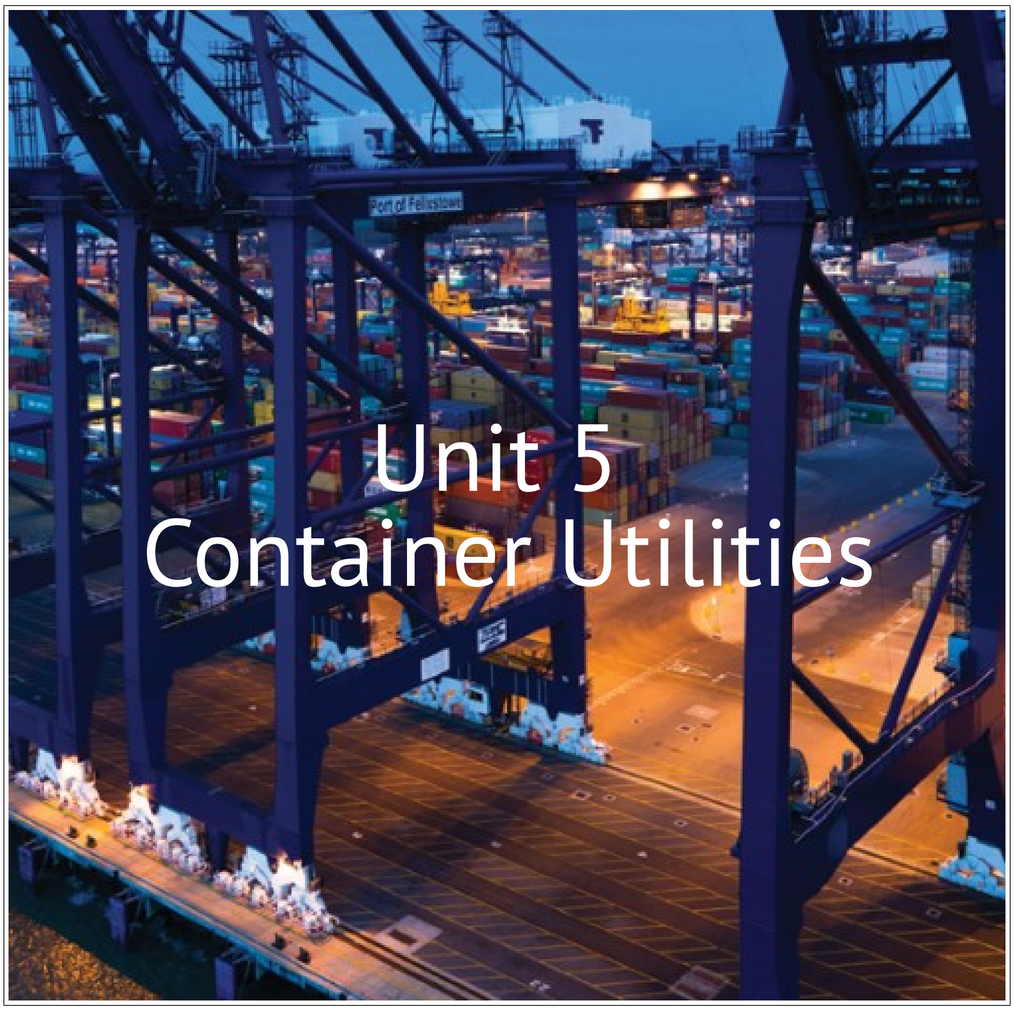


Unit 5
Container Utilities

Objectives

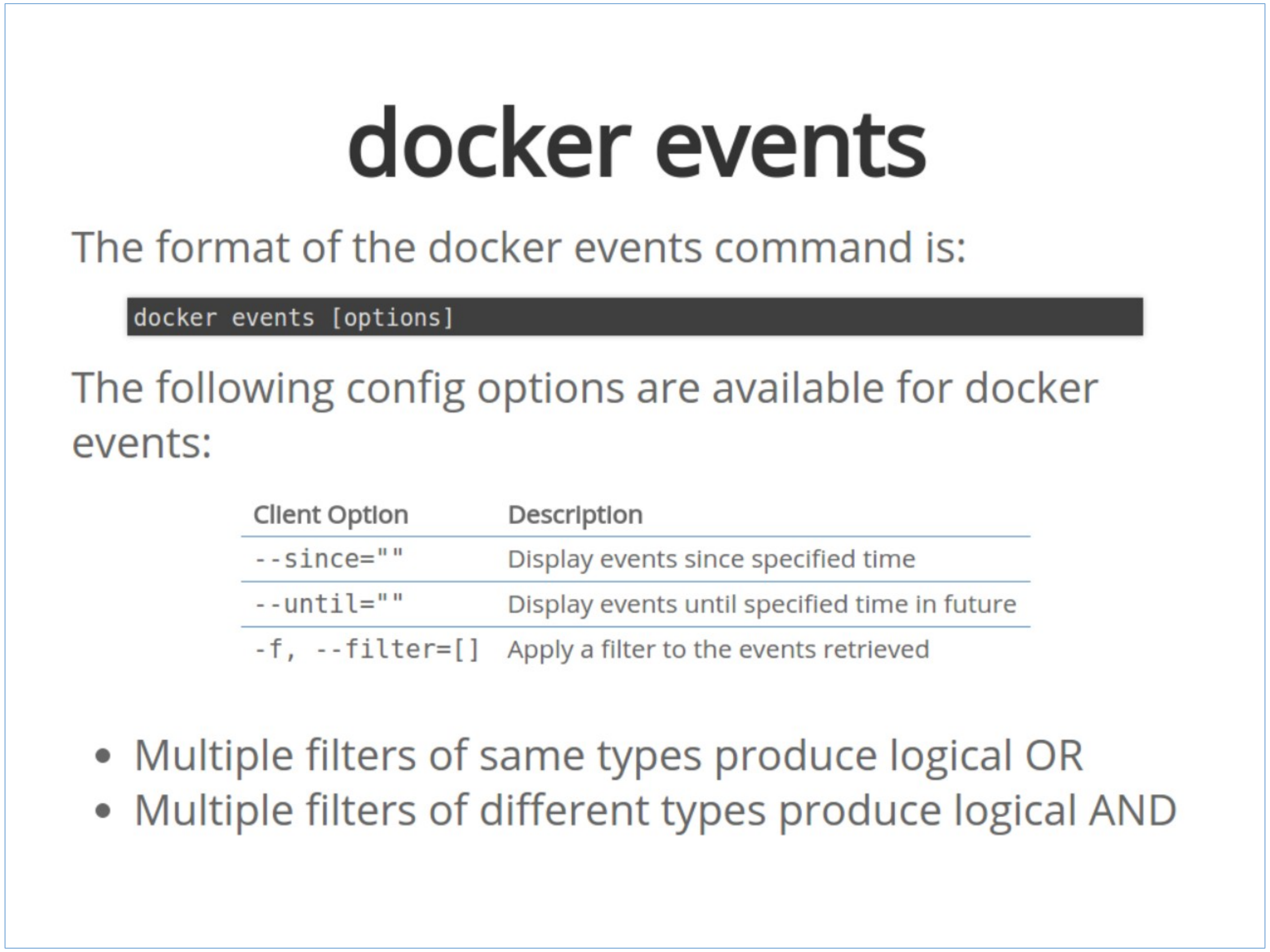
Upon completion of this unit, you will be able to:

