Part 3 › Official Images and trust

Official Images and trust

We've focused on using Docker as a tool to solve various types of problems. Meanwhile we have decided to push some of the issues until later and completely ignored others.

The goal for this part is to look into some of the best container practices and improve our processes.

In part 1, we mentioned how the Alpine Linux image is much smaller than Ubuntu, but we didn't dive any reasoning why we might pick one over the other.

On top of that, we have been running the applications as root, i.e. the super user, which is potentially dangerous.

Look into official images

Which version is considered to be the "official" version is up to the maintainers of Docker Official Images to decide. The official images repository contains a library of images considered official. They are introduced into the library by regular pull request processes. The extended process for verifying an image is described in the repository README.

Many of the most well-known projects are maintained under the docker-library organization. Those include images such as Postgres and Python. However, many of them are included in the library but are managed by a separate organization, like Ubuntu and Node.js

Let's look into the Ubuntu image on Docker Hub and trace the process.

The description/readme says:

```
What's in this image?

 This image is built from official rootfs tarballs provided by Canonical
 (see dist-* tags at https://git.launchpad.net/cloud-images/+oci/ubuntu-base).
```

We can see that the image is built from https://git.launchpad.net/cloud-images/+oci/ubuntu-base.

Let's take a closer look at Ubuntu to verify where it comes from. If you click the Dockerfile link of https://hub.docker.com/r/_/ubuntu/ you'll be given a json file instead of the Dockerfile contents. The digest seems to contain the valid digest for a single architecture version listed on Docker Hub (amd64). But after downloading it and checking the digest `docker pull ubuntu:22.04 && docker image ls --digests` the result does not match up.

We find a Dockerfile from the repository here. We can increase trust that it's the same as the image we downloaded with `image history`:

```
$ docker image history --no-trunc ubuntu:22.04
```

The output from image history matches with the directives specified in the Dockerfile. If this isn't enough, we could also build the image ourselves.

The first line states that the image starts FROM a **special** image "scratch" that is just empty. Then a file `ubuntu-*-oci-$LAUNCHPAD_BUILD_ARCH-root.tar.gz` is added to the root from the same directory.

Notice how the file is not extracted at any point. The `ADD` instruction documentation states that "If src is a local tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory."

We could verify the checksums of the file if we were interested. For the Ubuntu image automation from the launchpad takes care of creating the PRs to docker-library and the maintainers of the official images repository verify the PRs.

You can also visit the Docker Hub page for the image tag itself, which shows the layers and warns about potential security issues. You can see how many different problems it finds here.

Now we have learned that the build processes are open and we can verify it if we have the need. In addition, we learned that there's nothing that makes the "official" images special.

"You can't trust code that you did not totally create yourself."

- Ken Thompson (1984, Reflections on Trusting Trust)

Edit this page

Previous
« Introduction to Part 3

Next
Deployment pipelines »