Updating Docker Containers - What aBout my data?

- Update Docker - Keep your data too.

Update Docker - Keep your data too.

https://www.youtube.com/embed/KfpMUBnmmq4

If you've been following my channel on YouTube, or my articles here for any amount of time, you probably know that I really love Docker as a means to self host pretty much anything. While docker is an amazing tool, many of your have been on this journey of learning about docker with me for a few years now. I definitely still don't know everything, but I do know about updating containers, and keeping data safe and secure.

Admittedly, when I first started using docker, I didn't understand how to persist data between restarts of the same container, or updates of the container to newer image versions. I really just though something was broken about it, and it put me off of using docker for another year at least.

Once I realized that persisting data just took a couple more lines of yaml in a `docker compose`, or a couple of more flags in the `docker run` command, I was so much better off. Add that to a really good back up strategy, and you'll be able to update your docker containers with little concern over data loss in the process.

What You'll Need

- A system with Docker and optionally Docker Compose installed on it
- At least one docker based application running in a container, or an application you would like to run in mind.
- About 10 - 15 minutes of your time

Backing Up Docker Containers

Really and truly, this is an entire article all on its own. So much so, in fact, that I already have an article on the topic, as well as a video about how I handle backing up my docker containers and data.

In short, you should create a script (also available in that other article) that will move into each applications folder, and stop the running containers (docker compose is really handy for this), and repeat this for each folder / application you have. When they are all stopped, create an archive of the parent "docker" folder, then go back in an restart each application. Copy the archive to the final storage location (USB Drive, Network Share, separate drive or partition, cloud, etc). Finally, remove the archive you created from your host machine if you want.

The part where you stop the containers, can be the scariest, especially if you haven't setup your data volumes to be in the same folder as the docker-compose file. In this case, you need to know where your data is being stored on your host system.

Volume Mappings in Docker

Volume mapping is how you persist data (keep it from being deleted) in docker between container restarts / updates. Most of the time the person / group who makes a docker image available will list out the volumes you will want / need to map. Think of it like creating a symlink from some path on your host system, to a path predefined by the application inside the docker container.

For instance. My media for my Jellyfin server is actually stored in /mnt as it's on an NFS share that has 12 TB of space. None-the-less, I have a volume mapping that makes sure the data and configs are all persisted between updates.

My volume mapping for Jellyfin looks like:

```
volumes:
  - /home/brian/docker/jellyfin/config
  - /home/brian/docker/jellyfin/cache
  - /mnt/data/media:/media
```

Here, you can see that I've mapped my configuration folder to the same folder where I keep the Jellyfin docker-compose.yml which is `/home/brian/docker/jellyfin`.

I also have the cache mapped there. I could map the cache to some other location, but this is fine for my use case.

You can see that my media is mapped to `/mnt/data/media` which maps to the container location of `/media`.

Because I've made these mappings, when I start, stop, restart, recreate, or update my Jellyfin container my media, configurations, and cache of data are all persisted, and my data is not lost. If you don't map the volumes you need on a docker container, even one where you are using a database container, the data is considered "volatile". This means when you start, stop, restart, update, or recreate, the container, the data is removed and everything is recreated.

So, always make sure you keep your volumes mapped in docker. This will save you a ton of heartache when it comes time for updates.

Why Stop the Containers for Backup?

While it's not technically necessary, it's always a best practice to stop the containers. This helps to ensure that data will not be in the middle of a change while the backup is happening. Just like a primary OS, reads / writes of data while it's in the middle of a change can create data corruption. If you're like me, and are generally the only person using your applications (except for Jellyfin), then you could probably get away with running a backup script without stopping the containers. The thing you'd hate is finding out that your backup data is all corrupted on the day you actually need that backup...so I highly recommend stopping the containers, creating the archive, then starting them up again.

How do I Update?

There are several ways that you can update your containers. It's highly dependent on you, and your faith in the person(s) creating the images you're using for your containers.

