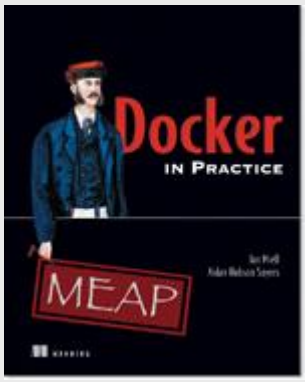

Docker: Staying Ship-Shape

By Ian Miell and Aidan Hobson Sayers

In this article, excerpted from the book Docker in Practice, we show you how and why you need to stay on top of the containers and volumes on your host.

If you're anything like us, your growing Docker addiction will mean that you start up numerous containers and download a variety of images to your chosen host.

As time goes on this Docker will take up more and more resources, and some housekeeping will be required. In this article, we show you how and why you need to stay on top of the containers and volumes on your host. We also introduce some visual tools for keeping your Docker environment clean and tidy.

Run docker without sudo

Problem--You want to be able to run the *docker* command without having to use sudo

Solution--Add yourself to the *docker* group

Discussion

Docker manages permissions around the Docker unix domain socket through a user group. For security reasons, distributions do not make you part of that group by default.

By adding yourself to this group, you enable the use of the *docker* command as yourself.

```
$ sudo addgroup -a username docker
```

Then restart docker and fully log out and in again; or just reboot your machine if that's easier. Now you don't need to remember to type sudo or set up an alias to run docker as yourself.

Housekeeping containers

A frequent gripe of new Docker users that are running up many containers while getting to grips with it is that in a very short space of time you can end up with many containers on your system in various states, and there are no standard tools to manage this on the command line.

This technique gives you some time-saving tools to manage this problem.

Problem--You want to prune the containers on your system.

Solution--Set up aliases to run the commands to tidy up old containers.

Discussion

The simplest approach here is to delete all containers. Obviously this is something of a nuclear option that should only be used if you're certain it's what you want! The following command will remove all containers on your host machine.

```
$ docker ps [#A]-a [#B]-q [#C]| xargs [#D]--no-run-if-empty docker
[#E]docker rm [#F]-f
```

#A - The -a flag lists all the containers on your host machine
#B - The -q flag lists only the container id field, and no headers
#C - The output is piped for processing by the xargs command
#D - If no output is generated by the docker ps command, then xargs is not run. This avoids seeing an error message in this case. The xargs command runs a command for each line of output received through the pipe
#E - For each container id received, call docker to remove the container
#F - The -f flag forces removal even if the container is running

If you have containers running that you may want to keep, but want to remove all those that have exited, then you can filter the items returned by the *docker ps* command.

```
$ docker ps -q [#A]--filter status=exited | xargs docker rm [#B]
```

#A - The --filter flag tells the *docker ps* command which containers we want returned. In this case we are restricting to containers that have exited. Other options are: running, restarting
#B - This time we do not force the removal of containers as they should not be running based on the filter we've given

As an example of a more advanced and selective removal, this command will remove all containers (running and not running) except those that have exited with a non-zero error code. You may need this if you have many containers on your system, but want to examine any containers that exited unexpectedly.

```
$ [#A] cat [#B]
<(docker ps -q) [#C]
<(comm [#D]
-1 [#E] <(docker ps -a -q --filter=status=exited |
[#H] sort) [#F] <(docker ps -a -q --filter=exited=0 |
[#H]sort)) [#G]
| xargs docker rm -f
```

#A - Push the output of the files created with the process substitution operators
#B - Run the standard *docker ps -q* command to get the running containers. This as a process substitution parameter, which creates a file that is taken as a parameter to the cat command
#C - Run a *comm* command to compare the output of two file objects
#D - The *-1* argument to *comm* suppresses lines that appear in only the first file's output (i.e. those that have exited with non-zero exit code)
#E - This process substitution outputs the exited container IDs within a file as an argument to *comm*
#F - This process substitution outputs the containers exited with code zero within a file as an argument to *comm*
#G - The output of the cat command is passed to the
#H - Both the output should be sorted for reliable consumption by the *comm* program

The above example is rather complicated, but shows the power you can get from combining different utilities together. It outputs (cat) the running container IDs (docker ps -q), and then outputs the container IDs of containers that are exited, but not those that have exited with status 0 (i.e. those that exited in an unexpected way). If you're struggling to follow it, then running each command separately and understanding that first can be a very helpful way to learn the building blocks.

Such a command could be useful when clearing out containers on production. You may want to adapt it to run a cron to clear out containers that exited in expected ways.

TIP **Make these one-liners available as commands**

You can add all these as aliases so that they're more easily run whenever you log into your host. To do this, add lines like the following to the end of your ~/.bashrc file:

```
alias dockernuke='docker ps -a -q | xargs --no-run-if-empty docker rm -f'
Then, when you next log in, running 'dockernuke' from the command line
will delete any docker containers found on your system.
```

This technique can save you a surprising amount of time! Be sure not to remove production containers this way. This is easily done, as the authors can attest…

Housekeeping volumes

Although volumes are a powerful feature of Docker, they come with a significant operational downside.

