How to clear Docker cache and free up space on your system

Written by
Kyle Galbraith

Published on
8 August 2023

Share

A breakdown of different Docker artifacts and build cache that can take up system resources like disk and how to prune them individually and use docker system prune to clear them all.

Docker persists build cache, containers, images, and volumes to disk. Over time, these things can take up a lot of space on your system. In this post, we'll look at the different Docker artifacts that can take up space on your system, how to clear them individually, and using `docker system prune` to clear Docker cache.

How much disk space is Docker using?

The first step is knowing the disk usage of Docker. We can use the `docker system df` command to get a breakdown of how much disk space is being taken up by various artifacts.

```shell
docker system df
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          138       34        36.18GB   34.15GB (94%)
Containers      74        18        834.8kB   834.6kB (99%)
Local Volumes   118       6         15.31GB   15.14GB (98%)
Build Cache     245       0         1.13GB    1.13GB
```

Docker uses 36.18GB for images, 834.8kB for containers, 15.31GB for local volumes, and 1.13GB for the docker build cache. About 50 GB of space in total, and a large chunk of it is reclaimable.

Removing containers

We can use the `docker container prune` command to clear the disk space used by containers. This command will remove all stopped containers from the system.

We can omit the `-f` flag here and in subsequent examples to get a confirmation prompt before artifacts are removed.

```shell
docker container prune -f
Deleted Containers:
399d7e3679bf9b14a1c7045cc89c056f2efe31d0a32f186c5e9cb6ebbbf42c8e

Total reclaimed space: 834.6kB
```

Which containers are unused?

We can see the ids of unused containers by running the `docker ps` command with filters on the status of the container. A container is unused if it's in status `exited` or `dead`.

```shell
docker ps --filter status=exited --filter status=dead -q
11bc2aa92622
355901f38ecb
263e9bde1f24
```

Note: If we want to know the size of the unused container, we can replace the `-q` flag with `-s` to get the size and other metadata about the container.

Removing all containers

If we want to remove *all* containers from the system, we can stop any running containers and then use the same prune command. We do this by feeding the output of `docker ps -q` into the `docker stop` command or `docker kill` command if you want to kill the container forcibly.

```shell
docker stop $(docker ps -q)
docker container prune
```

Another option is the `docker rm` command which can be used with `docker ps -a -q` to remove all containers.

```shell
docker rm $(docker ps -a -q)
```

Note: The `docker rm` command forces the removal of a running container via a `SIGKILL` signal. This is the same as the `docker kill` command. The `docker ps -a -q` command will list all containers on the system, including running containers, and feed that into the `docker rm` command.

Removing images

Docker images can take up a significant amount of disk space. We accumulate new images when base images change or build new ones via docker build, etc. We can use the `docker image prune` command to remove unused images from the system.

By default, it only removes dangling images. An image that is not associated with any container and doesn't have a tag.

```shell
docker image prune -f
Deleted Images:
deleted: sha256:6f096c9fa1568f7566d4acaf57d20383851bcc433853df793f404375c8d9
...

Total reclaimed space: 2.751GB
```

We reclaimed 2.7GB of space by removing dangling images. But, if we recall from our `docker system df` command, we have 34.15GB of reclaimable images.

Where is the rest of that space coming from? These are images on our system that are tagged or associated with a container. We can run the `docker image prune- a` command to force the removal of these images as well, assuming they're unused images.

```shell
docker image prune -a -f
Deleted Images:
untagged: k8s.gcr.io/etcd:3.4.13-0
untagged: k8s.gcr.io/etcd@sha256:4ad90a11b55313b182afc186b9876c8e891531b8db4
deleted: sha256:0369cf4303ffdb467dc219990960a9baa8512a54b0ad9283eaf55bd6c0ad
deleted: sha256:f3cecccfe2bea1cbd18db5eae847c3a9c8253663bf30a41288f541dc1470

Total reclaimed space: 22.66GB
```

We remove all unused images not associated with a container, not just the dangling ones.

Removing volumes

Volumes are never cleaned up automatically in Docker because they could contain valuable data. But, if we know that we no longer need the data in a volume, we can remove it with the

`docker volume prune` command. This removes all anonymous volumes not used by any containers.

```shell
docker volume prune -f
Total reclaimed space: 0B
```

Interestingly, we see that we didn't reclaim any space. This is because we have volumes that are associated with containers. We can see these volumes by running the `docker volume ls` command.

```shell
DRIVER    VOLUME NAME
local     0a44f085adc881ac9bb9cdcd659c28910b11fdf4c07aa4c38d0cca21c76d4ac4
local     0d3ee99b36edfada7834044f2caa063ac8eaf82b0dda8935ae9d8be2bffe404c
...
```

We get an output that shows the driver and the volume name. The command `docker volume prune` only removes anonymous volumes. These volumes are not named and don't have a specific source from outside the container. We can use the `docker volume rm -a` command to remove all volumes.

```shell
docker volume prune -a -f
Deleted Volumes:
c0c240b680d70fffef420b8699eeee3f0a49eec4cc55706036f38135ae121be0
2ce324adb91e2e6286d655b7cdaaaba4b6b363770d01ec88672e26c3e2704d9e

Total reclaimed space: 15.31GB
```

Removing build cache

To remove the Docker build cache, we can run the `docker buildx prune` command to clear the build cache of the default builder.

```shell
docker buildx prune -f
```

```
ID                                            RECLAIMABLE   SIZE        LAST
pw11qgl0xs4zwy533i2x61pef*                    true          54B         12 da
y37tt0kfwn1px9fnjqwxk7dnk                     true          0B          12 da
sq3f8r0qrqh4rniemd396s5gq*                    true          154.1kB     12 da

Total:  5.806GB
```

If we want to remove the build cache for a specific builder, we can use the `--builder` flag to specify the builder name.

```shell
docker buildx prune --builder builder-name -f
```

Removing networks

While Docker networks don't take up disk space on our machine, they do create network bridges, iptables, and routing table entries. So similarly to the other artifacts, we can remove unused networks with the `docker network prune` command to clean these up.

```shell
docker network prune -f
Deleted Networks:
test-network-1
```

Remove everything with `docker system prune`

We can remove all unused artifacts Docker has produced by running `docker system prune`. This will remove all unused containers, images, networks and build cache.

```shell
docker system prune -f
Deleted Images:
deleted: sha256:93477d5bde9ef0d3d7d6d2054dc58cbce1c1ca159a7b33a7b9d23cd1f

Deleted build cache objects:
```

```
6mm1daa19k1gdijlde3l2bidb
vq294gub98yx8mjgwila989k1
xd2x5q3s6c6dh5y9ruazo4dlm

Total reclaimed space: 419.6MB
```

By default, this command will not remove volumes and only removes dangling Docker images. We can use the `--volumes` flag to remove volumes as well. We can also add the `-a` flag again to remove all images not associated with a container.

```shell
docker system prune --volumes -a -f
```

Conclusion

Docker consists of different artifacts like containers, images, volumes, build cache, and even networks. These artifacts, over time, can chew up valuable resources, particularly disk space. It's handy to know how to use the individual `prune` commands and the `docker system prune` command to clean up these artifacts and free up disk space on your system.

Get started for free →

depot

PRODUCT

Pricing

Security

Drop Week #01

depot.ai

SUPPORT

Documentation

Support

System Status

COMPANY

About

Blog

Contact Us

Brand Assets

Terms of Service

Privacy Policy

Cookies