Running rootless Podman as a non-root user

Users should have the choice to run containers as a non-root account. Here's how to make that happen.

Posted: October 8, 2019 | Dan Walsh (Red Hat)

Image by "A Different Perspective" from Pixabay

Recently, someone opened an issue on Podman.io: Does Dockerfile USER make sense for podman? The user was attempting to set up a container to run a Postgresql container as non-root. He wanted to create a directory for the Postgresql database in his home directory, and volume mount it into the container:

```
$ podman run -it **-v "$PWD"/html:/output:Z** schemaspy/schemaspy:snapshot -u postgres -t pgsql11 -
host 192.168.4.1 -port 5432 -db anitya
...
INFO - Starting Main v6.1.0-SNAPSHOT on 9090a61652af with PID 1 (/schemaspy-6.1.0-SNAPSHOT.jar started
by java in /)
INFO - The following profiles are active: default
INFO - Started Main in 2.594 seconds (JVM running for 3.598)
INFO - Starting schema analysis
**ERROR - IOException**
**Unable to create directory /output/tables**
INFO - StackTraces have been omitted, use `-debug` when executing SchemaSpy to see them
```

Does running rootless Podman as non-root make sense?

More Linux resources

- Advanced Linux Commands Cheat Sheet for Developers
- Get Started with Red Hat Insights
- Download Now: Basic Linux Commands Cheat Sheet
- Linux System Administration Skills Assessment

The issue he was having was that the directory he created in the home directory, `$PWD/html` was owned by his UID. This UID looks like `root` inside of the container, and the Postgresql process inside of the container does not run as root: It runs as the `postgres` user (`-u postgres`). Therefore, the Postgresql process is unable to write to the directory.

The easy solution to this problem is to chown the `html` directory to match the UID that Postgresql runs with inside of the container. However, if the user attempts to chown the file:

```
chown postgres:postgres $PWD/html
chown: changing ownership of '/home/dwalsh/html': Operation not permitted
```

They get permission denied. This result is because the user is not root on the system, and is not allowed to chown files to random UIDs:

```
$ grep postgres /etc/passwd
postgres:x:26:26:PostgreSQL Server:/var/lib/pgsql:/bin/bash
```

If the user adds `sudo` to chown the directory, they will get a similar error. Now the directory is owned by UID 26, but UID 26 is not mapped into the container and is not the same UID that Postgres runs with while in the container. Recall from my previous articles on user namespace that Podman launches a container inside of the user namespace, which is mapped with the range of UIDs defined for the user in `/etc/subuid` and `/etc/subgid`.

On my system, I run with this user namespace:

```
$ podman unshare cat /proc/self/uid_map
      0    3267      1
      1    100000    65536
```

This result shows that UID 0 is mapped to my UID, 3267, while UID 1 is mapped to 100000, UID 2 is mapped to 100001, and so on. This result means that inside of the container, UID 26 runs as UID 100025.

Excellent, let's try that:

```
$ chown 100025:100025 $PWD/html
chown: changing ownership of '/home/dwalsh/html': Operation not permitted
```

Well, that did not work either. The problem is that even though my user account can run a user namespace with these mappings, I am not currently in a user namespace. I need to use the `podman unshare` command, which drops you into the same user namespace that rootless Podman uses, so things look exactly the same for `unshare` as they do for rootless:

```
$ podman unshare chown 100025:100025 $PWD/html
chown: changing ownership of '/home/dwalsh/html': Invalid argument
Error: exit status 1
```

Still incorrect. The issue now is that the `chown` is happening inside of the user namespace, so `chown` needs to use the original UID, not the mapped UID:

```
 $ podman unshare chown 26:26 $PWD/html
```

Outside of the user namespace, this result looks like:

```
$ ls -ld $PWD/html
drwxrwxr-x. 2 100025 100025 4096 Sep 13 07:14 /home/dwalsh/html
```

Now, when the user runs the container, it is successful. The Postgresql process inside of the container runs as UID 26 inside of the container (and 100025 outside).

But let's get back to the original question: "Does running rootless Podman as non-root make sense?" If you are already running the container as non-root, why should you run Postgresql as a different non-root in the Podman container?

By default, rootless Podman runs as root within the container. This policy means that the processes in the container have the default list of *namespaced capabilities* which allow the processes to act like root inside of the user namespace, including changing their UID and chowning files to different UIDs that are mapped into the user namespace. If you want to run the container as the Postgresql user, you want to prevent this access.

This setup also means that the processes inside of the container are running as the user's UID. If the container process escaped the container, the process would have full access to files in your home directory based on UID separation. SELinux would still block the access, but I have heard that *some people disable SELinux*. Doing so means that the escaped process could read the secrets in your home directory, like `~/.ssh` and `~/.gpg`, or other information that you would prefer the container not have access to.

If you run the processes within the container as a different non-root UID, however, then those processes will run as that UID. If they escape the container, they would only have world access to content in your home directory. Note that in order to work with the content in these directories, you need to run a `podman unshare` command, or set up the directories' group ownership as owned by your UID (root inside of the container).

With Podman, you want to allow users to run any container image on any container registry as non-root if the user chooses. And I believe that running containers as non-root should always be your top priority for security reasons.