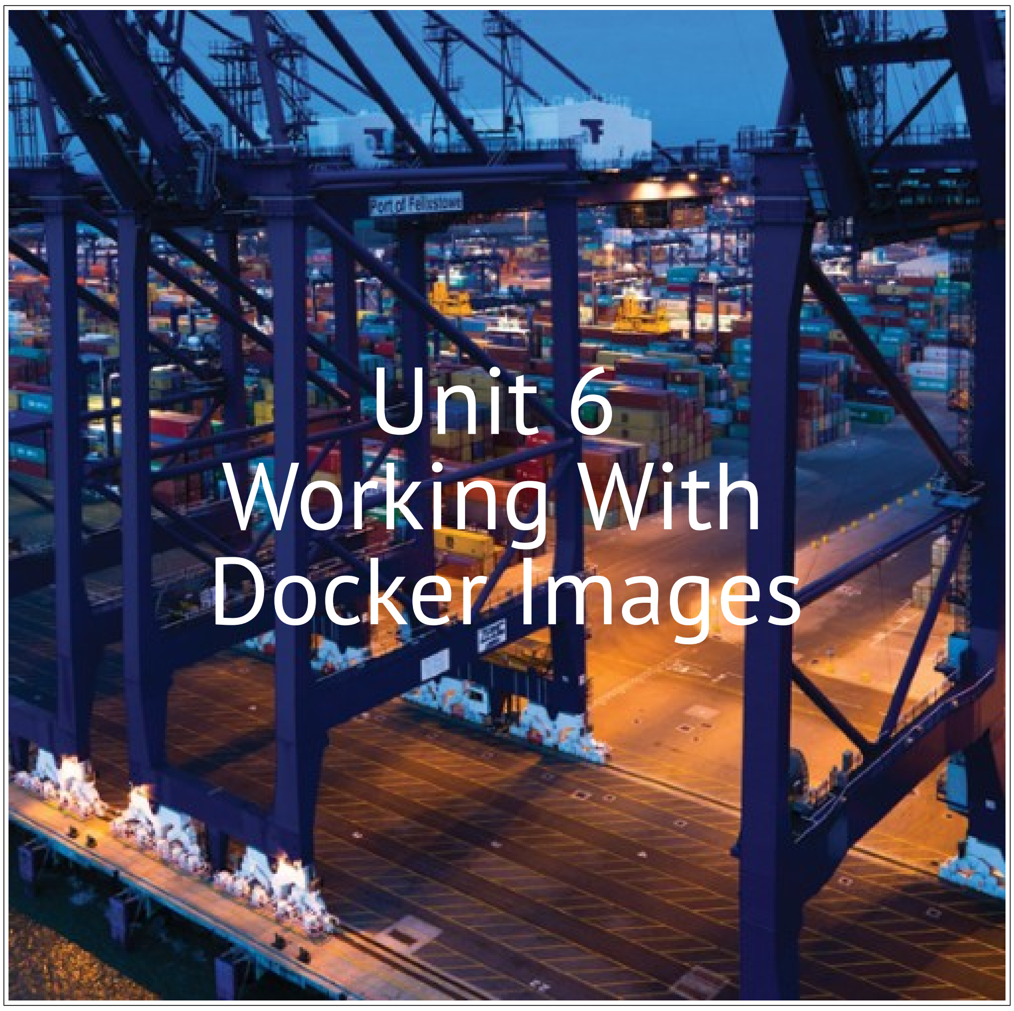


Unit 6 Working With Docker Images

Objectives

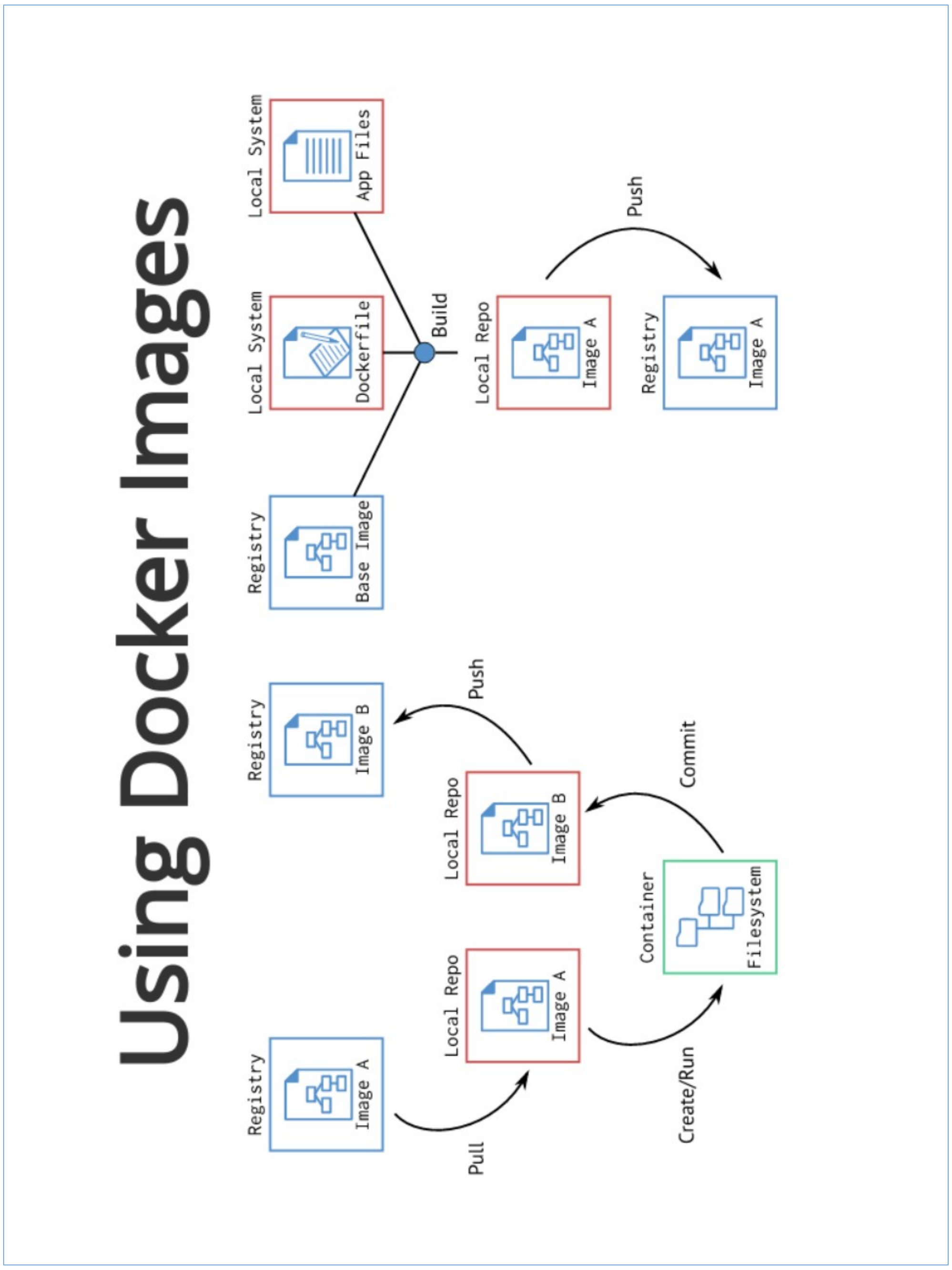
Upon completion of this unit, you will be able to:

- Explain how to find and use relevant Docker images
- Create new Docker images based on a container's filesystem
- Determine the composition of Docker images
- Describe how to manage Docker images and their characteristics

Agenda

The following topics will be covered in this unit:

- Finding, using and pulling Docker images
- Creating images from containers
- Retrieving image information
- Managing image sizes
- Archiving images
- Removing and tagging images
- Lab exercise: Working With Docker Images


Using Docker Images

Using Docker Images

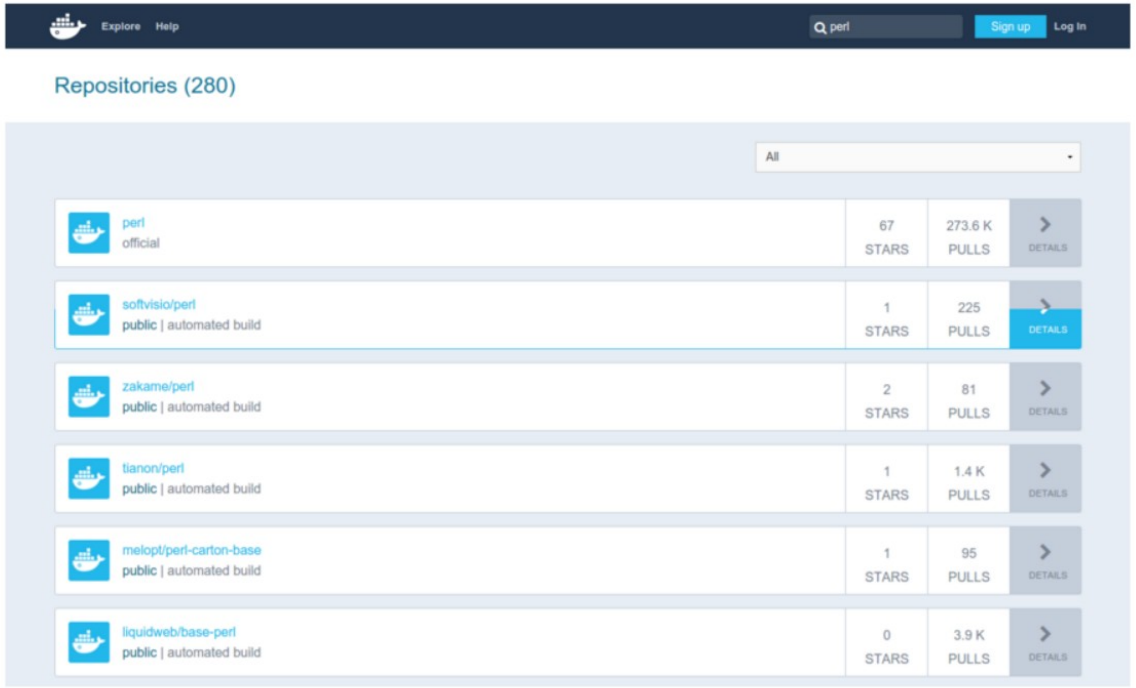
- Docker uses images from which to create a functioning container, rather like a template
- Docker images are held in the repositories of Docker registries, like the Docker Hub
- Publicly available images can be used by anyone, and the community encourages people to publish their images to the Docker Hub, in order for others to benefit
- Relevant images can be searched for using the Docker client CLI, the Docker Registry API, or on the Docker Hub registry website
- In order to use a pre-existing Docker image, it must be 'pulled' from the relevant registry, either explicitly using the `docker pull` command, or implicitly using the `docker create` or `docker run` commands
- Pulling Docker images from a registry prior to their use, significantly speeds up container start times
- New Docker images can be created in one of two ways:

