太长不看：如果需要在Dockerfile的ENTRYPONNT中指定运行命令的用户，用gosu代替sudo可以避免某些信号处理上的边界条件。不过这些边界条件比较罕见，就算不用也没多大关系

docker官方文档的Dockerfile部分，有一节讲的是ENTRYPOINT。在[这一节](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/#entrypoint)中，提到了如果在启动脚本中需要指定运行命令的用户，建议用gosu代替sudo，并给出了一个例子：

**#!/bin/bash**

set -e

**if** [ "$1" = 'postgres' ]; **then**

chown -R postgres "$PGDATA"

**if** [ -z "$(ls -A "$PGDATA")" ]; **then**

gosu postgres initdb

**fi**

exec gosu postgres "$@"

**fi**

exec "$@"

上面的脚本中，docker run指定的命令会以postgres用户的身份执行。

所谓的ENTRYPOINT，正如其名，就是该镜像的根命令。默认的ENTRYPOINT为/bin/sh -c，通过docker run或CMD指定的命令会作为ENTRYPOINT的参数执行。举个例子，docker run ubuntu:latest ls就是执行/bin/sh -c ls。有些时候我们需要指定ENTRYPOINT的值，比如换成自己的包装脚本。

默认docker中的命令都是以root身份启动的（因为默认只有root用户）。不过你也可以通过USER指令设置当前使用的用户。某些时候，你可能需要在docker build中使用多个用户，比如上面例子中，安装依赖需要root，运行程序时使用的是postgres。这时候就需要动态指定一个用户身份。

docker文档中建议，如果需要动态指定一个用户身份，需要使用[gosu](https://github.com/tianon/gosu)而非平常的sudo。

然而文档中并没有解释为什么。gosu的[项目主页](https://github.com/tianon/gosu)中也只提到gosu避免了strange and often annoying TTY and signal-forwarding behavior。（然后顺便黑了下sudo太过于复杂）。不过gosu的测试用例透露了些蛛丝马迹，可以看出它认为sudo至少有两点不好：

1. sudo会作为被授权的命令的父进程一直存在，直到该命令退出。
2. sudo模式下的HOME环境变量仍是用sudo者原来的值。

可以实证下这两个指责：

~ sudo ps -o pid,ppid,**cmd**

PID PPID CMD

12599 4281 sudo ps -o pid,ppid,**cmd**

12600 12599 ps -o pid,ppid,cmd

~ sudo **env** | grep HOME

HOME=/home/lzx

这两个现象确实存在，不过会造成什么危害呢？如果真有鬼，夜路走多了自然会碰见。然而平时都是用着sudo，也没遇到什么事呀。

我们先来看看第二点，sudo模式下HOME环境变量保存不变的事情。

这个事情涉及到sudo的应用场景。sudo用于扮演某个用户来执行给定的命令，这一点类似于su。个人认为，sudo跟su第二大不同，在于sudo是对使用者鉴权，而su是对目标权限进行鉴权。假定你是sudoer，运行sudo时你要输入自己的密码，也即证明自己有扮演的权限；而运行su时，你要输入的是要扮演的用户的密码，也即证明你有扮演的那个用户的权限。所以sudo会认为，那你使用sudo只是想临时使用某一身份。既然如此，sudo下HOME环境变量还是原来的样子，也不是什么bug，而是个feature。如果你不认同这个feature，可以使用sudo -H。

再来看看第一点，sudo作为命令的父进程会一直存在。sudo之所以退而不休，是因为它需要监控命令的输入输出。作为一个非常关注安全性的程序，sudo会重置自己的环境变量，尽量以干净的环境来执行命令。不止如此，它还允许用户定义安全策略，来处理命令的输入输出。不过有种情况下，sudo会直接exec给定的命令。那就是当用户没有指定安全策略，且执行的命令不需要占用伪终端的时候。举个例子，sudo sh -c 'sleep 20 &'时，sudo就真的不再作为父进程一直存在了（注意这里我用了个sh来分割整条命令.如果直接输入sudo sleep 20 &，会被解析成后台运行sudo sleep 20）。不过这种情况非常特殊，基本上可以忽略。这一点跟上面那条不同，不存在一个改变该默认行为的选项。

看来所谓的“annoying behavior”就是指这个了。不过平时用的时候从没考虑过这个呀，为什么到了docker里就不建议用呢？  
原因在于docker中处理signal的方式。很多程序，比如Apache和Nginx，允许用户通过发信号的方式来控制程序的生命周期（重启、关闭、停止，等等）。由于docker把进程封装了一层，如果想要给这些程序发信号，直接发给docker进程是不行的。那只会影响docker本身的行为。而且这些程序在docker里面运行时，不可能意识到自己在一个独立的容器里。它们所报告的pid，跟外界的pid是不符合的。  
为了跟UNIX的信号机制和谐相处，docker另外提供了发送信号的接口：docker stop和docker kill。docker stop会发两拨信号，一个是SIGTERM，另一个是SIGKILL。而docker kill则是kill的翻版。这两个命令有个奇怪的地方，就是它们发送信号，从来都只发给所谓的main process进程，也即ENTRYPOINT进程。如果该进程不会转发信号（比如默认的/bin/sh -c），目标进程就收不到信号，这个功能便废了。而当我们用sudo启动某个命令时，最终收到信号的会是sudo进程，而不是那个命令。

那么sudo是否会转发信号？答案是，如果可以的话，sudo会尽可能地转发信号。即使遇到了SIGTERM这样默认行为是终止进程的信号，sudo也不会直接终止，而会转发出去。所以尽管多了个sudo拦在路上，大多数情况下，想要发送给目标进程的信号还是能到达的。但是，SIGSTOP和SIGKILL两个信号是无法捕获的，sudo对此也无能为力。SIGKILL的话情况还好，因为main process进程（这里的sudo）退出后，整个docker进程都会退出，无意中也达到了一样的结果。不过SIGSTOP只会让sudo停下来，结果该停的没停，不该停的却停了。

gosu的实现很简单。它包括以下几个步骤：

* setgroup
* setuid
* setgid
* 设置$HOME
* exec 目标命令

除了最后关键的两步，其它跟sudo差不多。