José Miguel Parrella
Posted on Feb 1, 2020 • Updated on Feb 3, 2020

A quick guide to podman and toolbox in Debian (and maybe Ubuntu)

#debian #docker #containers

Over the last couple years I've been spending a lot of time playing with containerized development environments such as WSL2 and Crostini. I also run Fedora in a NUC to try and keep up with `systemd`, `cgroup2`, `podman` and other technologies. But since I brought my Debian laptop to FOSDEM20, I wanted to play with Podman and Toolbox natively.

Why `podman` is important

Remember when `docker` bundled daemon *and* tools? Although it was eventually decoupled, many of us learned a formulaic usage of the `docker` command and it's not unusual to find the legacy packages in many of our systems today.

That was certainly the case for me: I *knew* that we needed to decouple to achieve rootless containers, registry-side building and interchangeable runtimes. But the options expanded rapidly and since I was wary of keeping too many tools with overlapping experiences in my system, I continued to rely on the `docker-ce` packages.

I believe `podman` is a credible replacement for my needs, but if you want to play around with `toolbox` then `podman` is a requirement, even if one that you'll be happy with. All of this stack is integrated and tested in Fedora before other RPM-based distros (let alone Debian derivatives) so it does take a little bit of work but the results and basic functionality is pretty much comparable. Let's begin.

Installing `podman` and `toolbox` in Debian

While I was attending FOSDEM, two speakers (one from SUSE, one from Red Hat) wondered if `podman` and `toolbox` were available for Debian and derivatives, such as Ubuntu. They assumed so, but weren't quite sure. (A word of warning: there's a snap by Ondrej called "toolbox", but this is not what we're discussing here.)

- For `podman`, openSUSE's Kubic builds `deb` packages that work in Debian and Ubuntu. This is the current installation method, and it's what I used. Albeit rocky, there's ongoing work to get this package officially into Debian.
- For `toolbox`, which is a shell script, you can fetch a release and place the script in your `PATH`. Make sure you install `flatpak`, as that's needed (there could be other dependencies, but in my reasonably vanilla desktop system, I was only missing a `sudo apt install flatpak -y`)

One last thing, part of the rootless magic relies on user namespaces so make sure you `echo 1 | sudo tee /proc/sys/kernel/unprivileged_userns_clone` and understand the security implications of that.

What can `toolbox` do?

In Fedora Silverblue, `toolbox` is used to provide a mutable working environment on top of a (mostly) immutable operating system such as Silverblue or CoreOS (you can watch rishi's presentation for full impact)

In our case, since we're running Debian in an environment I regularly mutate, we care less about that aspect but we still want a working environment that's easy to step in and out of.

Either way, you might be wondering how that is any different from having a pet `docker run -it ... /bin/bash` or a playground VM whether that's with `libvirt` or `lxd`, in a public cloud, VPS provider or somewhere else. Or something custom you have with a mix of `pyenv` or `nix-shell`...

The difference with `toolbox` is that it overlays this environment on top of your profile, carries your shell settings and helps users resolve just like in the host. So you wouldn't need to worry about things like mounting a `9p` filesystem or sync'ing files and adjusting ownership, etc.

So when you enter the `toolbox` environment, you feel like you're in your regular environment, but things you change beyond your profile are kept to the container. Here's an example:

```
bureado@crucia:~$ echo hi-from-host > hello
bureado@crucia:~$ htop -v

Command 'htop' not found, but can be installed with:

sudo apt install htop

bureado@crucia:~$ toolbox enter --container debian-toolbox-latest
bureado@toolbox:~$ cat hello && echo hi-from-container > hello
hi-from-host
bureado@toolbox:~$ htop -v
htop 2.2.0 - (C) 2004-2019 Hisham Muhammad
Released under the GNU GPL.

bureado@toolbox:~$ logout
```

```
bureado@crucia:~$ cat hello
hi-from-container
```

In this example you see how I switched from my laptop (`crucia`) to the container (`toolbox`) sharing files and changes in my profile but keeping any additions (in this case, a previous `apt install htop` I made in `toolbox`) to the container.