    - run a container from an existing image, make the necessary changes, and then commit the changes to a new image, or
    - build an image from a base image using a Dockerfile

- Newly created Docker images will reside in the Docker host's local repository, but can be shared by 'pushing' the image to a Docker registry

Notes

Search for Docker Images



Repositories (280)

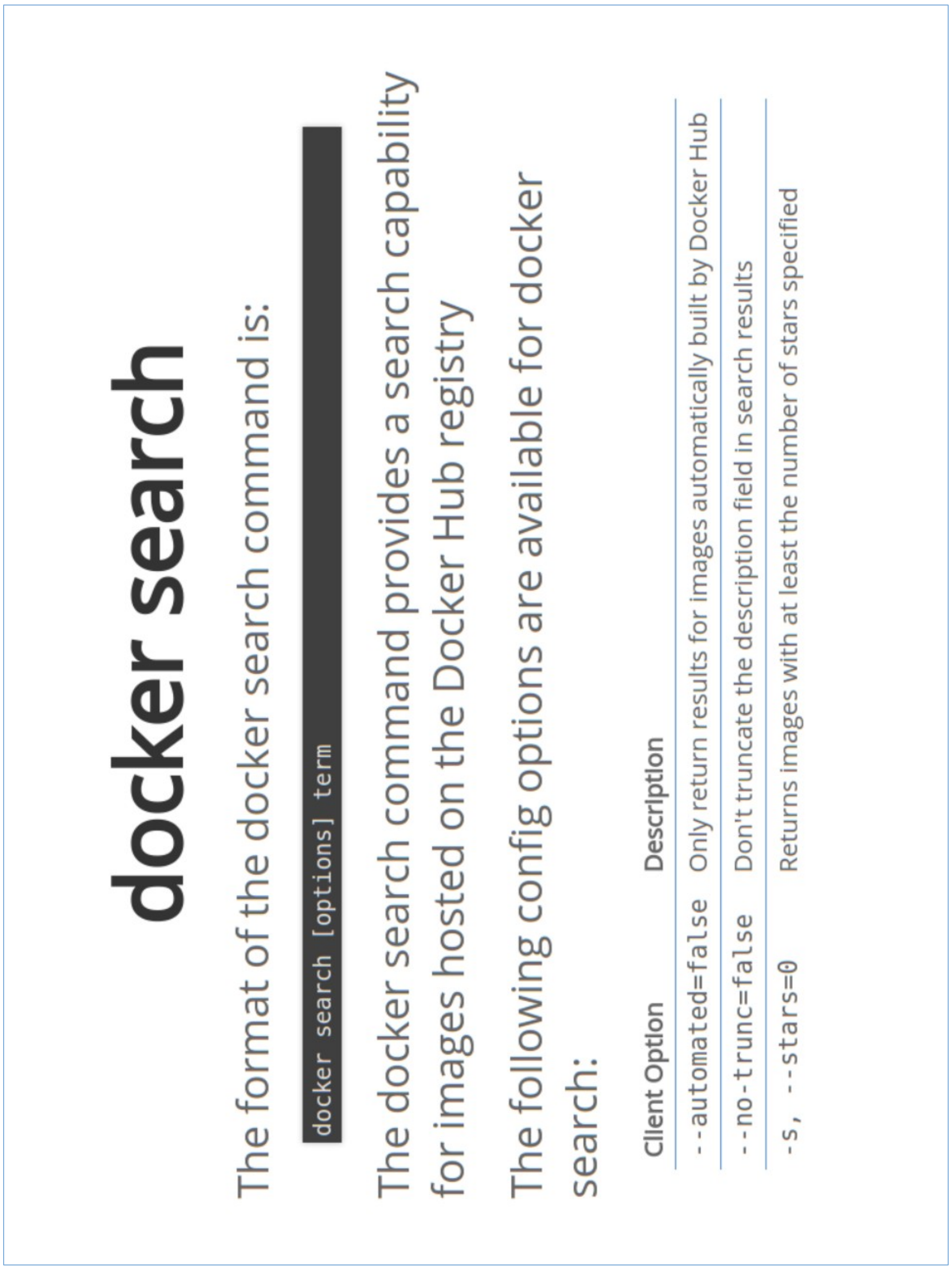
All

perl official	67 STARS	273.6 K PULLS	DETAILS
softvisio/perl public | automated build	1 STARS	225 PULLS	DETAILS
zakame/perl public | automated build	2 STARS	81 PULLS	DETAILS
tianon/perl public | automated build	1 STARS	1.4 K PULLS	DETAILS
melopt/perl-carton-base public | automated build	1 STARS	95 PULLS	DETAILS
liquidweb/base-perl public | automated build	0 STARS	3.9 K PULLS	DETAILS

Search for Docker Images

- The Docker Hub website is located at `https://registry.hub.docker.com`
- Docker and the community provide a library of official images, which can be found at `https://github.com/docker-library/official-images`
- The Docker Hub registry provides a search capability, which returns a list of images that contain the search string in the image name or its description
- The search results can be optionally filtered according to whether the images are official, or whether they were built automatically by the Docker Hub
- The search results can be sorted, based on the number of stars attributed to an image, or the number of downloads
- Each image listing provides metadata concerning the image; for example, whether the image is official, public or private (private images are not shown unless you are logged in and have permission to access the image)

Notes



docker search

The format of the docker search command is:

```
docker search [options] term
```

The docker search command provides a search capability for images hosted on the Docker Hub registry

The following config options are available for docker search:

Client Option	Description
`--automated=false`	Only return results for images automatically built by Docker Hub
`--no-trunc=false`	Don't truncate the description field in search results
`-s, --stars=0`	Returns images with at least the number of stars specified

docker search

- The Docker client CLI can also be used to search for images with the `docker search` command
- The format of the `docker search` command is:

```
docker search [options] term
```

- To return all images on the Docker Hub registry that have at least one star, and contain the search string 'smtp':

```
$ docker search -s 100 smtp
```

- Docker doesn't yet allow for the searching of images within a private registry, but this can be achieved using the Docker registry API*

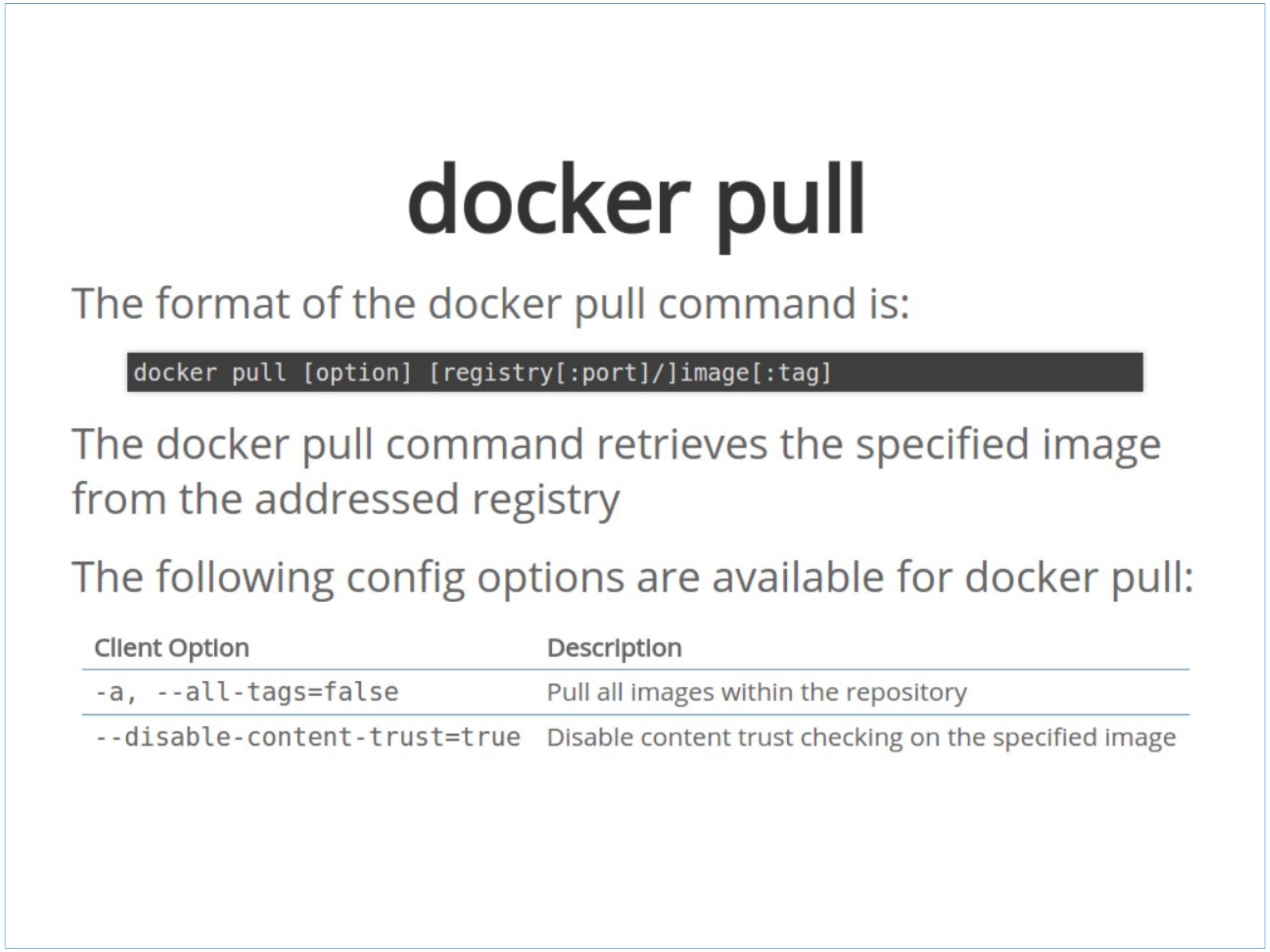
```
curl https://myregistry.com/v1/search?q=smtp
```