- Describe the Docker CLI commands available for retrieving data related to the operation of containers and their corresponding images
- Explain how to view container process' output that has been sent to `stdin` and `stderr` from the container
- Run appropriate commands to view characteristics of a container's runtime environment

Agenda

The following topics will be covered in this unit:

- docker events (container and image events)
- docker inspect (applying the Golang text/template)
- docker logs
- docker stats
- docker wait
- docker ps (filtering and formatting output)
- docker diff
- docker top
- docker port
- docker cp
- Lab exercise: Working With Container Utilities



docker events

The format of the docker events command is:

```
docker events [options]
```

The following config options are available for docker events:

Client Option	Description
`--since=""`	Display events since specified time
`--until=""`	Display events until specified time in future
`-f, --filter=[]`	Apply a filter to the events retrieved

- Multiple filters of same types produce logical OR
- Multiple filters of different types produce logical AND

docker events

- Docker provides a mechanism for retrieving 'events' that the server stores during the lifecycle of containers and images
- The `docker events` command is used to retrieve this information, and can optionally specify a time window from which to retrieve the event data
- Events are logged with the following fields:

  Timestamp: `2015-01-27T11:19:03.000000000Z`
  ID: `2e94b7fe7979b859fe6b5f174d85f4fd7e5ffe8e1925f4788b3ebc144ec9ee8f`
  Image Derivation: `(from centos:latest)`
  Event: `die`

