***Best approach to assign a version to a containers.***

Image is the most basic concept that you have known since you are new to Docker. But building an image to deploy to production is a much more complex story. How to know which code is running on production is at which commit and when to deploy. What do you want to rollback? Then must integrate with CI again.

Let us find out some ways to tag images to deploy to production effectively.

**Stable tags**

When it comes to deploying docker image on production, from the beginning, I also thought that using the latest tag or master for production, when deploying to staging, I could use the develop tag or staging tag. But this way is not okay. The same tag master can have multiple images. If your team deploys to production regularly, then tagging like that really does not make sense because it is completely indistinguishable from which. If you want to rollback to an older version, this is not possible because there is no way to distinguish the old image from the new image.

This is called stable tagging because our tags are always constant, but the image content is changed. Of course, as we have just seen above, this way is very unstable.

**Unique tags**

When deploying to production, you must give each image a unique tag. But which value should we use to tag it? The tag we use should satisfy the following requirements.

* Must be unique (of course .\_.)
* Must easily identify the code deployed at the time and easily find out the tag to be deployed.
* Easily find the image that was deployed last time for rollback if needed.
* Must be easy to use with CI / CD workflow.

Here are a few ways to tag images that I’ve ever seen people do.

**Timestamp**

foo/bar:2019-06-01\_12-23-32  
foo/bar:2019-06-08\_14-24-12  
foo/bar:2019-06-15\_12-12-32

This method simply uses the build time to tag the image as well. For example, on June 1, 2019 at 12:23:32, the tag will be 2019–06–01\_12–23–32 for example. This is a familiar version of the release version name.

Look fine, but to tag the image is not very good. This method can easily see the build time, easily know which image was built before or after. However, we cannot deduce the release version from build image time. From code to find the image name even more impossible. Then there is a different time zone.

Generally, the name is both long and meaningless, too many disadvantages.

**Semantic versioning**

foo/bar:1.0.0-beta.1  
foo/bar:1.0.0-beta.2  
foo/bar:1.0.0  
foo/bar:1.1.0  
foo/bar:1.1.1

You can easily see that the official images on the Docker Hub are tagged in this way. But if used to tag images to deploy, is it okay?

Not bad either. If you remember to increase the version each time you deploy, each tag only attaches to only one image. It is also easy to identify each version corresponding to which code is deployed if you remember to push the new tag to the repo every deploy. Because this is a semantic version, so even if you look at it, you will know which ones are new and which are old. Finding the previous version to deploy is too simple.

But now there is more work to be done every time deploy. You must bump the version yourself each time deploying because of how the CI knows when you want to increase the version. The deploy process is now more complicated with manual operations. Now not only have to worry about bugs in the code, but also to worry about bugs in deploying .

**Commit hash**

foo/bar:0f299b0  
foo/bar:b5915dc  
foo/bar:9cce148  
foo/bar:0845805  
foo/bar:e478a78

We will use the hash of the currently deployed commit to tag the image being built. This way is better than the 2 above. From the tag, it is easy to deduce the commit to be deployed, but it is also easy to find the deployed tag. Most CI support you easily follow this way. CI usually provides commit hashes in an environment variable for you to use. For example, with Gitlab like this.

docker build . -t foo/bar:${CI\_COMMIT\_SHA:0:7}

Without using CI, get commit hash from git is not difficult either.

git log -1 --pretty=%h

This is probably the most recommended and used way today. This approach is also nearly perfect, meeting all the requirements we set, just a little difficult to determine the new image order a little bit old.

There is still one more problem. What if you need to rebuild the image because the base image has changed and the code remains unchanged. For example, your image is built from node: 10-alpine. If there is a bug fix on the base image you want to update, your code will remain unchanged. At that time, your image still has the same tag because there are no new commits but the image content has changed because the base image has changed. The solution is that you need to use the base image with the full version, for example node: 10.16.2 instead of node: 10. Every time you want to update the minor version of the base image, you must update it manually, which takes a little more effort.

Also, now that I want to rollback, I don’t know what the previous image is, it will take a bit of work. If you use CI then you only need to see which previous commit is committed.

**CI build number**

foo/bar:circle-ci-30  
foo/bar:circle-ci-64  
foo/bar:circle-ci-72  
foo/bar:circle-ci-122

This method is of course only used if you use CI (if not used yet, give it a try). We will use the build number of the CI to tag the image being built. Along with that, the name of the CI always makes it easier to understand (for example, **gitlab-ci-123456**, **123456**can be easily confused with commit hash), also in case we change to another CI and build number may be duplicated. For example, with Gitlab CI, do this:

docker build . -t foo/bar:gitlab-ci-${CI\_PIPELINE\_ID}

CI will ensure this build number is unique and in ascending order, so look at what we know immediately which images are newer or older. From the build number we can easily find the deployed commit. From CI build, we can immediately deduce the image tag. We also have a history of deploying in CI when we look at the build list. When you trigger the build on the CI, the code does not change, for example, in the case of the update base image above, we also have a new image tag without fear of being duplicated like when using commit hashes.

Up to now, this is the best way I’ve ever seen and used.

**Utilizing build cache**

Go back to the stable tag a bit, actually it’s not quite as useless as I said from the beginning. When building an image, tags are not the only things to consider. Build cache is also an issue. Without it, the build time will increase significantly, having to repeat the same steps very time consuming.

But where to get the cache when we build on CI. No layers will be saved as they were built on your computer. So, you have to use the built image from before. But how to know which the image (or tag) is is previously deployed. This is where we need the stable tag, always constant and attached to the latest image. In addition to the unique tag we will add another stable tag. For example:

docker build . -t foo/bar:master -t foo/bar:gitlab-ci-${CI\_PIPELINE\_ID}

The next time building we can pull the previously deployed image and use it to build the cache:

docker pull foo/bar:latest  
docker build . -t foo/bar:master -t foo/bar:gitlab-ci-${CI\_PIPELINE\_ID} --cache-from foo/bar:master

**TL;DR**

* Do not use stable tags to deploy.
* Use commit hash or if using CI then use build number as <c significant — <build number>.
* It is also possible to use a stable tag in tandem with a unique tag to use the build cache.