The Manual Way

Docker Compose is your friend. Believe me. It takes a normally long docker run command, and gives it structure using yaml syntax, and makes it easier to put multiple services (container definitions) into a single file allowing you to more easily have them work together. It also creates a special docker network for each compose file unless you specify a network you want the containers to be on.

If you are using Docker Compose for your container setup and management, then updating your containers is super easy. Of course, always make sure you've done a recent backup (just in case). Next, go into the folder where the docker-compose.yml file is located, and run the command:

`docker compose pull` if you're on an older version of Docker Compose, and you get an error with that command, you may need to use `docker-compose pull` instead.

Presuming you are using the `latest` tag on your containers this will trigger docker to seek out any newer images available for your applications (services) and pull them down. This **will not** update the running container yet. It only pulls down the newer images if any exist.

Once you've pulled down the new images, you can run

`docker compose up -d`

This will update your running containers to the newest image version. I've rarely had an issue with this method of pulling and updating my containers. It's actually a very useful way to handle updates, as it updates many images in an application stack all at once, and can take away several

manual steps.

If you're really feeling brave, you can combine the two commands into one line, and have them run one after the other like this:

`docker compose pull && docker compose up -d`

If you're like me, and you like to follow the logs after an update, then you can add the command, on it's own, or in the single line like this:

On its own:

`docker compose logs -f`

In the single line:

`docker compose pull && docker compose up -d && docker compose logs -f`

I Don't Use Docker Compose

Docker Compose is not a requirement, it's simply a tool that helps make Docker a bit more organized and slightly simpler (IMO). You can update your containers using the straight Docker CLI as well.

To stop a container run:

`docker stop <container name>` for instance `docker stop jellyfin`.

If you're not sure of the names of your running containers, you can always get the names with `docker ps`.

To pull a new image, you can run

`docker pull <image name:tag>` for instance `docker pull jellyfin/jellyfin:latest`.

To update the container, you'll have to do a few more manual steps:

1. Stop the container
2. remove the container (yes, delete the container - but not to worry, your mapped volumes will persist the data for you).
3. create a new container with the same name, and same port and volume mappings as you used originally.

If we apply the above steps to Jellyfin, it looks like this:

`docker stop jellyfin`

`docker rm jellyfin`

`docker run -d --name jellyfin -e TZ=America/Chicago -p 8096:8096 -v /home/brian/docker/jellyfin/config:/config -v /home/brian/docker/jellyfin/cache:/cache -v /mnt/data/media:/media jellyfin/jellyfin:latest`

This is why I like Docker Compose, it's a bit easier to remember `docker compose pull` and then `docker compose up -d`.

The Automated Route

Let's say you're an adventurous and carefree type of person who likes to take a trip on the wild side... you know, the kind of person who adds bacon to a grill-cheese sandwich, or toasts the bread on a peanut-butter and jelly sandwich. Well, if that's you, then there are tools to help automate updates for your docker containers as well. One really exceptional tool is called Watchtower. I did a video and write up on it a couple of years ago, and it's still a great application for making sure you're staying up to date.

Watchtower, by default will watch all of the containers on your system, and will check on a schedule (that you can set) to see if any new images are available. If so, it will trigger docker to download the new image, and update the container with it. It's actually pretty awesome!

You can, however, setup watchtower to only check for new image updates, and then send you an email about the available updates. You use labels on the containers you don't want Watchtower to automatically update for you. The nice thing about a feature like this is it gives you the information that updates are available, but allows you the choice in updating or not.

What if something breaks?

This is precisely why we want a good backup strategy that we can depend on. I have had images pull down and break a running container. It's rare, but it does happen. I have also just done some really dumb things that messed up data or configurations for a running container that I then couldn't get back into and fix. The backups are super handy for quickly getting your entire docker folder, or just a single application folder out of, replaced on your production system, and back up and running just like ti was before you hit hard times.

I hope this information will give you solace in knowing that your data can be safe within Docker, and that you have a ton of power and control over your data integrity, data availability, data security, and over your systems reamining up to date.

Support My Channel and Content

Support my Channel and ongoing efforts through Patreon:
https://www.patreon.com/awesomeopensource