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

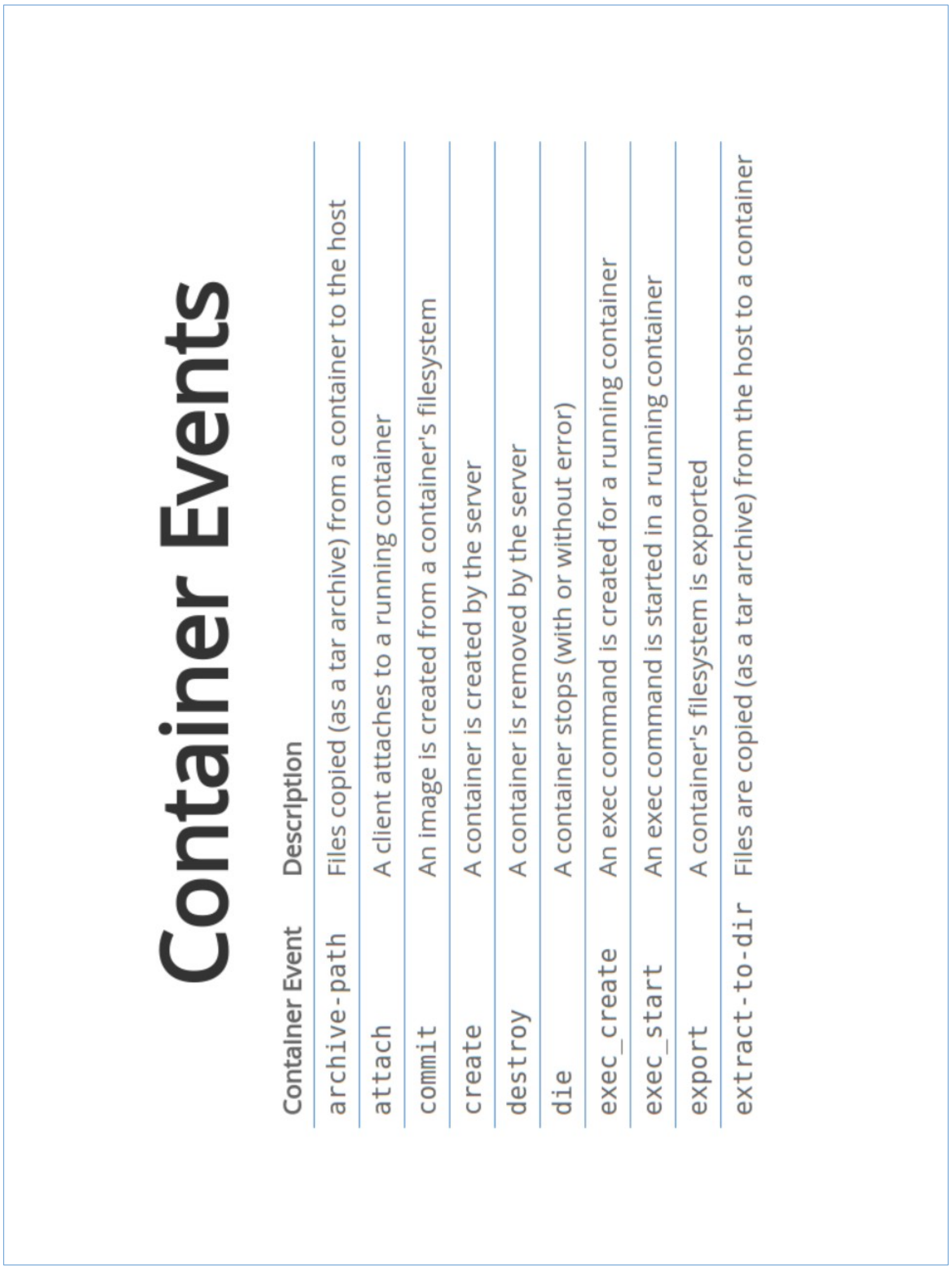
```
docker events [options]
```

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

`--since=””`: specifies a time in either the ISO 8601 format (e.g. `'2015-01-26T15:00:00'`), a UNIX time stamp (e.g. `1422284400`), or Golang duration strings (`'24h'` or `'2h45m'`), from which to display events (if the `--since` option is not used, `docker events` only displays new events)
`--until=””`: specifies a time in the future, up to which to display events
`-f, --filter=[]`: specifies a filter to apply to the events as a key/value pair, and the available filters are: container (`container=5b80485db9fe` or `container=amazing_mayer`), image (`image=962c193e2e02` or `image=debian:8.2`), event (`event=die`), or label (`label=environment` or `label=environment=prod`)

- When specifying multiple filters with similar event keys, the filter will behave like a logical OR function, whilst specifying multiple filter types with dissimilar event keys, the filter will behave like a logical AND function

Notes



Container Events

Container Event	Description
`archive-path`	Files copied (as a tar archive) from a container to the host
`attach`	A client attaches to a running container
`commit`	An image is created from a container's filesystem
`create`	A container is created by the server
`destroy`	A container is removed by the server
`die`	A container stops (with or without error)
`exec_create`	An exec command is created for a running container
`exec_start`	An exec command is started in a running container
`export`	A container's filesystem is exported
`extract-to-dir`	Files are copied (as a tar archive) from the host to a container

Container Events

The following container events are reported by the `docker events` command:

Container Event	Description
archive-path	files are copied (as a tar archive) from a container to the host
attach	a client attaches to a running container
commit	an image is created from a container's filesystem
create	a container is created by the server
destroy	a container is removed by the server
die	a container stops (with or without error)
exec_create	an exec command is created for a running container
exec_start	an exec command is started in a running container
export	a container's filesystem is exported
extract-to-dir	files are copied (as a tar archive) from the host to a container
kill	the server kills a running container
oom	a container exits due to being out of memory
pause	the server pauses a running container
rename	a container is renamed
resize	TTY for container process is resized
restart	a container is restarted
start	a container is started
stop	a container is sent the stop signal
top	processes running in the container are listed
unpause	the server unpauses a running container

