Keeping Up with Docker Official Images (atomist.com)

33 points by slimslenders 4 hours ago | flag | hide | past | favorite | 13 comments

add comment

mfontani 2 minutes ago [–]

I keep track of upstream image changes using https://github.com/crazy-max/diun

... but I also separately ensure all software installed in a docker image is pinned to a version, and have a process I run daily to check whether the upstream packages versions have changed, in which case I rebuild the images which then get the updated (possibly security) version.

It's fiddly, and a lot of bash and perl. I'd welcome a similarly trust-able tooling from a reputable source.

reply

itamarst 28 minutes ago [–]

Seems like the author is assuming the official images get security updates applied immediately. They don't. So you need to apply system package updates yourself, separately from updating the base image.

https://pythonspeed.com/articles/security-updates-in-docker/ has an example of a security update that was missing in the official Ubuntu base image, two weeks after Ubuntu released the updated package. I've also seen missing security updates in the Debian base images used for things like Python and OpenJDK official base images.

This means that you need to rebuild automatically yourself on a schedule (or as result of distro updates), e.g. https://pythonspeed.com/articles/docker-cache-insecure-image...

reply

slimslenders 17 minutes ago [–]

I came across your article on security updates when I was researching this. Thanks for writing it - it was super helpful. One of the sub goals for us is to quantify the lag that you've observed here. We're also reviewing an automated check that would flag times when an apt upgrade would be material. This is very much based on your bad arguments 1 through 4.

reply

michaelperel 24 minutes ago [–]

Shameless Plug: I wrote a cli-plugin for docker, docker-lock, to solve the mutable tag problem without having to manually specify hashes - https://github.com/safe-waters/docker-lock

It creates a Lockfile (think package-lock.json) that tracks the image digests (sha256 hashes) of your base images, so you will always know exactly which images you are using even if you only specify tags. This way, you can know if a base image has changed, yet still receive important security updates that you would not receive if you hardcode the digest. It supports any registry, so is useful even if you are not using Dockerhub. It also works with Dockerfiles, docker-compose files, and Kubernetes manifests.

I hope anyone dealing with this issue finds it helpful :)

reply

ImJasonH 1 hour ago [–]

Incidentally, I've proposed official image annotations[0] that would let an image tell you what its base image is, both by immutable digest and by mutable tag, so you could detect this drift automatically.

Automated tooling could look at these annotations and notify maintainers, or proactively rebuild/rebase when base images change[1].

By having this information on the images themselves, you don't have to deal with as much source repo churn, though you might want that too.

If your app layers have a strong enough contract with your base layers (buildpacks is really good for this!), then you can rebase[2] instead of rebuilding from source.

[0] OCI spec proposal: https://github.com/opencontainers/image-spec/pull/822 [1] Proof of concept in the `crane` tool: https://github.com/google/go-containerregistry/pull/960 [2] https://github.com/google/go-containerregistry/blob/main/cmd...

reply

slimslenders 33 minutes ago [–]

Thanks! And ya, we found the spec! We are in the middle of adding a feature to help devsecops teams notice when org.opencontainers.image.revision and org.opencontainers.image.source labels are missing (those were the first two which we considered mandatory to ensure that downstream admission controllers can validate checks that are indexed by sha). I had not realized that you were proposing further annotations for base images here. That sounds really promising. I think having a standard set of information available in the images is critical.

reply

denysvitali 1 hour ago [–]

Despite the idea is interesting, I'm not sure it's the right solution to the problem.

The problem should be fixed upstream, and tag updates should not happen for patch releases.

The biggest issue that I see with Docker is the fact that tags are easily overwritten, and as the author pointed out in the article, a moving tag breaks builds and makes them unreproducible.

I still wonder if there is a cleaner solution than using image digests, this whole trend is going to make the Dockerfile FROM more confusing.

reply

slimslenders 27 minutes ago [–]

Keeping the FROM readable is indeed a challenge. We thought about a Dockerfile comment to add the context that developers need. There's this peculiar aspect of docker image naming where you can include both a tag and a digest (repository:tag@digest) too. I also think that the pull request is a good opportunity to give good context about where you are now, and what the proposed upstream changes are really bringing.

reply

jzelinskie 3 hours ago [–]

I think GitHub will do this for you now: https://docs.github.com/en/code-security/supply-chain-securi...

reply

slimslenders 2 hours ago [–]

Ya, it's similar. We needed to add an initial pinning PR, a vulnerability and change log comparison, and scanning for unsupported tags. We also added support for private registries so that we could use the same pattern for non-official images. But the inspiration came from Official image repositories.

reply

0xbadcafebee 2 hours ago [–]

Distroless is a misnomer. Any "distribution" of files that has its own build process and unique end state, is a distribution. So, "distroless" containers are still "a distro", it just has less extraneous files. This has some useful properties, but at the same time downsides: bolting-on additional requirements is significantly harder, and development outside of a tightly controlled CI system becomes burdensome.

At the end of the day, any such change comes down to a single principle: the "distance" from development to production. If whatever you do puts more distance between how development is done and how production works, then in the long run it's counter-productive. Call this the "developer distance principle".

reply

slimslenders 2 hours ago [–]

Thank you, and yes. I love that idea of a distance metric. I think you're also pointing out that distroless images can

end up increasing this developer distance, right? I was originally drawn to the idea of distroless images as a way to reduce vulnerabilities. However, staying up to date with a well maintained distro is effective too.

reply

0xbadcafebee 22 minutes ago [–]

Yeah; specifically, if your developers don't use the distroless containers to develop their apps (or stop using them when they become burdensome to update), then the environments are becoming divergent, which will lead to divergent behavior. The solution [incl. for things like vulnerability management] is the same idea behind Shift Left: move as much of the work "left" (earlier) in the pipeline/value stream/etc as possible.

The more you shift left, the smaller the distance from dev to prod, the better the outcomes. Whatever environment the developers want to use to develop, make the production system the same; then improve the development environment in order to improve production. Over time this will need to change as complex systems are hard to replicate locally. But the closer they are, the better.

reply