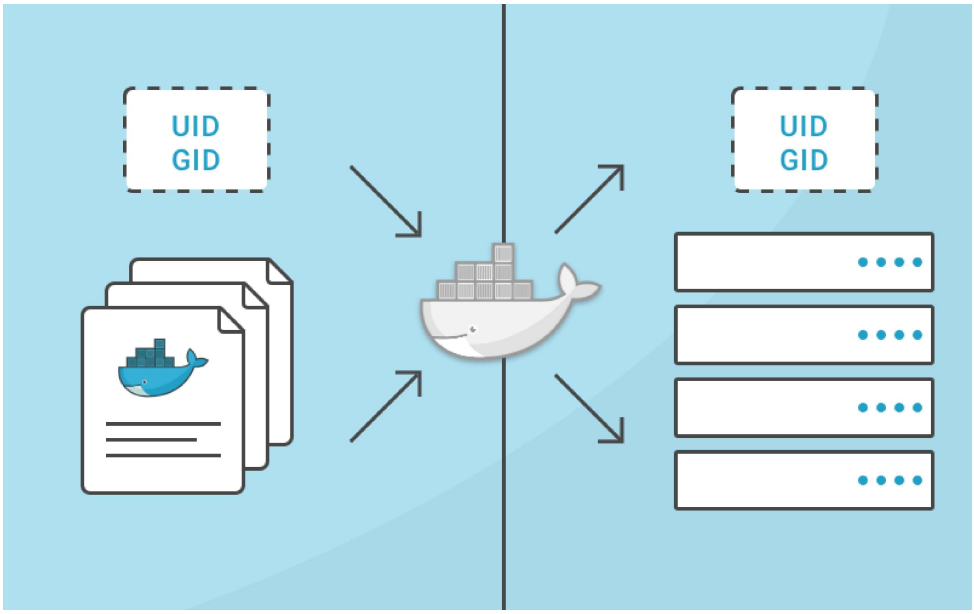
sparkdev

理解 docker 容器中的 uid 和 gid

默认情况下，容器中的进程以 root 用户权限运行，并且这个 root 用户和宿主机中的 root 是同一个用户。听起来是不是很可怕，因为这就意味着一旦容器中的进程有了适当的机会，它就可以控制宿主机上的一切！本文我们将尝试了解用户名、组名、用户 id(uid)和组 id(gid)如何在容器内的进程和主机系统之间映射，这对于系统的安全来说是非常重要的。说明：本文的演示环境为 ubuntu 16.04(下图来自互联网)。



先来了解下 uid 和 gid

uid 和 gid 由 Linux 内核负责管理，并通过内核级别的系统调用来决定是否应该为某个请求授予特权。比如当进程试图写入文件时，内核会检查创建进程的 uid 和 gid，以确定它是否有足够的权限修改文件。注意，**内核使用的是 uid 和 gid，而不是用户名和组名。**
简单起见，本文中剩下的部分只拿 uid 进行举例，系统对待 gid 的方式和 uid 基本相同。

很多同学简单地把 docker 容器理解为轻量的虚拟机，虽然这简化了理解容器技术的难度但是也容易带来很多的误解。事实上，与虚拟机技术不同：同一主机上运行的所有容器共享同一个内核(主机的内核)。容器化带来的巨大价值在于所有这些独立的容器(其实是进程)可以共享一个内核。这意味着即使由成百上千的容器运行在 docker 宿主机上，但**内核控制的 uid 和 gid 则仍然只有一套**。所以同一个 uid 在宿主机和容器中代表的是同一个用户(即便在不同的地方显示了不同的用户名)。
注意，由于普通的用来显示用户名的 Linux 工具并不属于内核(比如 id 等命令)，所以我们可能会看到同一个 uid 在不同的容器中显示为不同的用户名。但是对于相同的 uid 不能有不同的特权，即使在不同的容器中也是如此。

如果你已经了解了 Linux 的 user namespace 技术，参考《Linux Namespace : User》，你需要注意的是到目前为止，docker 默认并没有启用 user namesapce，这也是本文讨论的情况。笔者会在接下来的文章中介绍如何配置 docker 启用 user namespace。

容器中默认使用 root 用户

如果不做相关的设置，容器中的进程默认以 root 用户权限启动，下面的 demo 使用 ubuntu 镜像运行 sleep 程序：

```
$ docker run -d  --name sleepme ubuntu sleep infinity
```