Notes

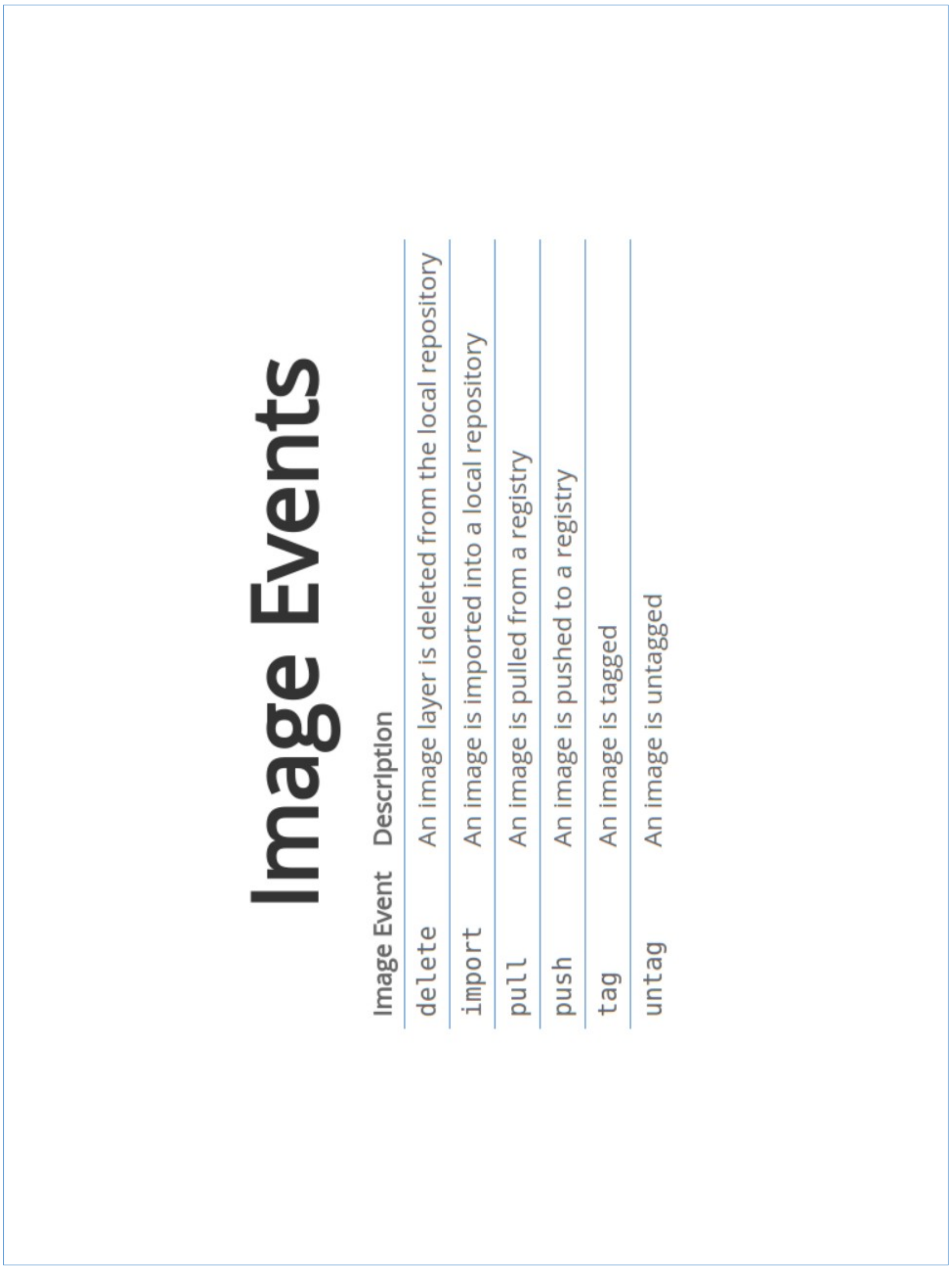


Image Events

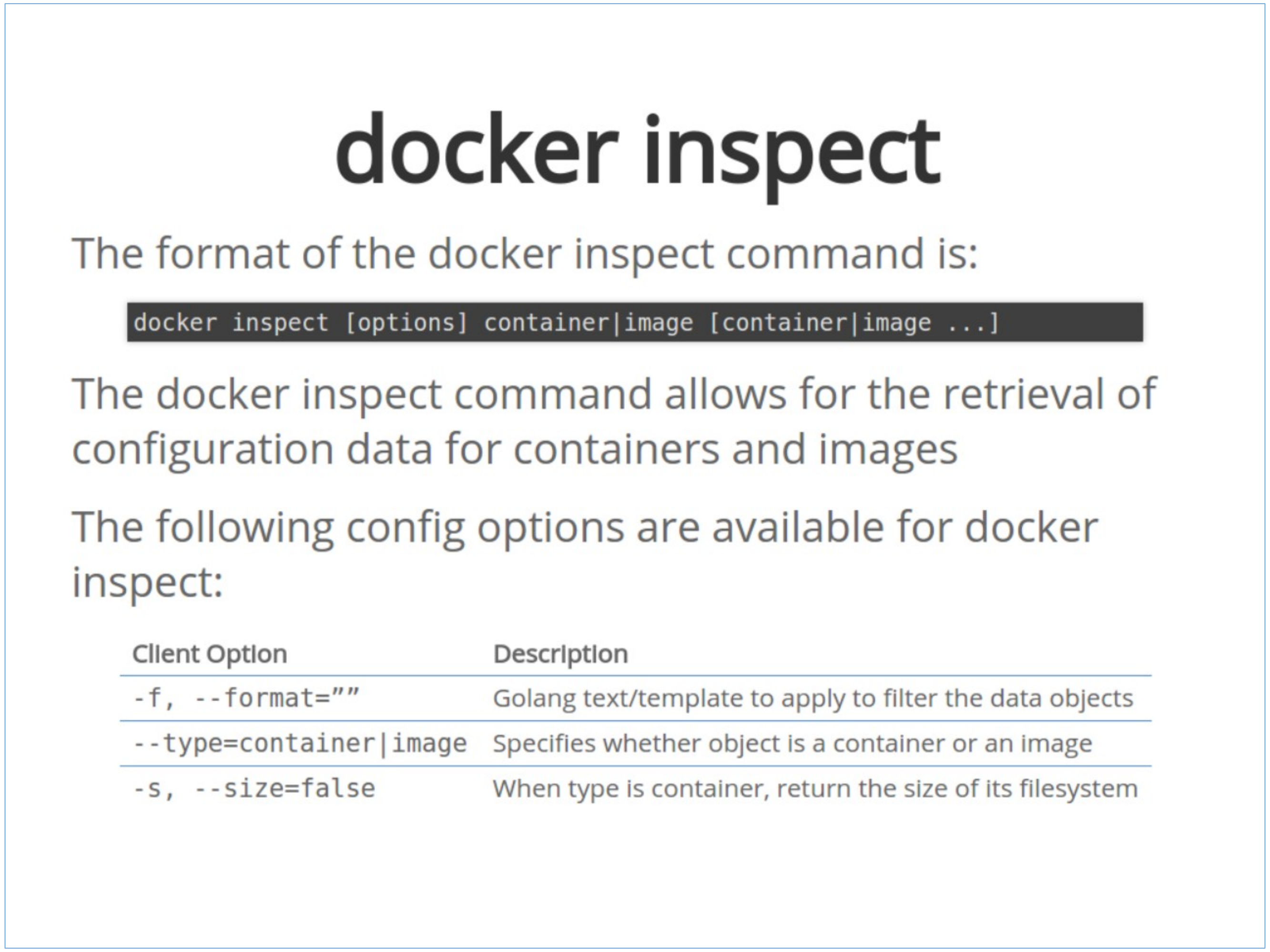
Image Event	Description
`delete`	An image layer is deleted from the local repository
`import`	An image is imported into a local repository
`pull`	An image is pulled from a registry
`push`	An image is pushed to a registry
`tag`	An image is tagged
`untag`	An image is untagged