The role of `podman`

As I mentioned earlier, `toolbox` is a script. A 2.5K+ SLOC script with close to 60 mentions to `podman` . So `podman` is a real hero here. If I run `toolbox list` , I get:

```
bureado@crucia:~$ toolbox list
IMAGE ID      IMAGE NAME                          CREATED
b31c77acc328  localhost/debian-toolbox:latest     About an hour ago
1787a6a86277  localhost/ubuntu-toolbox:latest     3 hours ago

CONTAINER ID  CONTAINER NAME         CREATED            STATUS                 IMAGE NAM
267d9c17c3f8  debian-toolbox-latest  About an hour ago  Up About an hour ago   localhost
baf2ed3ece9b  ubuntu-toolbox-latest  3 hours ago        Up 2 hours ago         localhost
```

This lists two OCI container images and two actual containers running. Doesn't that sound like the output of `docker ps` ? Well, that's what `podman` can replicate:

```
bureado@crucia:~$ podman images
REPOSITORY                 TAG         IMAGE ID       CREATED             SIZE
localhost/debian-toolbox   latest      b31c77acc328   About an hour ago   445 MB
localhost/ubuntu-toolbox   latest      1787a6a86277   3 hours ago         340 MB
```

```
docker.io/library/ubuntu   19.04      c88ac1f841b7   2 weeks ago         72.4 MB
docker.io/library/debian   unstable   0e26bcfa03fc   5 weeks ago         122 MB
bureado@crucia:~$ podman ps
CONTAINER ID  IMAGE                                COMMAND               CREATED
267d9c17c3f8  localhost/debian-toolbox:latest  toolbox --verbose...  About an hour ago
baf2ed3ece9b  localhost/ubuntu-toolbox:latest  toolbox --verbose...  3 hours ago
```

And of course, `podman` allows me to do operations like `start` and `stop` and `run` -- all rootless. In fact, I can take a Dockerfile and build it with `podman`, as you see below.

```
bureado@crucia:~$ podman build . -t debian-toolbox
STEP 1: FROM docker.io/library/debian:unstable
STEP 2: ENV NAME=debian-toolbox VERSION=unstable
--> Using cache 96545d7a49c3a47a39cb9f2fc8c6b40d5240b02dfa1a0c2ac9efcf976d67d44c
...
```

I encourage you take a look at `podman` which also runs on Mac (and even WSL2) by virtue of its remote-client support.

There are also underlying technologies to `podman` such as `conmon` and you can learn more about it replaying this FOSDEM session: Podman - The Powerful Container Multi-Tool.

Where did the image come from?

You probably noticed from my output that there are two `(debian|ubuntu)-toolbox-latest` *containers* that are using two `(debian|ubuntu)-toolbox` *images*. Where did those come from? (And yes, this also means this article is probably helpful if you use Ubuntu instead of Debian.)

This image is supposed to be functionally close to your actual host working environment (to provide consistency) and, in fact, it needs two special `LABEL`s asserting so in order for `toolbox` to ingest them.

Here's an example of a Debian image for toolbox where you can see the additional packages being installed and the labels being declared.

Once you do all of that (with `podman`), you instruct `toolbox` to recognize this image and create a standby container which you can `enter`. Altogether, it looks like this:

```
podman build -t debian-toolbox -f Dockerfile
toolbox create -i localhost/debian-toolbox:latest
toolbox enter -c debian-toolbox-latest
```

And everytime you `toolbox enter`, you can mutate that system without polluting your main one - except for files in your profile. That's it! I learned a good deal about the underlying technology just going through this process and reading the code.

In the coming months, I'll be evaluating this against my current setup of using `nix` and Python's `venv` while looking at more emerging technology that can be applied to this space, from other tools in this stack like `buildah` and `skopeo` to things like MicroK8s.

Let me know if you find this useful and/or interesting, and comments always welcome!

Top comments (1)

ebardie • Jun 10 '20 • Edited on Jun 10

Thanks for the article José. Some clarifications:

Build and install instructions are currently stuck in this pull-request. You'd like the consommé? Sure:

```
cd "${UNTARRED_TARBALL_DIR}"; mkdir build && cd build && meson -
Dprofile_dir=/etc/profile.d .. && ninja && sudo ninja install
```

All three files (`Dockerfile` , `extra-packages` , and `README.md`) from here (i.e. José's pull request #371 - another thank you :)) need to be in the same directory before running:

```
podman build -t debian-toolbox -f "${TARGET_DIR}"/Dockerfile .
```

Code of Conduct • Report abuse

José Miguel Parrella

Yet another open source enthusiast in Redmond

WORK
Principal Program Manager, Office of the Azure CTO at Microsoft

JOINED
Jan 2, 2019

Trending on DEV Community 🔥


Which is your First programming language ?
#developer #programming #career #firstpost


Is HTML a Programming Language?
#discuss #html #programming


Get your own DEV wrapped for 2023 🎁
#webdev #docker #tutorial #meta