Docker Tips: Running a Container With a Non Root User

Methods and examples

Luc Juggery
Sep 21, 2018 · 6 min read ★

TL;DR

One best practice when running a container is to launch the process with a non root user. This is usually done through the usage of the `USER` instruction in the Dockerfile. But, if this instruction is not present, it doesn't necessarily mean the process is run as root.

The rationale

By default, root in a container is the same root (`uid 0`) as on the host machine. If a user manages to break out of an application running as root in a container, he may be able to gain access to the host with the same root user. This access would be even easier to gain if the container was run with incorrect flags or with bind mounts of host folders in R/W.

Running a MongoDB container

If you don't know it yet, I highly recommend to give Play With Docker a try. Also known as *PWD*, it's an online playground where you can test all the latest Docker features without having to install anything locally. Once in PWD, you can create an instance, and you'll feel as if you're in the shell of a Linux VM.

Note: Under the hood, you'll have a shell but in an Alpine container in which the Docker daemon is installed. That's what is called *DinD*, for Docker in Docker, as the Docker daemon runs itself in a container.

Once in the terminal, let's run a container based on the MongoDB image:

```
[node1] (local) root@192.168.0.13 ~
$ docker container run -d -p 27017:27017 --name mongo mongo:4.0
8cce38822a23bbacb5349c5af63c50f1d2e371029f5b6332b1144fcc4f8cb723
```

And check, from the host machine, which user runs the `mongod` process:

```
[node1] (local) root@192.168.0.13 ~
$ ps aux | grep mongo
 1143 999 0:00 mongod --bind_ip_all
```

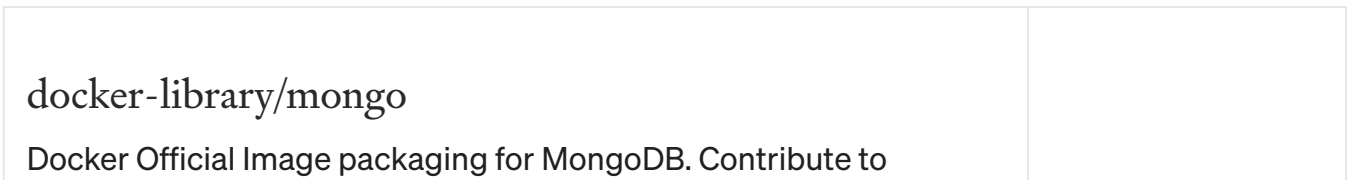
From the output above, we can see that the user identified by the `uid 999` is the one who owns the `mongod` process. Let's check the existing users on the host:

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/bin/sh
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/spool/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
postgres:x:70:70::/var/lib/postgresql:/bin/sh
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
dockremap:x:100:101:Linux User,,,:/home/dockremap:/bin/false
```

There is no user with `uid 999`, that's the reason why no user name can be mapped to this uid in the previous command.

Dockerfile

This is the Dockerfile used to build the MongoDB 4.0 image:



docker-library/mongo development by creating an account on...

github.com

In this file, there is no USER instruction, but we can see a new `mongodb` user is created in the image, and added to a `mongodb` group created at the same time. This is what the following instruction is used for:

```
RUN groupadd -r mongodb && useradd -r -g mongodb mongodb
```

As it's not specified through a USER instruction in the Dockerfile, this user is not used during the image construction; everything is done with root.

But, if we have a closer look at the end of the Dockerfile, we can see both `ENTRYPOINT` and `CMD` instructions.

```
ENTRYPOINT ["docker-entrypoint.sh"]

CMD ["mongod"]
```

As you probably know, the concatenation of those two instructions defines the command which is run when the container is started out of the mongo image. The command is then the following one:

```
$ docker-entrypoint.sh mongod
```

The ENTRYPOINT

Let's now take a look into the code of the `docker-entrypoint.sh file` :

docker-library/mongo

Docker Official Image packaging for MongoDB. Contribute to docker-library/mongo development by creating an account on...

github.com

The following piece of code at the beginning of the file is very interesting. This is the part where the user executing the process is changed from `root` to `mongodb` thanks to the `gosu` utility.