Image Events

The following image events are reported by the `docker events` command:

Image Event	Description
delete	an image layer is deleted from the local repository
import	an image is imported into a local repository
pull	an image is pulled from a registry
push	an image is pushed to a registry
tag	an image is tagged
untag	an image is untagged

Notes



docker inspect

The format of the docker inspect command is:

```
docker inspect [options] container|image [container|image ...]
```

The docker inspect command allows for the retrieval of configuration data for containers and images

The following config options are available for docker inspect:

Client Option	Description
`-f, --format=""`	Golang text/template to apply to filter the data objects
`--type=container|image`	Specifies whether object is a container or an image
`-s, --size=false`	When type is container, return the size of its filesystem

docker inspect

- The `docker inspect` command allows you to retrieve configuration information for a specific container or image
- Docker uses the methods provided by the Go programming language text/template package for manipulating container or image configuration data, for output
- A formatting option (a template) can be supplied to the command, which is applied to the configuration data objects, in order to reference the object's fields to filter the output
- If no formatting argument is supplied to the `docker inspect` command, Docker outputs the entire container or image configuration as a JSON array
- The format of the `docker inspect` command is:

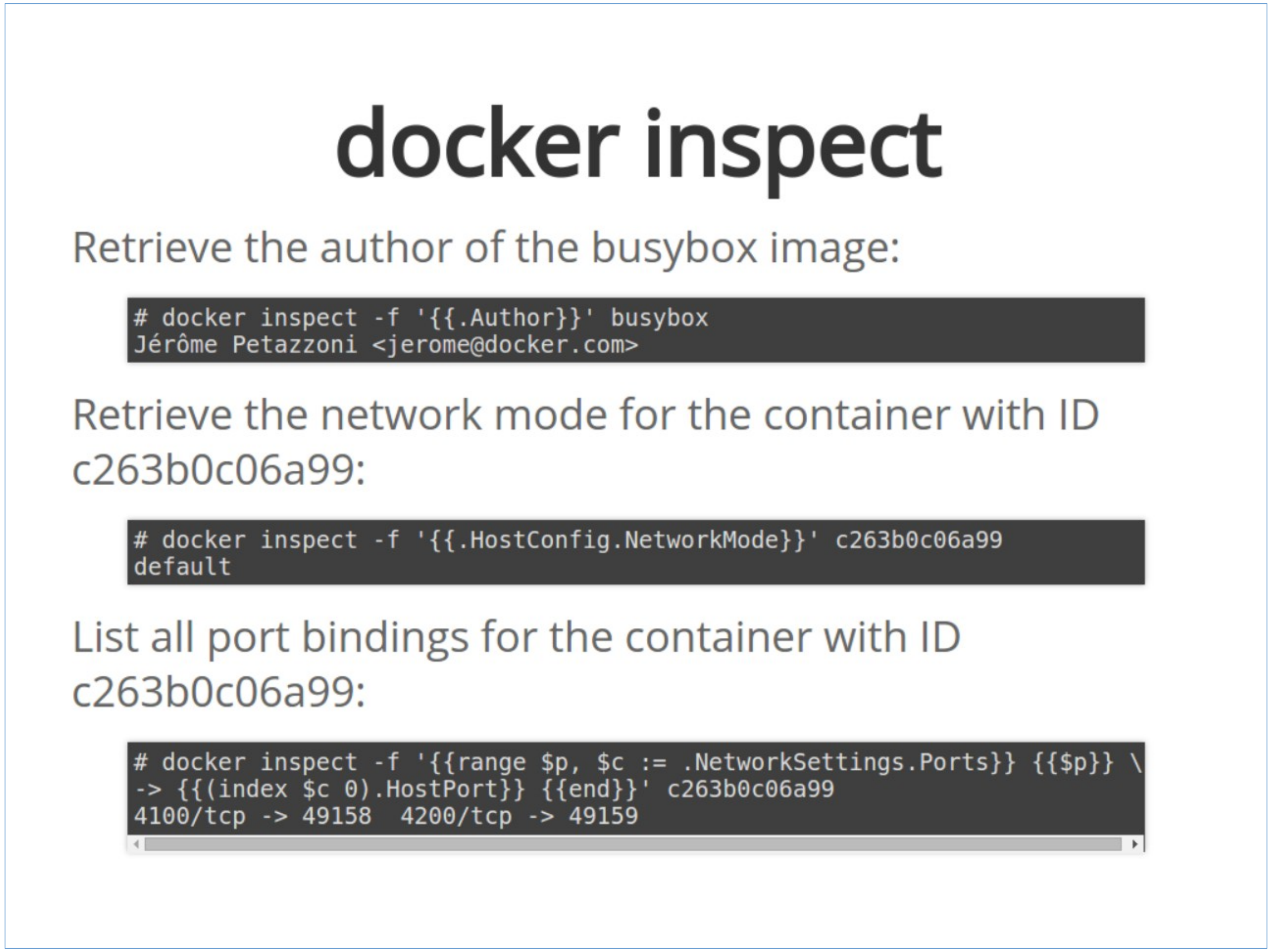
```
docker inspect [options] container|image [container|image ...]
```

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

`-f, --format=""`: specifies a Go text/template to apply to the data objects in order to filter the results
`--type=container|image`: when a container and an image have a similar name (e.g. `rhel7`), specifies whether the object to inspect is a container or an image
`-s, --size=false`: when the type is 'container', add the size of the filesystem to the output