注意上面的命令中并没有使用 sudo。笔者在宿主机中的登录用户是 nick，uid 为 1000：

```
nick@tigger:~$ id
uid=1000(nick) gid=1000(nick) groups=1000(nick),4(adm)
,24(cdrom),27(sudo),30(dip),46(plugdev),113(lpadmin),1
28(sambashare),999(docker)
```

在宿主机中查看 sleep 进程的信息：

```
$ ps aux | grep sleep
```

```
nick@tigger:~$ ps aux | grep sleep
root       41698  0.2  0.0   4532   828 ?        Ss   08:40   0:00 sleep infinity
```

sleep 进程的有效用户名称是 root，也就是说 sleep 进程具有 root 权限。
然后进入容器内部看看，看到的情况和刚才一样，sleep 进程也具有 root 权限：

```
nick@tigger:~$ docker exec -it sleepme bash
root@817bd4d98c42:/# ps aux | grep sleep
root         1  0.0  0.0   4532   800 ?        Ss   00:43   0:00 sleep infinity
```

那么，**容器内的 root 用户和宿主机上的 root 用户是同一个吗?**
答案是：是的，它们对应的是同一个 uid。原因我们在前面已经解释过了：整个系统共享同一个内核，而内核只管理一套 uid 和 gid。

其实我们可以通过数据卷来简单的验证上面的结论。在宿主机上创建一个只有 root 用户可以读写的文件：

```
nick@tigger:~$ ll testv/testfile
-rw------- 1 root root 12 Sep  7 09:01 testv/testfile
nick@tigger:~$ cat testv/testfile
cat: testv/testfile: Permission denied
```

然后挂载到容器中：

```
$ docker run --rm -it -w=/testv -v $(pwd)/testv:/testv ubuntu
```

在容器中可以读写该文件：

```
nick@tigger:~$ docker run --rm -it -w=/testv -v $(pwd)/testv:/testv ubuntu
root@199e72ab8b08:/testv# ll testfile
-rw------- 1 root root 12 Sep  7 01:01 testfile
root@199e72ab8b08:/testv# cat testfile
hello world
```

我们可以通过 Dockerfile 中的 USER 命令或者是 docker run 命令的 --user 参数指定容器中进程的用户身份。下面我们分别来探究这两种情况。

在 Dockerfile 中指定用户身份

我们可以在 Dockerfile 中添加一个用户 appuser，并使用 USER 命令指定以该用户的身份运行程序，Dockerfile 的内容如下：

```
FROM ubuntu
RUN useradd -r -u 1000 -g appuser
USER appuser
ENTRYPOINT ["sleep", "infinity"]
```

编译成名称为 test 的镜像：

```
$ docker build -t test .
```

```
nick@tigger:~/myapp$ docker build -t test .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu
 ---> 113a43faa138
Step 2/4 : RUN useradd -r -u 1000 appuser
 ---> Running in f261759acdbf
Removing intermediate container f261759acdbf
 ---> 0ec728608e72
Step 3/4 : USER appuser
 ---> Running in 8d7fe55f6d89
Removing intermediate container 8d7fe55f6d89
 ---> c087a8b25a90
Step 4/4 : ENTRYPOINT ["sleep", "infinity"]
 ---> Running in 2302f5d499ac
Removing intermediate container 2302f5d499ac
 ---> f0918323aaa2
Successfully built f0918323aaa2
Successfully tagged test:latest
```

用 test 镜像启动一个容器：

```
$ docker run -d --name sleepme test
```

在宿主机中查看 sleep 进程的信息：

```
nick@tigger:~$ ps aux | grep sleep
nick      44680  0.0  0.0   4532   752 ?        Ss   13:35   0:00 sleep infinity
```

这次显示的有效用户是 nick，这是因为在宿主机中，uid 为 1000 的用户的名称为 nick。再进入到容器中看看：

```
$ docker exec -it sleepme bash
```

```
nick@tigger:~$ docker exec -it sleepme bash
appuser@dfa4e6dc35b1:/$ ps aux | grep sleep
appuser       1  0.0  0.0   4532   752 ?        Ss   05:35   0:00 sleep infinity
```

容器中的当前用户就是我们设置的 appuser，如果查看容器中的 /etc/passwd 文件，你会发现 appuser 的 uid 就是 1000，这和宿主机中用户 nick 的 uid 是一样的。

让我们再创建一个只有用户 nick 可以读写的文件：

```
nick@tigger:~/testv$ ll testfile
-rw------- 1 nick nick 12 Sep  7 14:03 testfile
```