```
# allow the container to be started with `--user`
# all mongo* commands should be dropped to the correct user
if [[ "$originalArgOne" == mongo* ]] && [ "$(id -u)" = '0' ]; then
    if [ "$originalArgOne" = 'mongod' ];
        then chown -R mongodb /data/configdb /data/db
    fi
    # make sure we can write to stdout and stderr as "mongodb"
    # (for our "initdb" code later; see "--logpath" below)
    chown --dereference mongodb "/proc/$$/fd/1" "/proc/$$/fd/2" || :
    exec gosu mongodb "$BASH_SOURCE" "$@"
fi
```

Note: we can see in the Dockerfile, that the `gosu` utility is among the packages installed when the image is created.

Image based on Ubuntu

The first instruction in the Dockerfile indicates that ubuntu:xenial is the base image, the image from which the mongo image is created.

Let's run an interactive container based on Ubuntu and list the existing users:

```
$ docker container run -ti ubuntu:xenial

root@9e367c3d9ca1:/# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
```

```
systemd-timesync:x:100:102:systemd Time
Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network
Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd
Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus
Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
```

Let's now create a dummy user and group:

```
root@9e367c3d9ca1:/# groupadd -r mygrp && useradd -r -g mygrp myuser
```

and list the users once again:

```
root@9e367c3d9ca1:/# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time
Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network
Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd
Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus
Proxy,,,:/run/systemd:/bin/false
_apt:x:104:65534::/nonexistent:/bin/false
myuser:x:999:999::/home/myuser:
```

We can see the new user created as the `uid 999` , which is the uid of the first user created out of a fresh `ubuntu:xenial` image. This uid is the one used to run the `mongod` process as we saw before. As a reminder:

```
[node1] (local) root@192.168.0.13 ~
$ ps aux | grep mongo
 1143 999 0:00 mongod --bind_ip_all
```

Image based on Alpine

Application images are not necessarily based on `ubuntu:xenial`. A lot of them are based on Alpine (tiny distribution focused on security).

Let's add a new user out of a fresh alpine container and check its uid.

```
$ docker container run -ti alpine:3.8
/ # adduser -D myuser
/ # cat /etc/passwd
root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/bin/sh
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/spool/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
postgres:x:70:70::/var/lib/postgresql:/bin/sh
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
myuser:x:1000:1000:Linux User,,,:/home/myuser:
```

As we can see here, the id of the first user in an `alpine` image is 1000, different from the `uid 999` of an `ubuntu` image. If we add a user in an `alpine` image and run a process with this user (using the USER instruction in the Dockerfile, for instance), we will see the `uid 1000` as the owner of the process. Let's try it.

Let's use a simple Dockerfile that adds a user to an `Alpine` image and defines a basic `sleep 1000` command.

```
FROM alpine:3.8
RUN adduser -D myuser
USER myuser
ENTRYPOINT ["sleep"]
CMD ["1000"]
```

Let's build an image from it:

```
$ docker image build -t sleep:1.0 .
Sending build context to Docker daemon 1.775MB
Step 1/5 : FROM alpine:3.8
3.8: Pulling from library/alpine
4fe2ade4980c: Pull complete
Digest:
sha256:621c2f39f8133acb8e64023a94dbdf0d5ca81896102b9e57c0dc184cadaf5
528
Status: Downloaded newer image for alpine:3.8
 ---> 196d12cf6ab1
Step 2/5 : RUN adduser -D myuser
 ---> Running in a7474167f27d
Removing intermediate container a7474167f27d
 ---> 7a17f0862780
Step 3/5 : USER myuser
 ---> Running in b0a7eea711a4
Removing intermediate container b0a7eea711a4
 ---> d63533ce5be1
Step 4/5 : ENTRYPOINT ["sleep"]
 ---> Running in f0dfc3ea4495
Removing intermediate container f0dfc3ea4495
 ---> 763dd8ac4f40
Step 5/5 : CMD ["1000"]
 ---> Running in 14db1ea262f9
Removing intermediate container 14db1ea262f9
 ---> 978294e76184
Successfully built 978294e76184
Successfully tagged sleep:1.0
```

And then run a container from the newly created image:

```
[node1] (local) root@192.168.0.8 ~
$ docker container run -d sleep:1.0
534e340780a89b3a86917aff2c20405dadbd7d50cfe5cb03e9cb6786a0517f21
```

If we check the owner of the `sleep` process on the host, we can see it belongs to the user with `uid 1000`, the one that is created in the image.

```
[node1] (local) root@192.168.0.8 ~
$ ps aux | grep sleep
 1181 1000 0:00 sleep 1000
```

Summary

I hope those examples help you to understand some of the ways containers can be run with non root user, either through the image of the `USER` instruction in the Dockerfile or by changing the user at runtime (usually done within an `entrypoint` script). Another way we did not cover here would be through the usage of the `--user` flag when running a container.