- The Go text/template is defined with curly braces, `{{ … }}`, and the fields are referenced with the notation, `.Field`, and sub-fields are referenced with, `.Field.Sub-Field`
- Some configuration items may have multiple items (e.g. volumes, ports), and will be stored in an array or map
- The format syntax provides for looping through a field which is an array or map, in order to retrieve the desired values

Notes



docker inspect

Retrieve the author of the busybox image:

```
# docker inspect -f '{{.Author}}' busybox
Jérôme Petazzoni <jerome@docker.com>
```

Retrieve the network mode for the container with ID c263b0c06a99:

```
# docker inspect -f '{{.HostConfig.NetworkMode}}' c263b0c06a99
default
```

List all port bindings for the container with ID c263b0c06a99:

```
# docker inspect -f '{{range $p, $c := .NetworkSettings.Ports}} {{$p}} \
-> {{(index $c 0).HostPort}} {{end}}' c263b0c06a99
4100/tcp -> 49158  4200/tcp -> 49159
```

docker inspect

The following examples serve to illustrate the use of the `-f`, `--format` command line option for the `docker inspect` command:

Retrieve the author of the busybox image

```
# docker inspect -f '{{.Author}}' busybox
Jérôme Petazzoni <jerome@docker.com>
```

Retrieve the network mode of the container with ID c263b0c06a99

```
# docker inspect -f '{{.HostConfig.NetworkMode}}' c263b0c06a99
default
```

Retrieve a list of the volumes mounted RW used by the container with ID c263b0c06a99

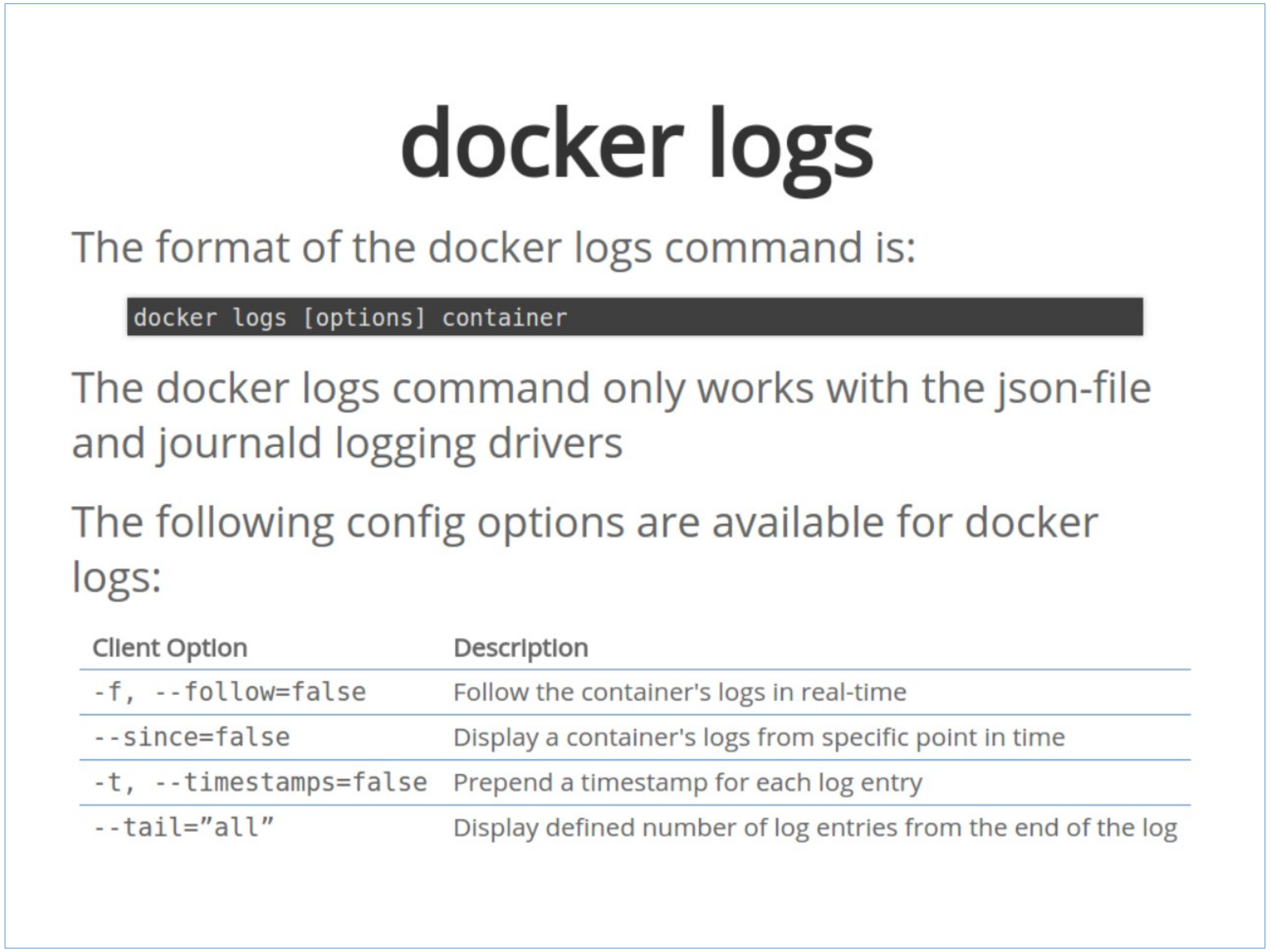
```
# docker inspect -f '{{.VolumesRW}}' c263b0c06a99
map[/app/conf:true /app/data:true]
```

List all port bindings for the container with ID c263b0c06a99

```
# docker inspect -f '{{range $p, $c := .NetworkSettings.Ports}} {{$p}} \
-> {{(index $c 0).HostPort}} {{end}}' c263b0c06a99
4100/tcp -> 49158  4200/tcp -> 49159
```

