Container Images

A Docker Image is made of file-systems layered over each other. The base is a boot file-system `bootfs` similar to the one Linux uses. Followed by a root file-system `rootfs/kernel,` this holds the operative system layer, different from Linux. This one remains in read-only mode, and Docker uses the `UFS` union file-system to add more read-only file-systems on top of it.

> The union file-system `UFS` allows multiple file-systems to be mounted at once and appear as one, so the final visible mount will contain branches from all the merged file-systems.

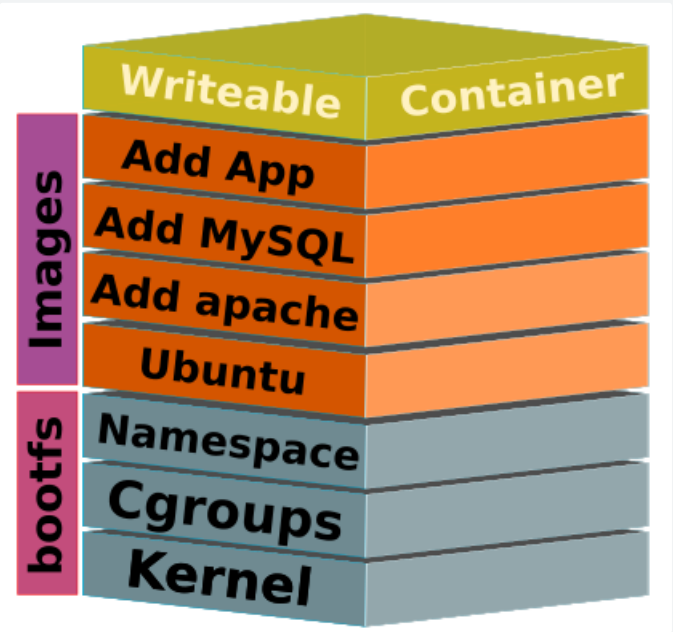

Fig. 1 - Container image layers and spaces

Docker calls each one of these layers images. The image below is called `parent image .` You can move around them until you get to the bottom, called `base image.` The Union file system mounts files as follows:

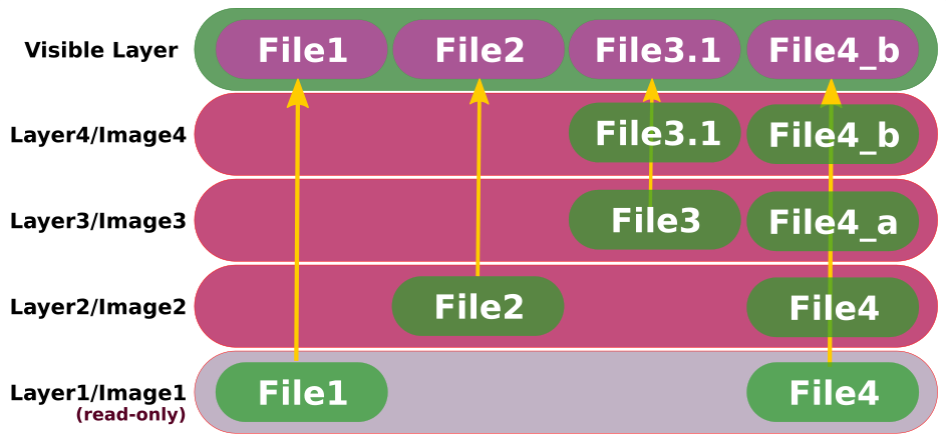

Fig. 2 - Overview on how files are managed on an image across multiple layers

Managing Images

Docker offers a series of commands to manage the images.

Command	Description
`docker build`	Build an image from a Dockerfile
`docker commit`	Create a new image from a container's changes
`docker history`	Show the history of an image
`docker images`	List images
`docker inspect`	Display detailed information on one or more images
`docker pull`	Pull an image or a repository from a registry
`docker push`	Push an image or a repository to a registry
`docker rmi`	Remove one or more images
`docker search`	Search the Docker Hub for images
`docker tag`	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
`docker image prune`	Remove unused images

To easily understand the options, let us separate in smaller groups:

- Exploring images
- Creating images
- Using repositories of images
- Maintaining images

Exploring images

listing

List all the images in the system.