Because volumes can be shared between different containers they can't be deleted when a container that mounted them is deleted. Imagine the scenario outlined in the following figure:

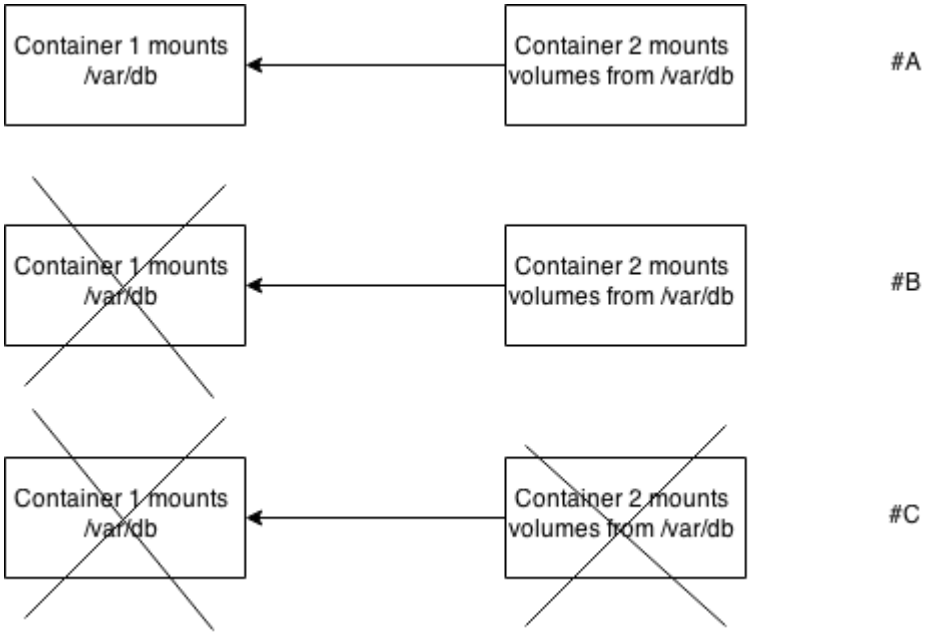

Figure 1 What happens to /var/db when we do simple docker rm's to remove containers?

#A - We have a normal setup where Container 1 has been started with a -v /var/db argument, and Container 2 has mounted the volumes from Container 1. Container 2 can access Container 1's /var/db/ volume
#B - Next container 1 is removed. /var/db in Container 2 persists, managed by the Docker daemon
#C - Now Container 2 is removed. /var/db remains on the filesystem

"Easy!" you might think, "just delete the volume when the last-referencing container is removed!" Indeed, that is an option that Docker could have taken, and this approach is the one that object oriented programming languages take when they remove objects from memory - when no other object references it, it can be deleted.

However, Docker judged that this could leave people open to losing valuable data accidentally, and preferred to make it a user decision whether a volume should be deleted on removal of the container. An unfortunate side effect of this means that by default, volumes remain on your Docker daemon's host disk until it is removed manually.

If these volues are full of data then your disk can get full of them. This technique explains approaches to managing these orphaned volumes.

Problem--You are using too much disk space because orphaned docker mounts exist in your host

Solution--Use the -v flag when calling docker rm, or use a script to destroy them if you forget

Discussion

In the scenario in Figure 1, we can ensure that /var/db is deleted if we always call docker rm with the -v flag. The -v flag removes any associated volumes if no other container still has it mounted. Fortunately it's smart enough to know whether any other container has the volume mounted, so there are no nasty surprises!

The simplest approach is to get into the habit of typing -v whenever you remove a container. That way you retain control of whether they are removed.

The problem with both of these approaches is that you might not want to do that! If you're writing a lot of data to these volumes, then it's quite likely that you don't want to lose that data. Additionally, if you get into such a habit then it's likely to become automatic and you'll only realise you've deleted something important when it's too late.

In these scenarios you'll need to use a script, which is - naturally - dockerized for your convenience!

Note that you'll need root permissions to run this.

```
$ docker run [#A]
-v /var/run/docker.sock:/var/run/docker.sock [#B]
-v /var/lib/docker:/var/lib/docker [#c]
--privileged dockerinpractice/docker-cleanup-volumes
```

#A - Mount the Docker server socket so we can call docker from within the container
#B - Mount the Docker directories so we can delete the orphaned volumes
#C - Escalate privileges so that we are able to delete the orphaned volumes

The above command will remove any volumes no longer accessed by any existing containers. The output will look like this:

```
[dockerinpractice@alberti /space/git/docker-cleanup-volumes]
$ docker run -v /var/run/docker.sock:/var/run/docker.sock
 -v /var/lib/docker:/var/lib/docker --privileged 951acdb777bf
```

```
Delete unused volume directories from /var/lib/docker/volumes
Deleting 659cfdc5d394ec7ad5942862ba5feb1d24c9f67ca314462207835ef5bf657131
In use 6ae01c5524267c8f01f1d1e83933b494fdb5c709d9468122b470bfcdd5a5b03d
Deleting 73260d192a0a4d0ebc3606d9daf7137ab220e41cbbfe919ef1dded01a2f37b29

Delete unused volume directories from /var/lib/docker/vfs/dir
Deleting 659cfdc5d394ec7ad5942862ba5feb1d24c9f67ca314462207835ef5bf657131
In use 6ae01c5524267c8f01f1d1e83933b494fdb5c709d9468122b470bfcdd5a5b03d
Deleting 73260d192a0a4d0ebc3606d9daf7137ab220e41cbbfe919ef1dded01a2f37b29
```

If you're nervous about running this and potentially deleting things you don't want to delete, you can call it with --dry-run at the end to prevent it from actually deleting anything.

That's how and why you need to stay on top of the containers and volumes on your host. Use these visual tools to keep your Docker environment clean and tidy.