作他用。

将ENTRYPOINT与CMD组合使用时，ENTRYPOINT的效用将会更清楚，但我们在后文讨论这种用法。

Shell vs. Exec

ENTRYPOINT与CMD指令都支持两种不同的参数格式：*Shell格式*与*Exec格式*，在上面的例子中，我们使用了*shell格式*:

```
1 CMD executable param1 param2
```

Shell

当使用*Shell格式*时，容器启动时会使用`/bin/sh -c`来执行指定的可执行/二进制/文件。容器启动后，运行`docker ps`就可以清楚看到：

```
$ docker run -d demo
15bfcddb11b5cde0e230246f45ba6eeb1e6f56edb38a91626ab9c478408cb615

$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED
15bfcddb4312 demo:latest "/bin/sh -c 'ping localhost'" 2 seconds ago
```

我们再次运行了“demo”容器，可以看到容器启动命令为：`/bin/sh -c 'ping localhost`

这看起来没什么问题，命令也正常被运行。但是当我们使用shell格式来传递ENTRYPOINT或CMD参数时还是会有一些微妙的小问题。现在我们进入到容器内，查看容器内的进程就会看到如下信息：

```
$ docker exec 15bfcddb ps -f
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 20:14 ? 00:00:00 /bin/sh -c ping localhost
root 9 1 0 20:14 ? 00:00:00 ping localhost
root 49 0 0 20:15 ? 00:00:00 ps -f
```

请注意，PID为1的进程并不是我们所期望的`ping`命令，而是`/bin/sh`。这会导致当我们需要向容器发送任何类型的POSIX信号，`/bin/sh`不会将信号转发给子进程(详细原理可参考： Gracefully Stopping Docker Containers)。

除了PID1的问题外，可能很多轻量化的image并不会包含任何shell程序。当容器启动时，并不会检查容器内是否有shell程序可用。如果你的镜像并不包含`/bin/sh`命令，那么显然容器会启动失败。

Exec

所以更好的选择是使用*Exec格式*来传递ENTRYPOINT与CMD参数，例如：

```
CMD ["executable","param1","param2"]
```

注意，CMD指令后的参数被格式化成了JSON数组

当使用Exec格式的CMD指令后，容器启动时该命令将不会通过Shell的方式运行，而是直接执行。

让我们将上述的Dockerfile改为Exec格式看看实际效果：

```
FROM ubuntu:trusty
CMD ["/bin/ping","localhost"]
```

重新构建image，查看容器启动命令：

```
$ docker build -t demo .
[truncated]

$ docker run -d demo
90cd472887807467d699b55efaf2ee5c4c79eb74ed7849fc4d2dbfea31dce441
```

```
$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED
90cd47288780 demo:latest "/bin/ping localhost" 4 seconds ago
```

可以看到，`ping`命令在没有shell介入的情况下直接运行。并且是容器内的PID1进程

所以无论使用ENTRYPOINT或CMD指令，都推荐使用*Exec格式*。因为它可以使你的应用清晰的运行在容器的PID1进程上。

ENTRYPOINT and CMD

到目前为止，我们讨论了如何使用ENTRYPOINT**或**CMD指令来指定image默认启动命令。然而，在一些情况下，我们可以同时使用ENTRYPOINT**与**CMD。

将ENTRYPOINT与CMD组合使用依旧可以指定image默认启动命令，同时也了指定image启动命令的默认参数。同时该参数可以被方便的重写。让我们看下述例子：

```
FROM ubuntu:trusty
ENTRYPOINT ["/bin/ping","-c","3"]
CMD ["localhost"]
```

重新构建image并不附加任何参数启动容器：

```
$ docker build -t ping .
[truncated]

$ docker run ping
```

```
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.038 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.051 ms

--- localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.025/0.038/0.051/0.010 ms

$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED
82df66a2a9f1 ping:latest "/bin/ping -c 3 localhost" 6 seconds ago
```

请注意，启动命令为Dockerfile中ENTRYPOINT与CMD值的组合。当同时使用ENTRYPOINT与CMD指令时，CMD指令的值会被追加到ENTRYPOINT值的后面，组合成为一条启动命令。且CMD指令仍然保留容易被重写的特性，用户可以很方便的通过在`docker run`后添加参数来重写CMD值（即启动命令的参数）。下例展示了如何修改`ping`命令的默认参数：

```
$ docker run ping docker.io
PING docker.io (162.242.195.84) 56(84) bytes of data.
64 bytes from 162.242.195.84: icmp_seq=1 ttl=61 time=76.7 ms
64 bytes from 162.242.195.84: icmp_seq=2 ttl=61 time=81.5 ms
64 bytes from 162.242.195.84: icmp_seq=3 ttl=61 time=77.8 ms

--- docker.io ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 76.722/78.695/81.533/2.057 ms

$ docker ps -l --no-trunc
CONTAINER ID IMAGE COMMAND CREATED
0d739d5ea4e5 ping:latest "/bin/ping -c 3 docker.io" 51 seconds ago
```

现在，运行image就像运行一个普通可执行文件（命令）一样，指定要执行的可执行文件（image），并在其后指定相应的参数。

请注意，作为ENTRYPOINT的一部分包含的-c 3参数实际上成为了ping命令的"硬编码"参数。它包含在image的每次调用中，重写CMD参数并不会影响ENTRYPOINT中的参数。

Always Exec

当同时使用ENTRYPOINT与CMD，请注意必须使用*Exec格式*来传递参数。你会发现，使用Shell格式或混合使用，将永远不会得到你想要的效果:

```
Dockerfile    Command
ENTRYPOINT /bin/ping -c 3
CMD localhost    /bin/sh -c '/bin/ping -c 3' /bin/sh -c localhost
ENTRYPOINT ["/bin/ping","-c","3"]
CMD localhost    /bin/ping -c 3 /bin/sh -c localhost
ENTRYPOINT /bin/ping -c 3
CMD ["localhost"]    /bin/sh -c '/bin/ping -c 3' localhost
ENTRYPOINT ["/bin/ping","-c","3"]
CMD ["localhost"]    /bin/ping -c 3 localhost
```

上述例子中，只有当同时使用*Exec格式*的ENTRYPOINT与CMD时，才能实现我们想要的功能。

总结

ENTRYPOINT与CMD指令都可以帮助你在Dockerfile中配置容器启动命令。但二者各对应了不同的应用场景，实际使用中还需挑选一个合适的指令。并且二者并不是互斥了，在某些场景下同

时使用两个指令也是必须的。但无论如何使用，请忘记*Shell格式*，在任何情况下*Exec格式*都一定是正确的选择。

> 参考：https://www.ctl.io/developers/blog/post/dockerfile-entrypoint-vs-cmd/

最后编辑于： 2022.03.31 16:37:23