- The configuration options available for the `docker search` command are:

`--automated=false`: only lists those images that match the search term, and that are built automatically by the Docker Hub
`--no-trunc=false`: doesn't truncate the description field of the search results
`-s, --stars=0`: only lists those images that match the search term, and have at least the number of stars specified

* Currently, searching is only available in V1 of the registry

Notes



docker pull

The format of the docker pull command is:

```
docker pull [option] [registry[:port]/]image[:tag]
```

The docker pull command retrieves the specified image from the addressed registry

The following config options are available for docker pull:

Client Option	Description
`-a, --all-tags=false`	Pull all images within the repository
`--disable-content-trust=true`	Disable content trust checking on the specified image

docker pull

- Once a Docker image has been identified for use, it can be downloaded to a local repository on the Docker host, using the `docker pull` command
- The format of the `docker pull` command is:

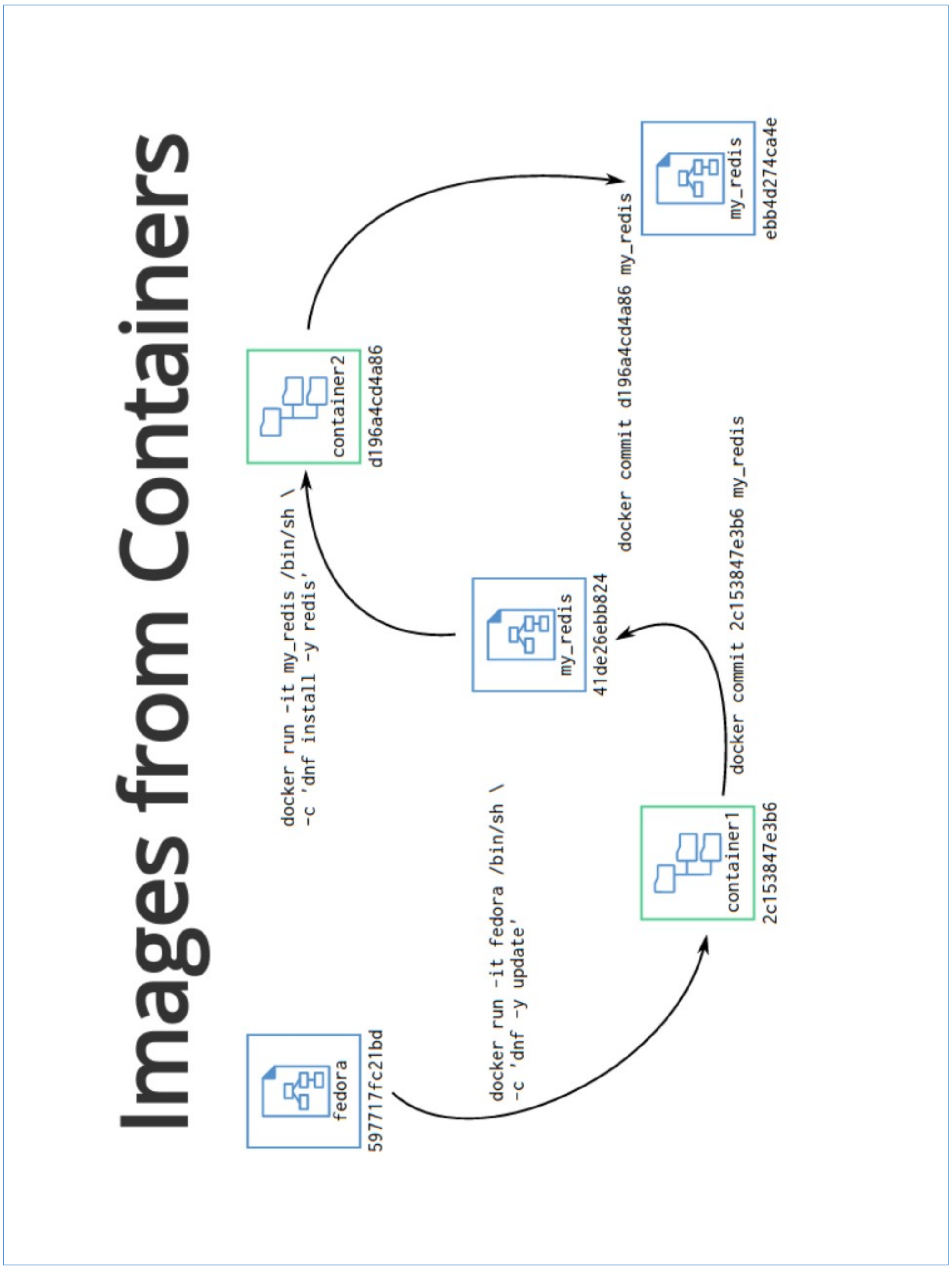
```
docker pull [option] [registry[:port]/]image[:tag]
```

- Before downloading an image, Docker checks to see whether the image already resides in a local repository on the Docker host
- If it already resides in a local repository, Docker checks with the registry to determine whether the image is the most up to date version, and downloads a newer version if one exists
- Repositories contain sets of related images, so unless a specific tag is provided as part of the image name during a pull, Docker assumes ‘latest’
- The configuration options available for the `docker pull` command are:

`-a, --all-tags=false`: specifies that all of the tagged images in the repository should be downloaded
`--disable-content-trust=true`: specifies whether content trust operations should be applied to the command

- With `--disable-content-trust=false`, the Docker Engine only allows the CLI to pull images whose tags have been digitally signed, or those pulled with a valid content hash
- Use of the docker pull command line option `-a`, `--all-tags`, and specifying a tag as part of the image name, are mutually exclusive
- Docker checks whether the latest image already resides in the local repository, and if not; downloads each layer of the image in parallel, expands the tar archive into the runtime storage area, and displays the digest associated with the image's manifest on completion

Notes



Images from Containers

```
docker run -it fedora /bin/sh \
-c 'dnf -y update'
```

```
docker commit 2c153847e3b6 my_redis
```

```
docker run -it my_redis /bin/sh \
-c 'dnf install -y redis'
```

```
docker commit d196a4cd4a86 my_redis
```

Creating Images from Containers

- New images can be created from a running container using the `docker commit` command
- By way of example, let's install the key/value store server, Redis, on top of a Fedora base image, in order to create a new image
- First, a container needs to be run from an existing Fedora image, e.g.

```
$ docker run -it fedora /bin/sh -c 'dnf -y update'
```

- The container runs and executes the Fedora repo update, which results in the container's filesystem being updated
- To commit this change to a new image using `docker commit`, we first need to determine the container ID to pass to the command:

```
$ docker ps -lq
2c153847e3b6
$ docker commit 2c153847e3b6 my_redis
```

- The command returns the ID of the new image, which is saved in the local repository
- The next step is to install Redis using another container based on our new image:

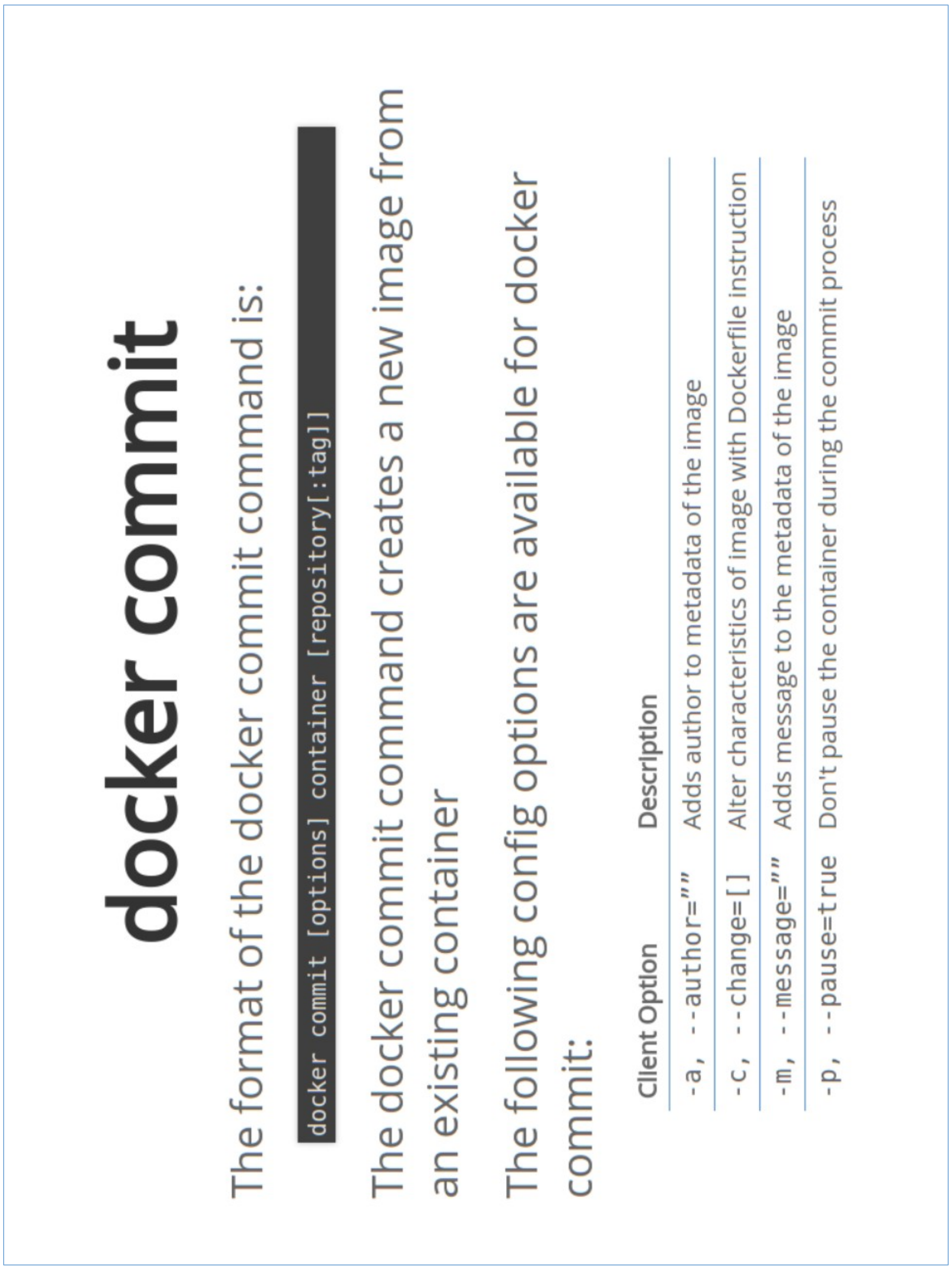
```
$ docker run -it my_redis /bin/sh -c 'dnf install -y redis'
```

- A new container is created based on our recently created image, and executes an install of the Redis server, which updates the container's filesystem again
- Once again, we can commit the change, but this time to our recently created image:

```
$ docker ps -lq
d196a4cd4a86
$ docker commit d196a4cd4a86 my_redis
```

- Docker returns the ID of the newer image, which is given the tag 'latest'

Notes



docker commit

The format of the docker commit command is:

```
docker commit [options] container [repository[:tag]]
```

The docker commit command creates a new image from an existing container

The following config options are available for docker commit:

Client Option	Description
`-a, --author=""`	Adds author to metadata of the image
`-c, --change=[]`	Alter characteristics of image with Dockerfile instruction
`-m, --message=""`	Adds message to the metadata of the image
`-p, --pause=true`	Don't pause the container during the commit process

docker commit

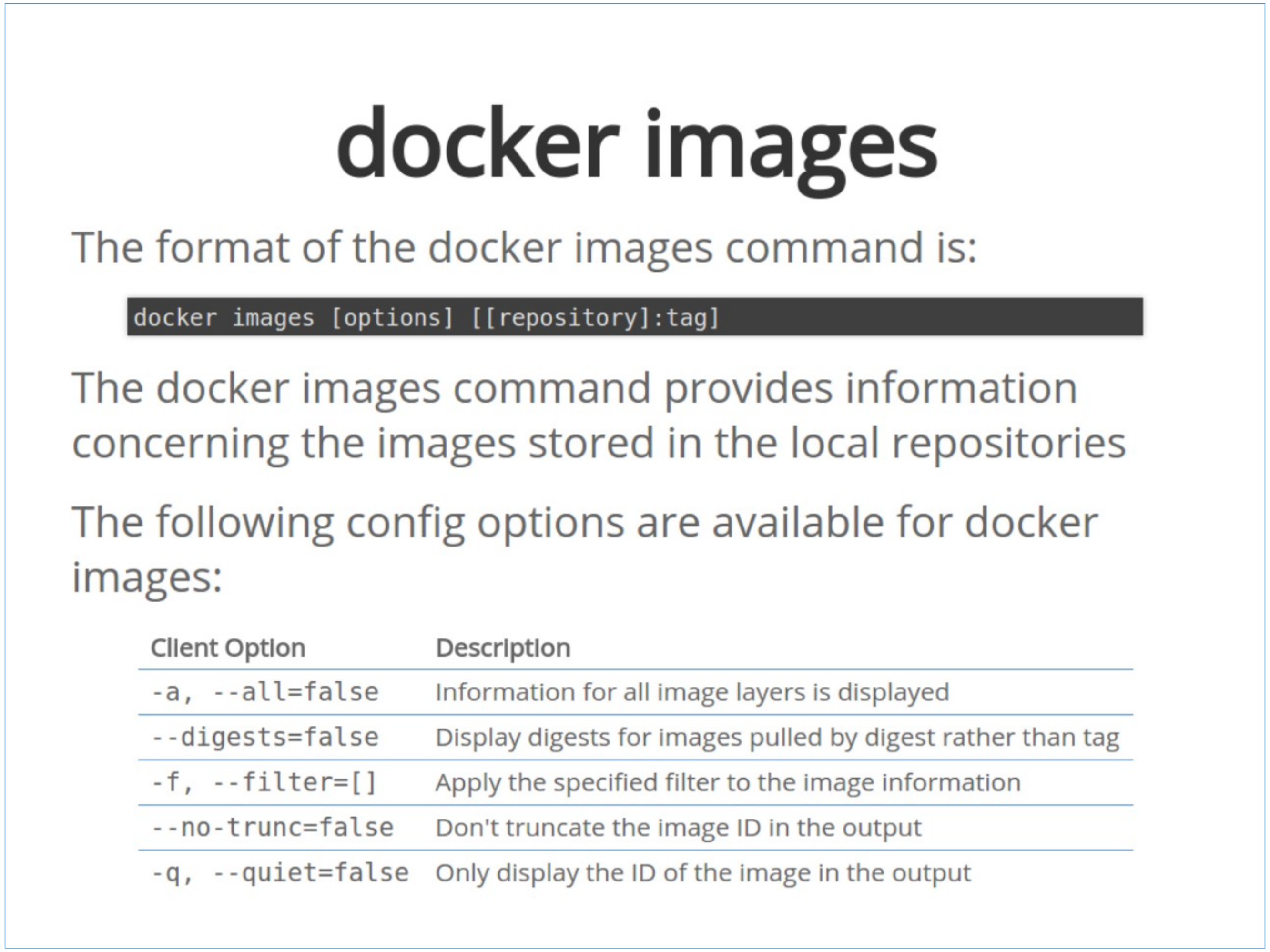
- The format of the `docker commit` command is:

```
docker commit [options] container [repository[:tag]]
```

- The configuration options available for the `docker commit` command are:

`-a, --author=””`: specifies the author of the images, stored as part of the image's metadata, which can be retrieved using the `docker inspect` command
`-c, --change=[]`: change characteristics of the image with specified Dockerfile instruction, which can be one of `CMD`, `ENTRYPOINT`, `ENV`, `EXPOSE`, `LABEL`, `ONBUILD`, `USER`, `VOLUME`, `WORKDIR`
`-m, --message=””`: specifies a message specific to the commit, which can be retrieved using the `docker inspect` command
`-p, --pause=true`: specifies whether the container’s execution should be paused during the commit

Notes



docker images

The format of the docker images command is:

```
docker images [options] [[repository]:tag]
```

The docker images command provides information concerning the images stored in the local repositories

The following config options are available for docker images:

Client Option	Description
`-a, --all=false`	Information for all image layers is displayed
`--digests=false`	Display digests for images pulled by digest rather than tag
`-f, --filter=[]`	Apply the specified filter to the image information
`--no-trunc=false`	Don't truncate the image ID in the output
`-q, --quiet=false`	Only display the ID of the image in the output

docker images

- It's possible to list all of the images that reside in the local repositories of the Docker host, using the `docker images` command
- With none of the optional configuration options applied, Docker simply lists the top-most image layers or leaves of the image tree
- For each image in the local repositories, Docker lists the repository name, the tag (which differentiates one branch from another), the ID of the image, when it was created, and the cumulative size of the image layers
- A list of all the images (layers) that reside in the local repositories can be listed using the `-a`, `--all` command line option, but those images that are intermediate layers do not have a repository name or tag associated with them
- The format of the `docker images` command is:

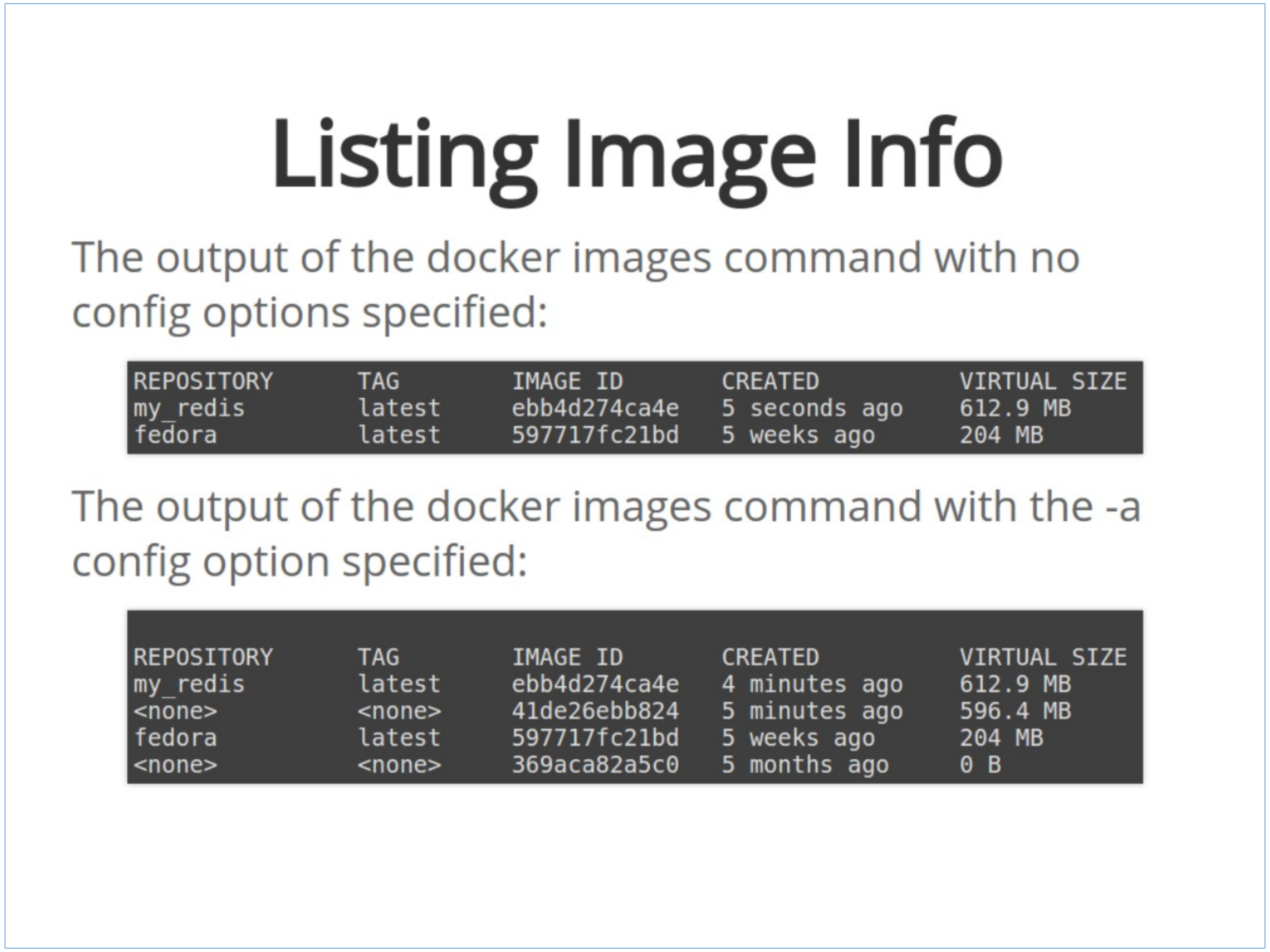
```
docker images [options] [[repository]:tag]
```

- The configuration options available for the `docker images` command are:

`-a, --all=false`: specifies that all images (layers) that reside in the local repositories should be listed
`--digests=false`: specifies that the digests for images should be displayed (only applies where images were pulled by digest as opposed to tag)
`-f, --filter=[]`: specifies a filter to apply to the output based on a key/value pair; available filters are `dangling=true` and `label=key[=value]`
`--no-trunc=false`: specifies that the output of the listed images does not truncate the image's ID
`-q, --quiet=false`: specifies that the output of the listed images should only contain the ID of each image, which is useful when chaining commands together

- Dangling images are those that represent leaf images, but do not have an associated repository and tag

Notes



Listing Image Info

The output of the docker images command with no config options specified:

```
REPOSITORY          TAG         IMAGE ID        CREATED          VIRTUAL SIZE
my_redis            latest      ebb4d274ca4e    5 seconds ago    612.9 MB
fedora              latest      597717fc21bd    5 weeks ago      204 MB
```

The output of the docker images command with the -a config option specified:

```
REPOSITORY          TAG         IMAGE ID        CREATED          VIRTUAL SIZE
my_redis            latest      ebb4d274ca4e    4 minutes ago    612.9 MB
<none>              <none>      41de26ebb824    5 minutes ago    596.4 MB
fedora              latest      597717fc21bd    5 weeks ago      204 MB
<none>              <none>      369aca82a5c0    5 months ago     0 B
```

Listing Image Information

- The following is the output of the `docker images` command with no command line options specified:

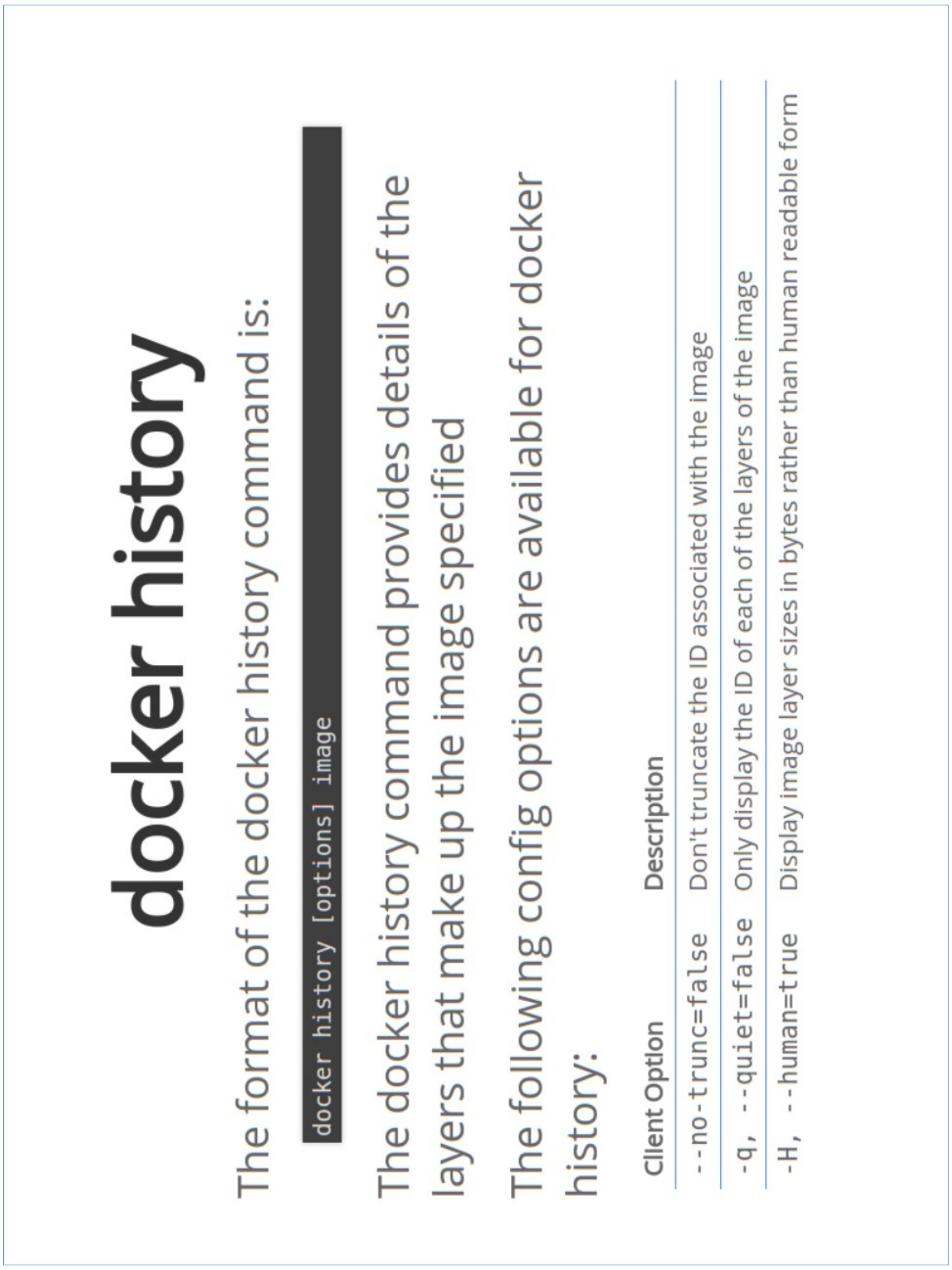
```
$ docker images
REPOSITORY       TAG       IMAGE ID       CREATED          VIRTUAL SIZE
my_redis         latest    ebb4d274ca4e   5 seconds ago    612.9 MB
fedora           latest    597717fc21bd   5 weeks ago      204 MB
```

- The following is the output of the `docker images` command with the `-a`, `--all` command line option specified:

```
$ docker images -a
REPOSITORY       TAG       IMAGE ID       CREATED          VIRTUAL SIZE
my_redis         latest    ebb4d274ca4e   4 minutes ago    612.9 MB
<none>           <none>    41de26ebb824   5 minutes ago    596.4 MB
fedora           latest    597717fc21bd   5 weeks ago      204 MB
<none>           <none>    369aca82a5c0   5 months ago     0 B
```

- The images that have `<none>` specified for their repository and tag, are branches and can be found in branches of multiple, completely unrelated repositories (e.g. `library/mysql`), whereas those with a string in the repository and tag fields represent leaves of the tree
- Some images have an accumulated virtual size of `0` bytes; and this is because when the layer was added, no files were added, and previous layers also had no files added

Notes



docker history

The format of the docker history command is:

```
docker history [options] image
```

The docker history command provides details of the layers that make up the image specified

The following config options are available for docker history:

Client Option	Description
`--no-trunc=false`	Don't truncate the ID associated with the image
`-q, --quiet=false`	Only display the ID of each of the layers of the image
`-H, --human=true`	Display image layer sizes in bytes rather than human readable form

docker history

- In addition to the facilities provided by the `docker images` and `docker inspect` commands, it's also possible to glean some historical information relating to an image, using the `docker history` command
- The `docker history` command provides a listing of the individual images (layers) in the image branch specified, providing the image's ID, when it was created, the command used to create the image, and the size of the image (as opposed to the cumulative size)
- The format of the `docker history` command is:

```
docker history [options] image
```

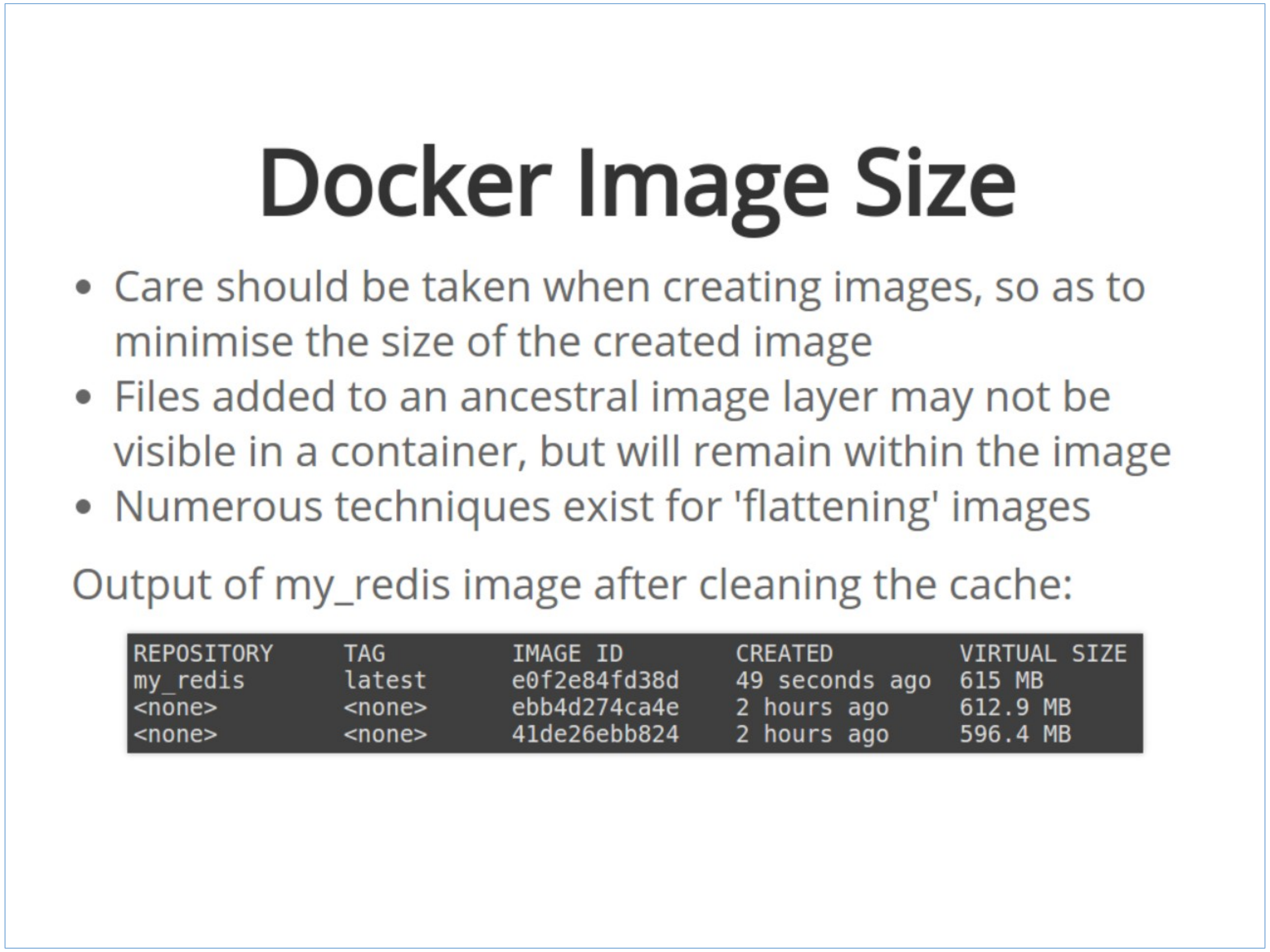
- To list our recently created Redis image, we can use the command below, which provides information showing that the first two layers are those of the `fedora:latest` image, and the final two layers being those we added to create our Redis image:

```
$ docker history my_redis
IMAGE        CREATED            CREATED BY                                     SIZE
ebb4d274ca4e 25 seconds ago     sh -c dnf install -y redis                     16.57 MB
41de26ebb824 About a minute ago sh -c dnf update -y                            392.4 MB
597717fc21bd 5 weeks ago        /bin/sh -c #(nop) ADD file:f867e28f7b12f9d64d  204 MB
369aca82a5c0 5 months ago       /bin/sh -c #(nop) MAINTAINER Adam Miller <max  0 B
```

- The configuration options available for the `docker history` command are:

`--no-trunc=false`: specifies that the output of the image history does not truncate the image's ID
`-q, --quiet=false`: specifies that the output of the image history should only contain the ID of each image
`-H, --human=true`: specifies that images sizes are displayed in human readable format

Notes



Docker Image Size

- Care should be taken when creating images, so as to minimise the size of the created image
- Files added to an ancestral image layer may not be visible in a container, but will remain within the image
- Numerous techniques exist for 'flattening' images

Output of my_redis image after cleaning the cache:

```
REPOSITORY      TAG         IMAGE ID        CREATED           VIRTUAL SIZE
my_redis        latest      e0f2e84fd38d    49 seconds ago    615 MB
<none>          <none>      ebb4d274ca4e    2 hours ago       612.9 MB
<none>          <none>      41de26ebb824    2 hours ago       596.4 MB
```