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
ubuntu        latest    4e5021d210f6   15 months ago   64.2MB
fedora        latest    536f3995adeb   16 months ago   193MB
hello-world   latest    fce289e99eb9   2 years ago     1.84kB
```

searching

You can search images from Docker Hub with `search.`

```
$ docker search puppet
NAME                              DESCRIPTION
      STARS     OFFICIAL   AUTOMATED
puppet/puppetserver               A Docker Image for running Puppet Server.
Wi…   105
alekzonder/puppeteer              GoogleChrome/puppeteer image and
screenshots…   80                      [OK]
buildkite/puppeteer               A Puppeteer Docker image based on
Puppeteer'…   76                     [OK]
puppet/puppetdb                   A Docker image for running PuppetDB
      37
devopsil/puppet                   Dockerfile for a container with puppet
insta…   31                      [OK]
macadmins/puppetmaster            Simple puppetmaster based on CentOS 6
      26                     [OK]
puppet/puppetboard                The Puppet Board dashboard for PuppetDB
      19
puppet/puppet-agent-alpine        Puppet Agent as a Docker Image. Based on
Alp…   17
zenato/puppeteer-renderer         Puppeteer(Chrome headless node API) based
we…   15                       [OK]
```

The above will return matching strings found in:

- Repository names.
- Image Descriptions.
- Stars that measure the popularity of the image.
- Official images. Build by upstream developers.
- Automated images. Built automatically by Docker's processes.

history

If we want details on how an image was created, we can use the `history` command.

```
$ docker history 6f715d38cfe0
IMAGE          CREATED         CREATED BY
       SIZE      COMMENT
6f715d38cfe0   10 months ago   /bin/sh -c #(nop)  CMD ["nginx" "-g"
"daemon…   0B
<missing>      10 months ago   /bin/sh -c #(nop)  STOPSIGNAL SIGTERM
      0B
<missing>      10 months ago   /bin/sh -c #(nop)  EXPOSE 80
       0B
<missing>      10 months ago   /bin/sh -c #(nop)  ENTRYPOINT ["/docker-
entr…   0B
<missing>      10 months ago   /bin/sh -c #(nop) COPY
file:0fd5fca330dcd6a7…   1.04kB
<missing>      10 months ago   /bin/sh -c #(nop) COPY
file:1d0a4127e78a26c1…   1.96kB
<missing>      10 months ago   /bin/sh -c #(nop) COPY
file:e7e183879c35719c…   1.2kB
<missing>      10 months ago   /bin/sh -c set -x     && addgroup -g 101
-S …   16.5MB
<missing>      10 months ago   /bin/sh -c #(nop)  ENV PKG_RELEASE=1
       0B
<missing>      10 months ago   /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.3
       0B
```

```
<missing>      10 months ago   /bin/sh -c #(nop)  ENV NGINX_VERSION=1.19.2     0B
<missing>      10 months ago   /bin/sh -c #(nop)  LABEL maintainer=NGINX Do…   0B
<missing>      13 months ago   /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      13 months ago   /bin/sh -c #(nop) ADD file:c92c248239f8c7b9b…   5.57MB
```

inspect

Retrieves all the details of an image in `json` format.

```
 docker inspect 553092d64a86
[
    {
        "Id": "sha256:553092d64a8639e8477cff152781814ffdba18e138facb196a00ddb5f1d4f797",
        "RepoTags": [
            "helloworld:v0.1"
        ],
        "RepoDigests": [],
        "Parent": "sha256:111e3b6f89d672ab06a98dc6c0364c14fac21775eb0ac615767d2cce5a6de858",
        "Comment": "",
        "Created": "2021-06-15T13:32:10.091099185Z",
        "Container": "2ab8f300bc7809b326b61f522f8701996d4c7069f976527880a101de1b24c6c1",
        "ContainerConfig": {
            "Hostname": "2ab8f300bc78",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"/server\"]"
            ],
            "Image": "sha256:111e3b6f89d672ab06a98dc6c0364c14fac21775eb0ac615767d2cce5a6de858",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
```