同样以数据卷的方式把它挂载到容器中：

```
$ docker run -d --name sleepme -w=/testv -v $(pwd)/testv:/testv test
```

```
nick@tigger:~$ docker exec -it sleepme bash
appuser@6b7c5cb82fa9:/testv$ ls -l testfile
-rw------- 1 appuser 1000 12 Sep  7 06:03 testfile
appuser@6b7c5cb82fa9:/testv$ cat testfile
hello world
```

在容器中 testfile 的所有者居然变成了 appuser，当然 appuser 也就有权限读写该文件。

这里到底发生了什么？而这些又这说明了什么？

首先，宿主机系统中存在一个 uid 为 1000 的用户 nick。其次容器中的程序是以 appuser 的身份运行的，这是由我们通过 USER appuser 命令在 Dockerfile 程序中指定的。

事实上，系统内核管理的 uid 1000 只有一个，在宿主机中它被认为是用户 nick，而在容器中，它则被认为是用户 appuser。

所以有一点我们需要清楚：在容器内部，用户 appuser 能够获取容器外部用户 nick 的权利和特权。在宿主机上授予用户 nick 或 uid 1000 的特权也将授予容器内的 appuser。

从命令行参数中指定用户身份

我们还可以通过 docker run 命令的 --user 参数指定容器中进程的用户身份。比如执行下面的命令：

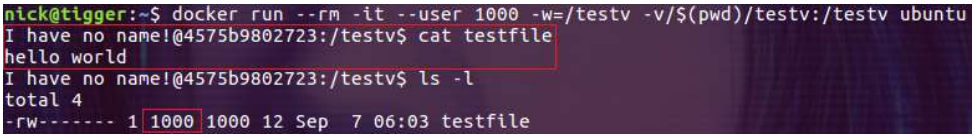
```
$ docker run -d --user 1000 --name sleepme ubuntu sleep infinity
```

```
nick@tigger:~$ ps aux | grep sleep
nick      45880  0.0  0.0   4532   824 ?        Ss   14:38   0:00 sleep infinity
```

因为我们在命令行上指令了参数 --user 1000，所以这里 sleep 进程的有效用户显示为 nick。进入到容器内部看一下：

```
$ docker exec -it sleepme bash
```

这是个什么情况？用户名称居然显示为 "I have no name!"！去查看 /etc/passwd 文件，里面果然没有 uid 为 1000 的用户。即便没有用户名称，也丝毫不影响该用户身份的权限，它依然可以读写只有 nick 用户才能读写的文件，并且用户信息也由 uid 代替了用户名：



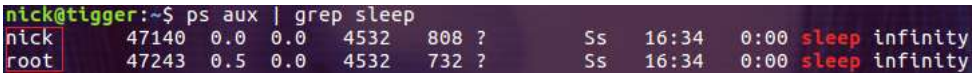
需要注意的是，在创建容器时通过 docker run --user 指定的用户身份会覆盖掉 Dockerfile 中指定的值。
我们重新通过 test 镜像来运行两个容器：

```
$ docker run -d test
```

查看 sleep 进程信息：


```
$ docker run --user 0 -d test
```

再次查看 sleep 进程信息：



指定了 --urser 0 参数的进程显示有效用户为 root，说明命令行参数 --user 0 覆盖掉了 Dockerfile 中 USER 命令的设置。

总结

从本文中的示例我们可以了解到，容器中运行的进程同样具有访问主机资源的权限(docker 默认并没有对用户进行隔离)，当然一般情况下容器技术会把容器中进程的可见资源封锁在容器中。但是通过我们演示的对数据卷中文件的操作可以看出，一旦容器中的进程有机会访问到宿主机的资源，它的权限和宿主机上用户的权限是一样的。所以比较安全的做法是为容器中的进程指定一个具有合适权限的用户，而不要使用默认的 root 用户。当然还有更好的方案，就是应用 Linux 的 user namespace 技术隔离用户，笔者会在接下来的文章中介绍如何配置 docker 开启 user namespace 的支持。

参考：
Understanding how uid and gid work in Docker containers
Introduction to User Namespaces in Docker Engine
Isolate containers with a user namespace

分类： Docker

标签： docker

« 上一篇： Linux ugo 权限
» 下一篇： 隔离 docker 容器中的用户

posted @ 2018-09-10 08:54 sparkdev 阅读(58910) 评论(15) 编辑 收藏 举报