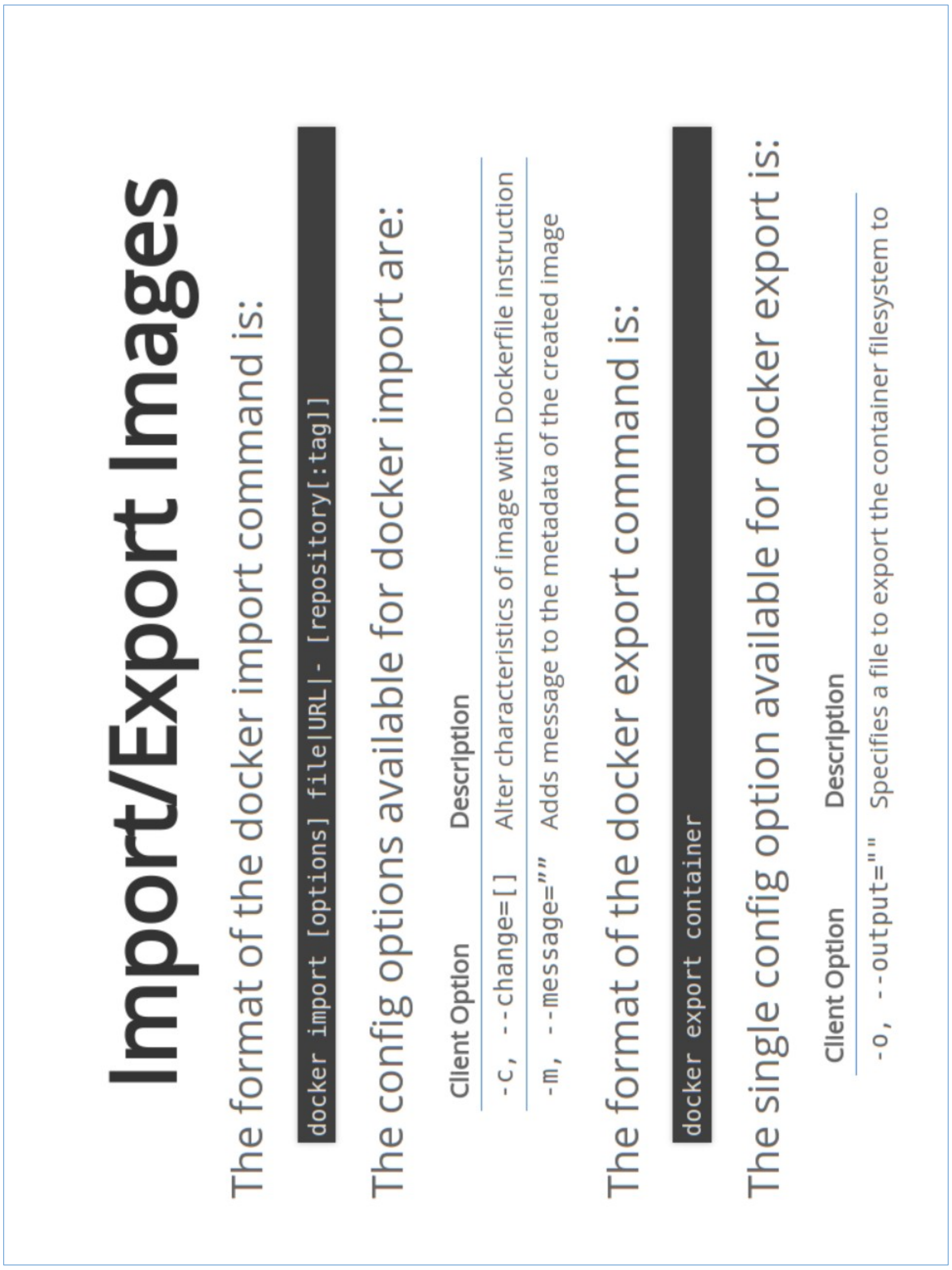
Docker Image Size

- Care should be taken not to make Docker images unnecessarily large; after all, one of the main principals behind Docker is leanness
- Is it necessary to provide a complete Linux distribution for a container's filesystem, minimalist as it may be?
- In theory, all that is required are the binaries, libraries and configuration files that the container's process needs to execute
- The very same Copy-On-Write (COW) capability that gives Docker one of its most attractive benefits with the sharing of images between containers, also provides a significant disadvantage if care is not taken
- Files that are required as part of building a particular image, but which are ultimately redundant in the final image, can be 'removed', but remain part of the image if they are removed in a separate step of the build
- The first step of building our `my_redis` image involved updating the packages for Fedora, which almost doubled the size of the image
- The majority of the increase in the image size was down to the DNF cache, which is redundant after the update and subsequent install of Redis
- However, cleaning the cache with dnf `clean all` will remove the unwanted files from the container, but after a further commit, will leave the image at the same size (actually, minimally larger!)

```
REPOSITORY      TAG          IMAGE ID        CREATED           VIRTUAL SIZE
my_redis        latest       e0f2e84fd38d    49 seconds ago    615 MB
<none>          <none>       ebb4d274ca4e    2 hours ago       612.9 MB
<none>          <none>       41de26ebb824    2 hours ago       596.4 MB
```

- The reason is that the files are still present in layer `db26c9654140`, even though they will not be visible in a derived container's filesystem, due to Docker's COW capability

Notes



Import/Export Images

The format of the docker import command is:

```
docker import [options] file|URL|- [repository[:tag]]
```

The config options available for docker import are:

Client Option	Description
`-c, --change=[]`	Alter characteristics of image with Dockerfile instruction
`-m, --message=""`	Adds message to the metadata of the created image

The format of the docker export command is:

```
docker export container
```

The single config option available for docker export is:

Client Option	Description
`-o, --output=""`	Specifies a file to export the container filesystem to

Importing and Exporting Docker Images

- One method for 'flattening' Docker images is to make use of the `docker export` and `docker import` commands
- The `docker export` command exports the specified container's filesystem to `STDOUT` in the form of a tar archive, which can be redirected to a file on the host system
- The archive is of a flat filesystem rather than a layered filesystem, and therefore doesn't retain the history of additions, deletions and amendments that a COW filesystem retains, or any associated image metadata
- In contrast, the `docker import` command creates an empty filesystem image, and then imports a tar archive into it, which may also be compressed (supported compression formats: gzip, bzip2 or xz)
- The `docker import` command can take its input from a file, URL or from `STDIN`
- The format of the `docker import` command is:

```
docker import [options] file|URL|- [repository[:tag]]
```

- The configuration options available for the `docker import` command are:

`-c, --change=[]`: apply characteristics to the image with specified Dockerfile instruction, which can be one of `CMD`, `ENTRYPOINT`, `ENV`, `EXPOSE`, `ONBUILD`, `USER`, `VOLUME`, `WORKDIR`
`-m, --message=””`: specifies a commit message for the image

- The format of the docker export command is:

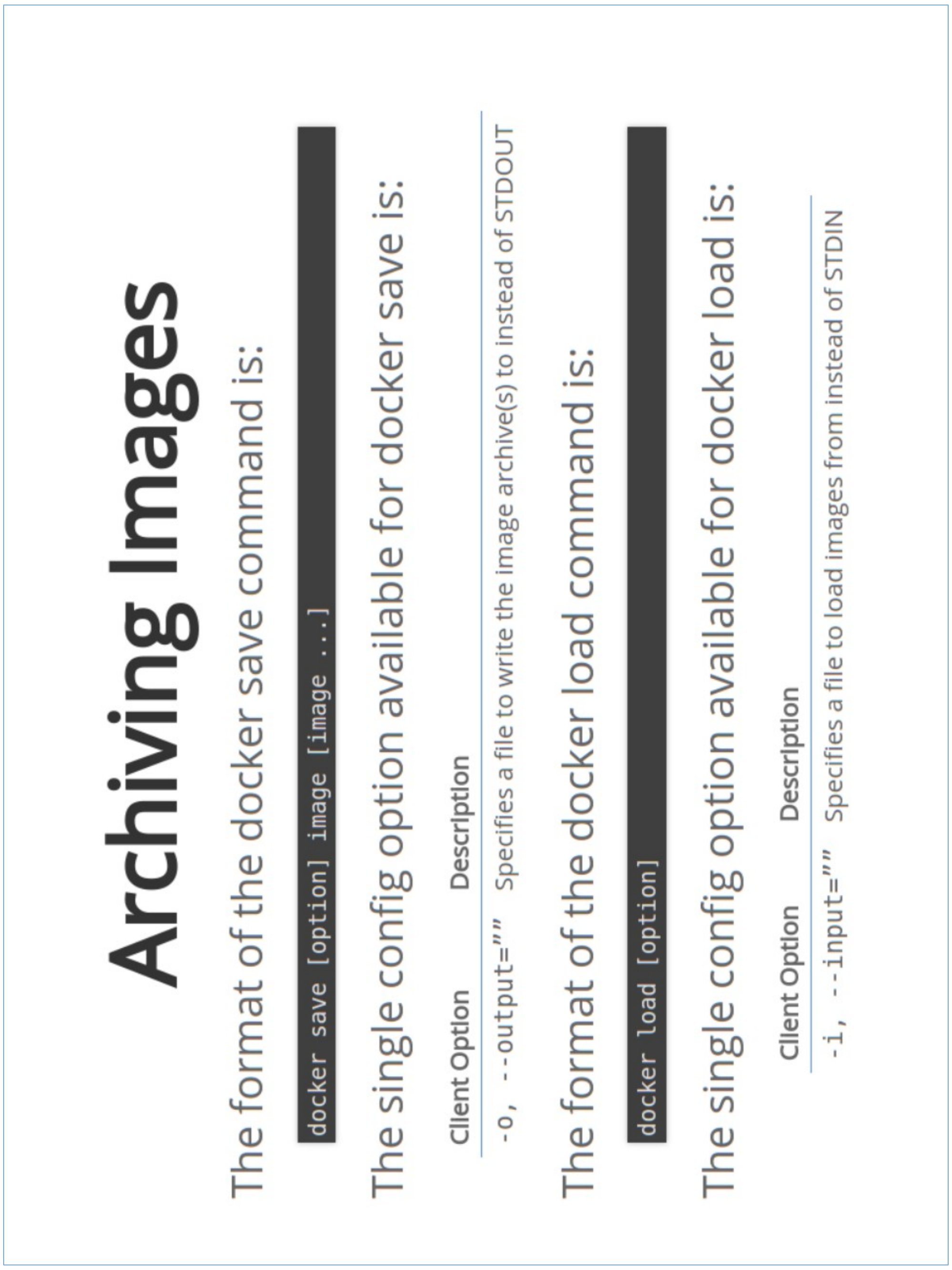
```
docker export container
```

- The configuration options available for the `docker export` command are:

`-o, --output=""`: specifies a file to to write the container's filesystem to

- With the flattening capabilities provided with a `docker export` followed by a `docker import`, the my_redis image can be reduced in size from `615` MB to `351.9` MB

Notes



Archiving Images

The format of the docker save command is:

```
docker save [option] image [image ...]
```

The single config option available for docker save is:

Client Option	Description
`-o, --output=""`	Specifies a file to write the image archive(s) to instead of STDOUT

The format of the docker load command is:

```
docker load [option]
```

The single config option available for docker load is:

Client Option	Description
`-i, --input=""`	Specifies a file to load images from instead of STDIN

Archiving Docker Images

- The commands `docker save` and `docker load` allow for backing up and restoring Docker images in the form of a tar archive
- It is particularly important to back up Docker images using `docker save`, before upgrading Docker to a newer version, or before changing the storage driver; images can be re-loaded using `docker load`
- Unlike `docker export`, which works on a container's filesystem, `docker save` uses the image's graph structure as its source, and all of the specified image's layers and metadata are saved as part of the archive
- It's possible to save an entire repository of images (e.g. `ubuntu`), or a specific tagged image (e.g. `ubuntu:14.04`)
- After a `docker load`, the image(s) contained within the tar archive (which may be compressed) are restored to local repositories on the Docker host
- The format of the `docker save` command is:

```
docker save [option] image [image ...]
```

- The single configuration option available for the `docker save` command is:

`-o, --output=””`: specifies that the tar archive of the image data be written to a specific file rather than `STDOUT`