```
        "DockerVersion": "20.10.2",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [
                "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/server"
            ],
            "Image": "sha256:111e3b6f89d672ab06a98dc6c0364c14fac21775eb0ac615767d2cce5a6de858",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": null
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 13524674,
        "VirtualSize": 13524674,
        "GraphDriver": {
            "Data": {
                "LowerDir": "/var/lib/docker/overlay2/78cf8106/diff:/var/lib/docker/overlay2/2905f573f5396/diff",
                "MergedDir": "/var/lib/docker/overlay2/c7ca02a9c40bcb7e5e4a3e14f0e934f1e8f97257935428/merged",
                "UpperDir": "/var/lib/docker/overlay2/c7ca02a9c40bcb7e5e856eb072df494f1e8f97257935428/diff",
                "WorkDir": "/var/lib/docker/overlay2/c7ca02a9c40bcb7e5e85f0e934f1e8f97257935428/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [
                "sha256:b2d5eeeaba3a22b9b8aa97261957974a6bd65274ebd43e1d81d0a7b8b752b116",
                "sha256:2e7bf52e9ddfdd54c35b778037706af69a5fc58c0614768b287ae4539ca958c9",
                "sha256:6b02de410e093db9c221e3db985c128521c229c8514f1d0444f1fb22f3018f1c"
```

```
            ]
        },
        "Metadata": {
            "LastTagTime": "2021-06-16T08:06:13.721130099-05:00"
        }
    }
]
```

Creating images

There are two ways to build images:

```
1. Via the `docker commit` command that takes an existing image and builds a new one from it.
2. Via the `docker build` command that builds an entirely new image using a `Dockerfile.`
```

commit

This is much like a commit in Git.

```
[Run a container]⟶[Apply your changes]⟶[Save them in a new image]
```

Run a container

```
$ docker run -ti ubuntu /bin/bash
root@23eec8f86583:/#
```

Apply your changes

```
# Installing apache as example from within the container
$ apt-get -yqq update; apt-get -y install apache2
...
# Verifying our changes
$ apt list apache2
Listing... Done
apache2/bionic-updates,bionic-security,now 2.4.29-1ubuntu4.16 amd64 [installed]

# Exiting the container
exit

# commit
docker commit 11425ae6f4ac jmedinar/apache2
```

Verifying the `ID` of the container

```
$ docker ps -lq
23eec8f86583
```

Save the container with the changes made into a new image using `commit.`

```
$ docker commit 23eec8f86583 jmedinar/my-new-apache
sha256:ca0cd033eee9965d89d4f438fb2fdf4fe4f6a127eb1fc75bea25f168007b3a4f
```

We can provide more data about the changes we are committing

```
$ docker commit -m "A new image with apache" -a "Prof Medina"
23eec8f86583 jmedinar/my-new-apache2
sha256:651d22f04dff45ef4acc5dd96bcdc1ae7e8a9bd027da731b3b150b11af3a0a1e
```

- `-m' Commit message
- `-a' Author of the image

We will see this information when we use the `inspect` command on the image created

We can view this information with inspect

```
$ docker inspect --format='{{json .Comment}}{{println}}{{json .Author}}'
jmedinar/my-new-apache2
"A new image with apache"
"Prof Medina"
```

We can run the container as follows:

```
$ docker run -ti jmedinar/my-new-apache2
root@cb2cb0ad383f:/# apt list apache2
Listing... Done
apache2/bionic-updates,bionic-security,now 2.4.29-1ubuntu4.16 amd64
[installed]
exit
```

> The `commit` is not recommended as building with a `Dockerfile` is more flexible and powerful.

build

You do not create an image from scratch but start from a base build from an upstream developer like RedHat using a Dockerfile that is much more flexible. This is a topic on its own that we will review in the next module.

tag

Docker tags are a simple way to label or differentiate our images from each other besides their names.

- By versioning.
- By setting a variant name.
- By flagging it.

Tags are aliases for the `ID` of your image.

You set them at the creation of your images when you are setting the name of the same as follows:

```
$ docker build -t username/image_name:tag_name .
```

This is just an **administration feature** that gives us more control.

You can also explicitly tag an existing image.

```
$ docker tag jmedinar/my-new-apache2 jmedinar/my-new-apache2:version2
```

So it will look similar to this.

```
$ docker images | grep my-new-apache2
jmedinar/my-new-apache2         latest          651d22f04dff   19 minutes ago   203MB
jmedinar/my-new-apache2         version2        651d22f04dff   19 minutes ago   203MB
```

Notice how by default, when you do not specify a tag, your image is automatically tagged as `latest.`

One more thing to consider is that you can have multiple tags.

```
$ docker tag jmedinar/my-new-apache2 jmedinar/my-new-apache2:prod
$ docker tag jmedinar/my-new-apache2 jmedinar/my-new-apache2:do-not-run
```

So now it will look similar to this.