A good tutorial on the use of the text/template can be found in the blog post, Docker Inspect Template Magic, by Adrian Mouat

Notes



docker logs

The format of the docker logs command is:

```
docker logs [options] container
```

The docker logs command only works with the json-file and journald logging drivers

The following config options are available for docker logs:

Client Option	Description
`-f, --follow=false`	Follow the container's logs in real-time
`--since=false`	Display a container's logs from specific point in time
`-t, --timestamps=false`	Prepend a timestamp for each log entry
`--tail="all"`	Display defined number of log entries from the end of the log

docker logs

- The `docker logs` command displays a log of the `stdout` and `stderr` output from the container's main process
- The `docker logs` command only works if the logging driver for the container is either `json-file` or `journald`
- The logs are located in the `/var/lib/docker/containers` directory, in the directory named with the container's ID*
- Each line of `stdout` or `stderr` is written to either the container's logfile (which is called `<container>-json.log`, where <container> is the container's ID), or the systemd journal service, recording; the data, the stream and a timestamp in ISO 8601 format
- When no options are supplied to specify otherwise, the `docker logs` command reads the entire log history and dumps this to the terminal, with the logs in chronological order
- The format for the `docker logs` command is:

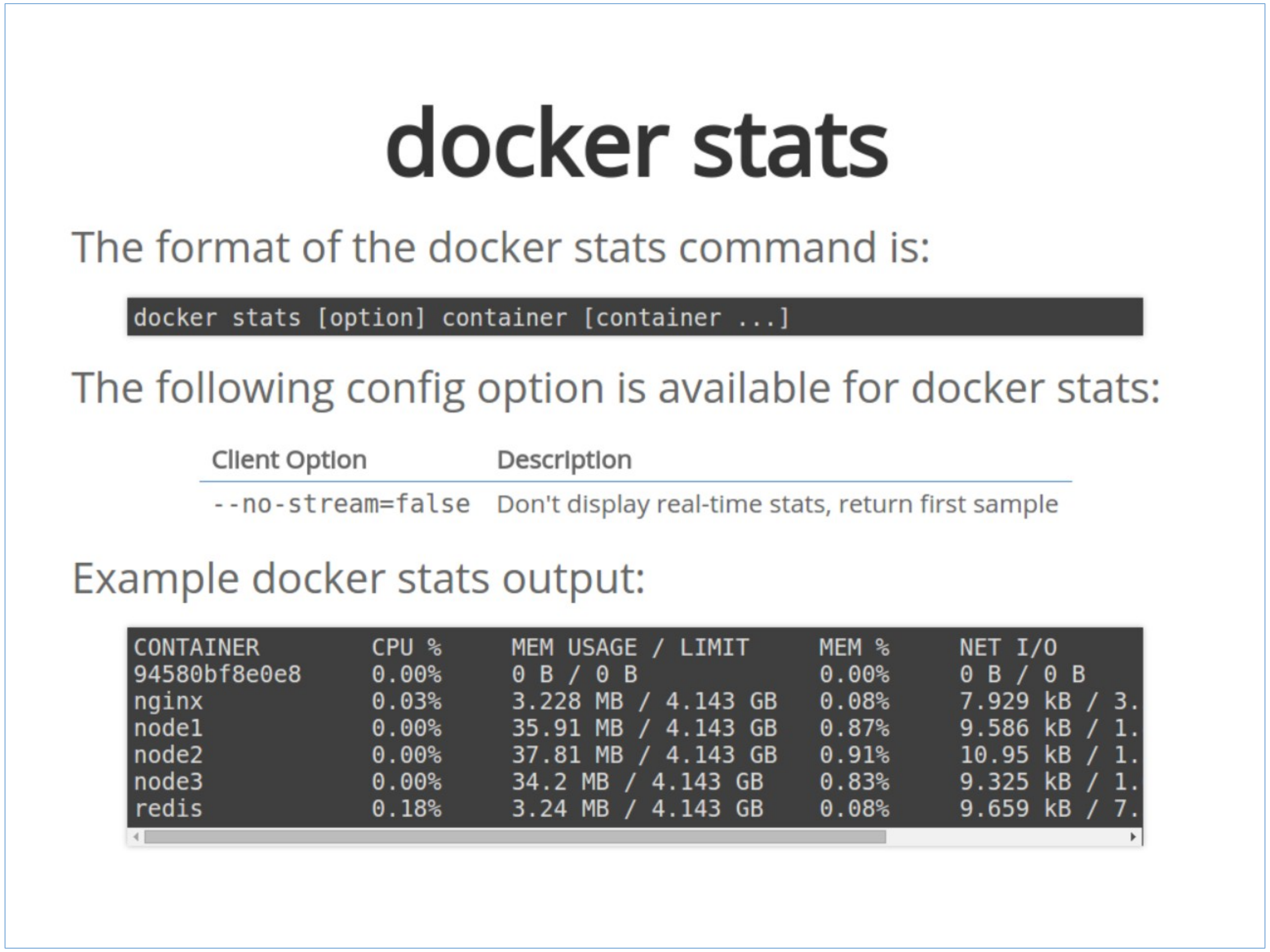
```
docker logs [options] container
```

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

`-f, --follow=false`: specifies that the logs are followed, just like the Linux command `tail -f`, where all the logs are read and dumped to the terminal, and thereafter the client is attached to the container in order to receive any further real -time output
`--since=false`: specifies that only logs after a given moment in time are displayed, specified in either the ISO 8601 format (e.g. '2015-01-26T15:00:00'), a UNIX time stamp (e.g. 1422284400), or Golang duration strings ('24h' or '2h45m')
`-t, --timestamps=false`: specifies that the output provided by the command be prepended with the timestamp associated with the log entry
`--tail=”all”`: specifies that a number of lines at the end of the logs be displayed, and if a number is not specified, it defaults to all of the logs