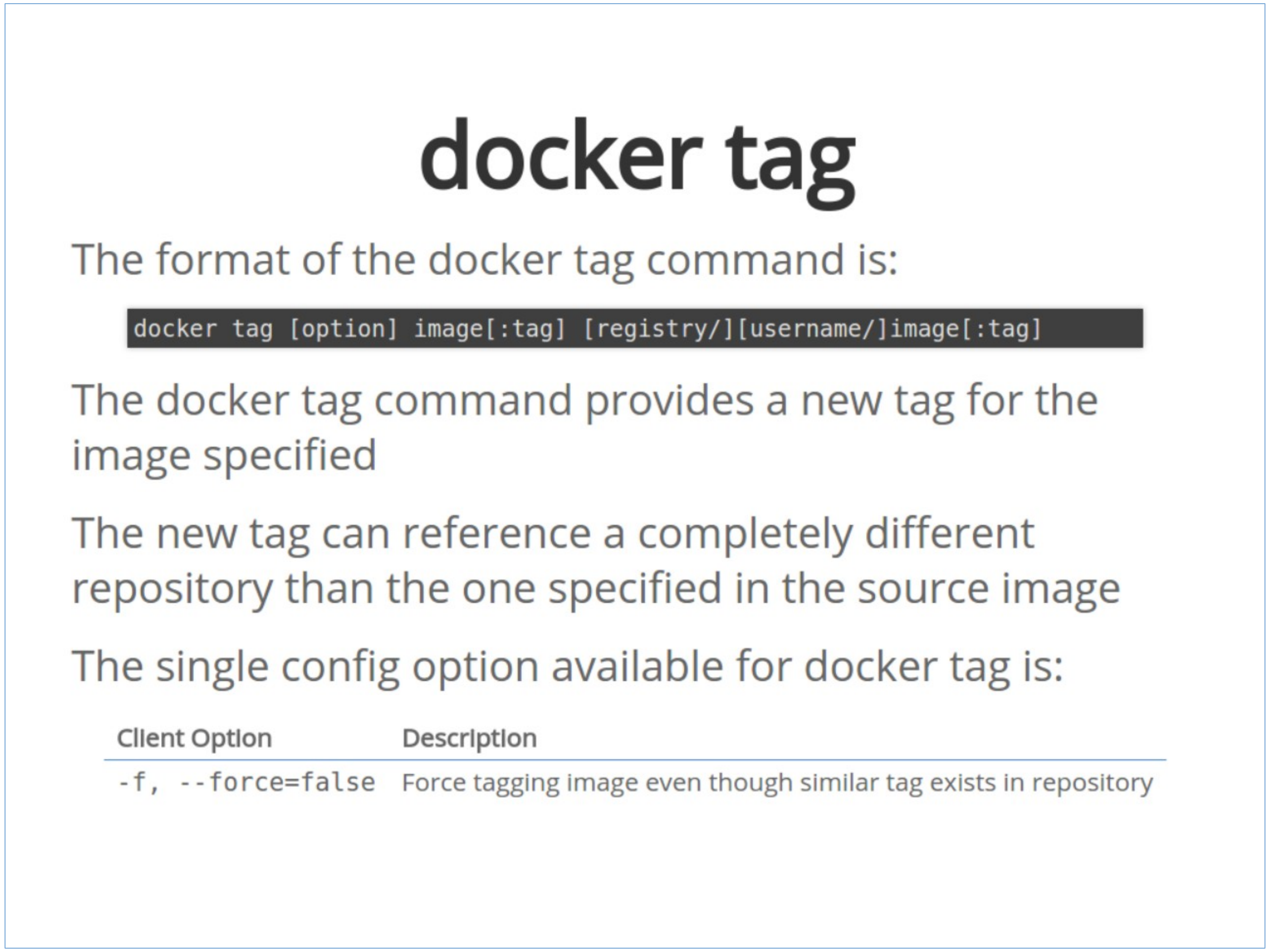
- The format of the `docker load` command is:

```
docker load [option]
```

- The single configuration option available for the `docker load` command is:

`-i, --input=””`: specifies that the tar archive should be read from a specific file rather than `STDIN`

Notes



docker tag

The format of the docker tag command is:

```
docker tag [option] image[:tag] [registry/][username/]image[:tag]
```

The docker tag command provides a new tag for the image specified

The new tag can reference a completely different repository than the one specified in the source image

The single config option available for docker tag is:

Client Option	Description
`-f, --force=false`	Force tagging image even though similar tag exists in repository

docker tag

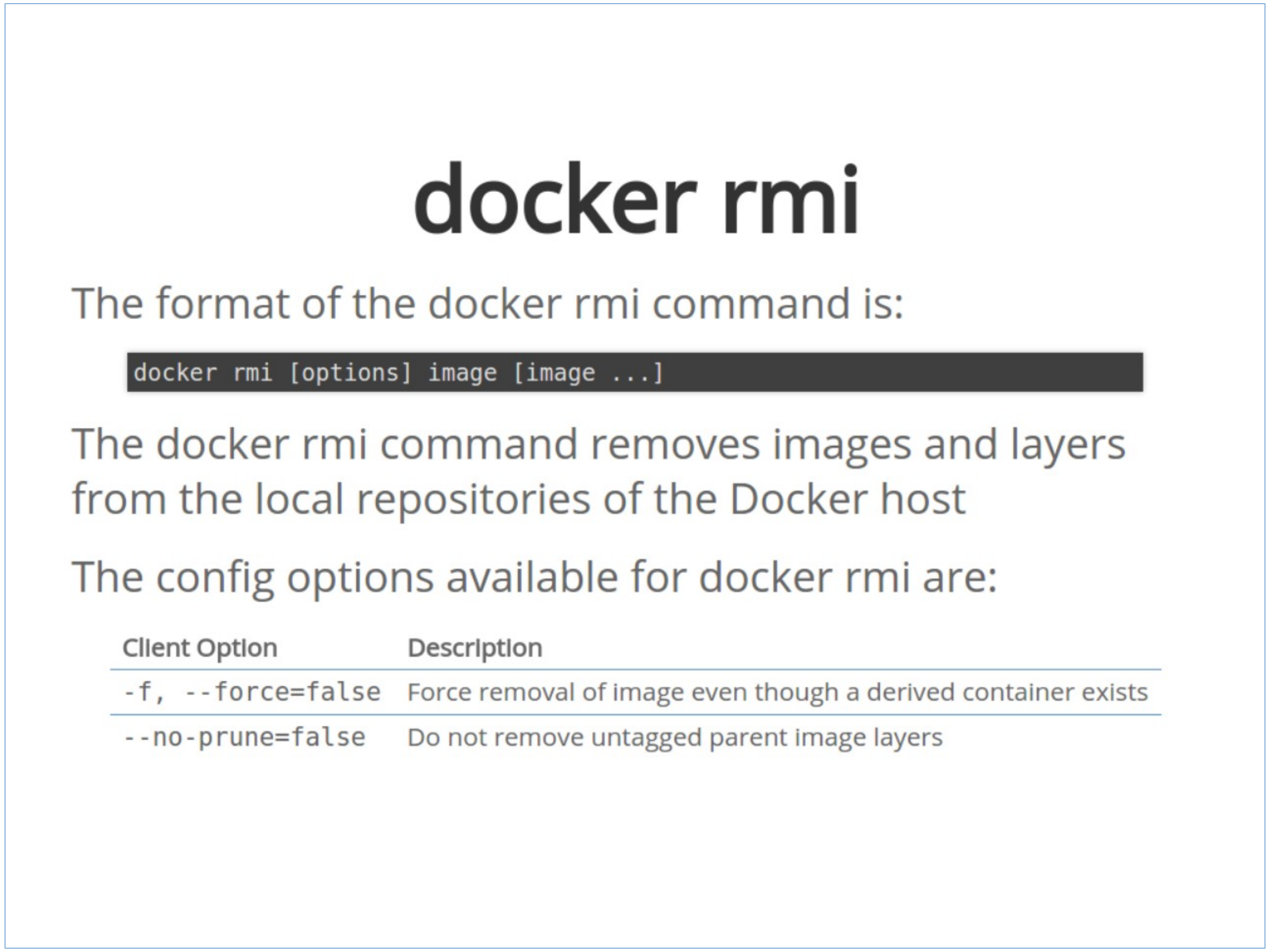
- Sometimes it is relevant to add to or change the tags associated with an image within a repository, perhaps because a new application version has been released, and the `docker tag` command enables you to do this
- A 'target' image is identified either by its ID or by its name, and optionally its tag, and a new image name is specified
- The new image name may refer to a completely different repository to the one in which the referenced image resides
- If the source repository contains a number of related, different images, it's important to either use the ID or name:tag combination in order to correctly identify the image to be tagged
- To replace an image's tag, first add the new tag and then remove the redundant tag using `docker rmi` (see next slide)
- Docker assumes uniqueness in tag usage within repositories, and will therefore not allow two different images to have the same tag, unless the `-f`, `--force` command line option is used to override the default behaviour
- The image may exist in a local repository, a repository in a private registry, or a repository in the Docker Hub registry
- The format of the docker tag command is:

```
docker tag [option] image[:tag] [registry/][username/]image[:tag]
```

- The command line option available for the `docker tag` command is:

`-f, --force=false`: specifies that the provided tag should be applied despite its similarity to a tag associated with a different image in the same repository

Notes



docker rmi

The format of the docker rmi command is:

```
docker rmi [options] image [image ...]
```

The docker rmi command removes images and layers from the local repositories of the Docker host

The config options available for docker rmi are:

Client Option	Description
`-f, --force=false`	Force removal of image even though a derived container exists
`--no-prune=false`	Do not remove untagged parent image layers

docker rmi

- It's not unusual for images to collect in local repositories over a period of time, particularly in development environments, and some housekeeping is required in order to ensure that redundant images are removed when they are no longer required
- Docker provides the `docker rmi` command for removing redundant images (not to be confused with the `docker rm` command, which is used for removing containers)
- An image (layer) will not be removed from the local repository until all references to it have been removed; that is, no other image in the local repositories uses the said image layer
- If a container already exists (running or not) that is derived from the image being removed, the image will not be removed unless the `-f`, `--force` command line option is used
- It only makes sense to remove tagged images, so trying to remove an unnamed, parent image (layer) will fail
- It's possible to preserve untagged parent images when removing tagged images, by specifying the `--no-prune` command line option
- The format of the `docker rmi` command is:

```
docker rmi [options] image [image ...]
```

- The command line options available for the `docker rmi` command are:

`-f, --force=false`: specifies that the image is to be removed from the repository, even though a container currently exists which has been derived from the image
`--no-prune=false`: specifies that untagged, parent images in the image branch should not be removed along with the tagged, leaf image(s)

Notes