```
$ docker images | grep my-new-apache2
jmedinar/my-new-apache2         do-not-run      651d22f04dff   26 minutes ago   203MB
jmedinar/my-new-apache2         latest          651d22f04dff   26 minutes ago   203MB
jmedinar/my-new-apache2         prod            651d22f04dff   26 minutes ago   203MB
jmedinar/my-new-apache2         version2        651d22f04dff   26 minutes ago   203MB
```

Using repositories of images

Images live inside repositories, and these live inside registries.

Registry

The registry is stored to control and distribute images.

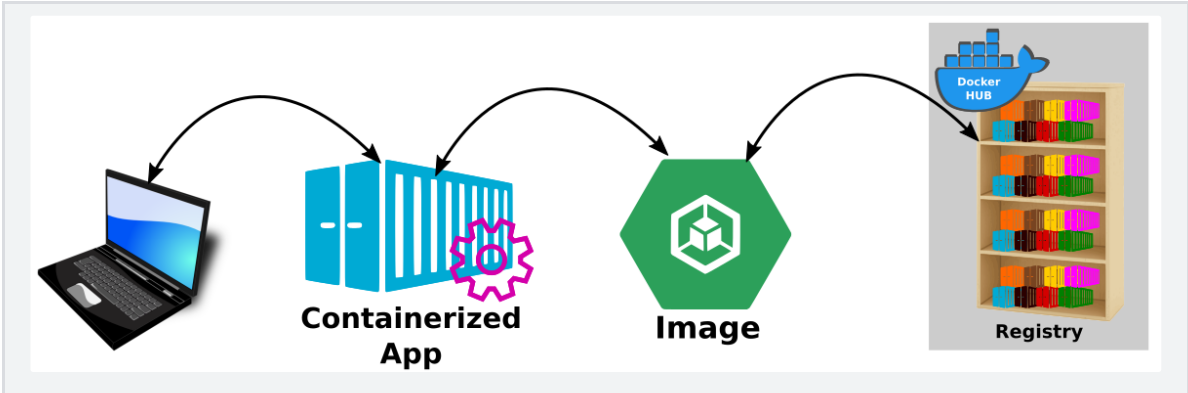

Fig. 3 - Overview of the proccess to upload a new created image to a registry.

The default registry is the Public `Docker Hub` .

We use `tags` to identify the images in the registry because they allow us to have multiple images of the same OS.

There are two types of repositories:

1. **User Repositories:** Images contributed by Docker users. (Use them with caution!!!).
2. **Top-Level Repositories:** Controlled by Docker.

Refer to a user repository by adding the name of the user, the image name, and possible tags.

```
jmedinar/puppet:1.0
```

Login

Register for a Docker ID

Before you can start using the services of Docker, you have to register to get a `Docker ID.`

1. Go to the Docker Hub signup page.
2. Enter a username that is also your Docker ID.

Your Docker ID must be between 4 and 30 characters long and only contain numbers and lowercase letters.

1. Enter a unique, valid email address.
2. Enter a password. Note that the password must be at least 9 characters.
3. Complete the Captcha verification and then click **Sign up**.

   > Docker sends a verification email to the address you provided.

4. Verify your email address to complete the registration process.

   > **Note:** You cannot log in with your Docker ID until verifying your email address.

```
$ docker login
Username: jmedinar
Password:

Login Succeeded
```

Your credentials will be stored at $HOME/.docker/config.json, so you do not have to type them every time you try to connect.

logout

Remember to log out every time you are in a shared environment or a public computer.

```
$ docker logout
```

pull

Will download images if they are not already present in the host. This saves time when launching containers.

```
$ Docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
c549ccf8d472: Pull complete
Digest:
sha256:aba80b77e27148d99c034a987e7da3a287ed455390352663418c0f2ed40417fe
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

push

Once you complete creating and testing your image, you are ready to make it public if you want to by pushing it to DockerHub, making it available for other users.

DockerHub also has private repositories as a paid feature.

```
$ docker push jmedinar/my-new-apache2
Using default tag: latest
The push refers to repository [docker.io/jmedinar/my-new-apache2]
fc02d4dcd423: Pushed
e80c789bc6ac: Mounted from jmedinar/my_custom_app
6c3332381368: Mounted from jmedinar/my_custom_app
ef1a1ec5bba9: Mounted from jmedinar/my_custom_app
a1aa3da2a80a: Mounted from jmedinar/my_custom_app
latest: digest:
sha256:0998da23dae955b5edd96a9f8c0b7f2a9e441a8cd3716585ccf9803362d249fd
size: 1364
```

Testing the new registry

Let us list our images **Not the local ones but the ones in the repository**

```
$ docker search jmedinar
NAME                                DESCRIPTION   STARS     OFFICIAL
AUTOMATED
jmedinar/kubia-unhealthy                          0
jmedinar/kubia                                    0
jmedinar/fortune                                  0
jmedinaresolv/resuelve-prueba-zero                0
jmedinar/my_custom_app                            0
jmedinar/my-new-apache2                           0
```

I have deleted all the local versions of the image to test this works, so now we can build containers using this image from the registry.

```
$  docker run -ti jmedinar/my-new-apache2
Unable to find image 'jmedinar/my-new-apache2:latest' locally
latest: Pulling from jmedinar/my-new-apache2
5667fdb72017: Already exists
d83811f270d5: Already exists
ee671aafb583: Already exists
7fc152dfb3a6: Already exists
b3e3e1897a2a: Already exists
Digest:
sha256:0998da23dae955b5edd96a9f8c0b7f2a9e441a8cd3716585ccf9803362d249fd
Status: Downloaded newer image for jmedinar/my-new-apache2:latest
root@a3a67f5dd505:/#
```

Maintaining images

rmi

You can delete one or more images.

```
$ # docker images jmedinar/my-new-apache2
REPOSITORY                TAG       IMAGE ID       CREATED       SIZE
jmedinar/my-new-apache2   latest    651d22f04dff   7 hours ago   203MB
jmedinar/my-new-apache2   new       651d22f04dff   7 hours ago   203MB

[root@oc4083742478 ~]# docker rmi jmedinar/my-new-apache2:latest
jmedinar/my-new-apache2:new
Untagged: jmedinar/my-new-apache2:latest
Error response from daemon: conflict: unable to remove repository
reference "jmedinar/my-new-apache2:new" (must force) - container
cb2cb0ad383f is using its referenced image 651d22f04dff
```

If a running container is using the image, you will not delete it unless you stop the container or use the force option `-f` .

```
$ docker rmi -f jmedinar/my-new-apache2:latest jmedinar/my-new-
apache2:new
Untagged: jmedinar/my-new-apache2:new
Untagged: jmedinar/my-new-
apache2@sha256:0998da23dae955b5edd96a9f8c0b7f2a9e441a8cd3716585ccf9803362
d249fd
Deleted:
sha256:651d22f04dff45ef4acc5dd96bcdc1ae7e8a9bd027da731b3b150b11af3a0a1e
```

> This only deletes the image **locally** NOT from `DockerHub` .

image prune

The image prune command will automatically delete `ALL dangling` images and, if used with the `-a' option, delete` ALL unused` images.

Remember that images are created in layers!. So you might have many layers from old images not currently being used in any of the images in the system.

Let us quickly count the total of images in the system.

```
$ docker images | wc -l
101
```

Now let us prune

```
$ docker image prune
WARNING! This will remove all dangling images.
Are you sure you want to continue? [y/N] y
Deleted Images:
deleted:
sha256:09da1ee7e34d06bb9e839c9f5a921f9733b4e80ee1d1e93a371917f916de0ed9
deleted:
sha256:ae7f8e83170101b0a82f2f8c5e964f54d1e16251ca239874e36a4590403058fd
deleted:
sha256:007652d877a230dfe01ef943646f795750798312ad93512c72918498639e90d4
deleted:
sha256:1b1b94b193bfb364b171ef6a216a0a333698a701f699553dfb9beaa2129ee849
deleted:
sha256:296ec2ab21cdac2378d096c33b7bf9d526eeff30c413ce1ab1b03f26496ce0db
deleted:
sha256:0fe050bc73e59adfe469d48b9bc1059c1e1854307ab94e4e2ae296027420d3cc

Total reclaimed space: 3.236GB
```

Let us see how many images we have now.

```
$ docker images | wc -l
101
```

The same amount? Yes, because dangling images are only those intermediate layers that are not being used, we still released some room in our local computer.

Let us now prune with the `-a' option.

DANGER! This will delete anything you are not using, so use it carefully!

```
$ docker image prune -a
WARNING! This will remove all images without at least one container
associated with them.
Are you sure you want to continue? [y/N] y
Deleted Images:
...
...
...
Total reclaimed space: 12.32GB
```

Now let us count our images again.

```
$ docker images | wc -l
37
```