* Assumes the root of Docker's runtime is `/var/lib/docker`

Notes



docker stats

The format of the docker stats command is:

```
docker stats [option] container [container ...]
```

The following config option is available for docker stats:

Client Option	Description
`--no-stream=false`	Don't display real-time stats, return first sample

Example docker stats output:

```
CONTAINER           CPU %     MEM USAGE / LIMIT     MEM %     NET I/O
94580bf8e0e8        0.00%     0 B / 0 B             0.00%     0 B / 0 B
nginx               0.03%     3.228 MB / 4.143 GB   0.08%     7.929 kB / 3.
node1               0.00%     35.91 MB / 4.143 GB   0.87%     9.586 kB / 1.
node2               0.00%     37.81 MB / 4.143 GB   0.91%     10.95 kB / 1.
node3               0.00%     34.2 MB / 4.143 GB    0.83%     9.325 kB / 1.
redis               0.18%     3.24 MB / 4.143 GB    0.08%     9.659 kB / 7.
```

docker stats

- The `docker` stats command provides a view of the system resources used by a container, or list of containers
- More specifically, the command provides information on the amount of CPU used by the container (as a percentage of total), the memory used in actual and relative terms, and the network and block IO usage
- The format of the `docker stats` command is:

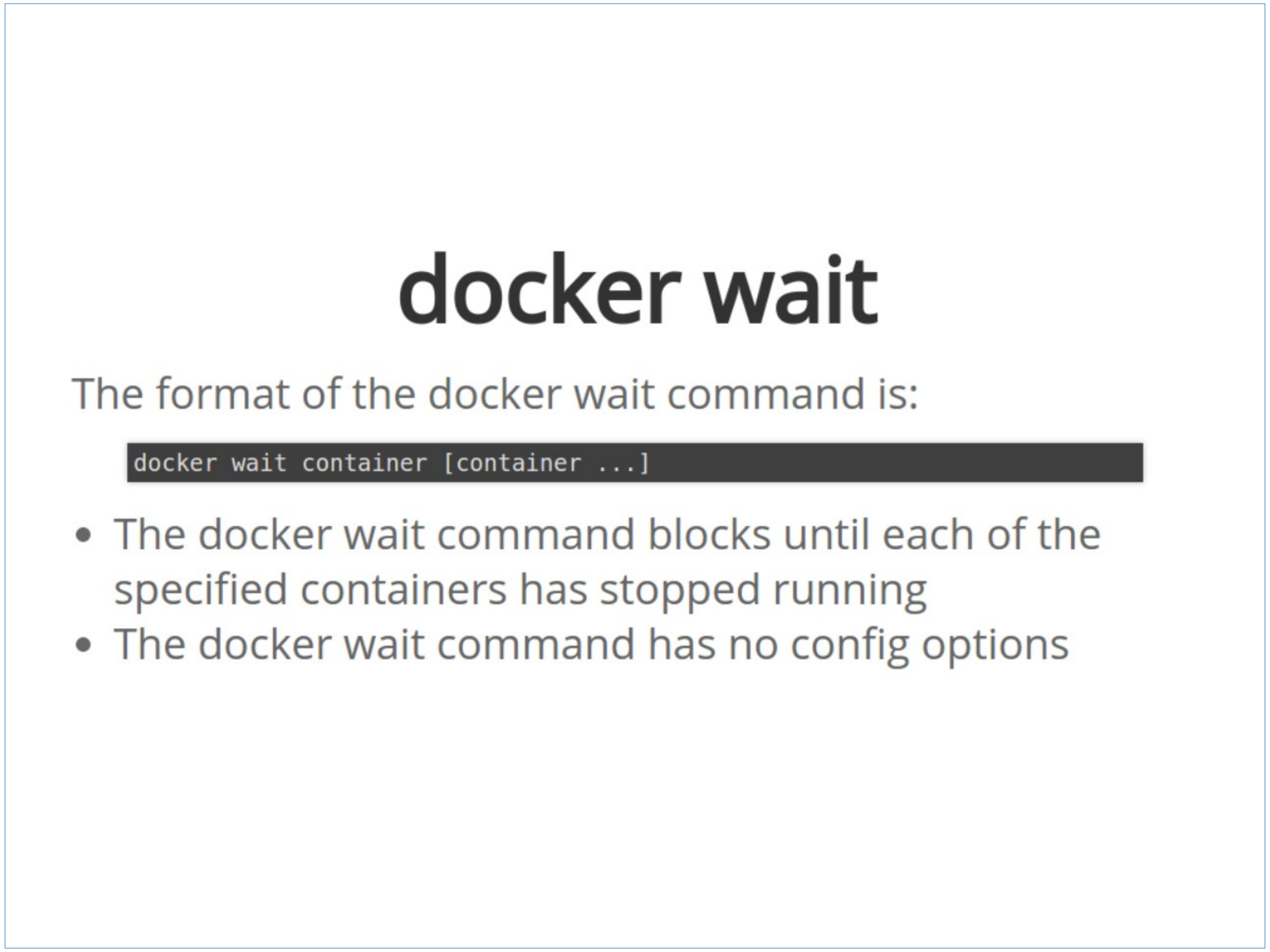
```
docker stats [option] container [container ...]
```

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

`--no-stream=false`: specifies that live stats are suppressed, and only returns the first sample

- Stats are only provided for running containers; stopped containers are listed without any information

Notes



docker wait

The format of the docker wait command is:

```
docker wait container [container ...]
```

- The docker wait command blocks until each of the specified containers has stopped running
- The docker wait command has no config options

docker wait

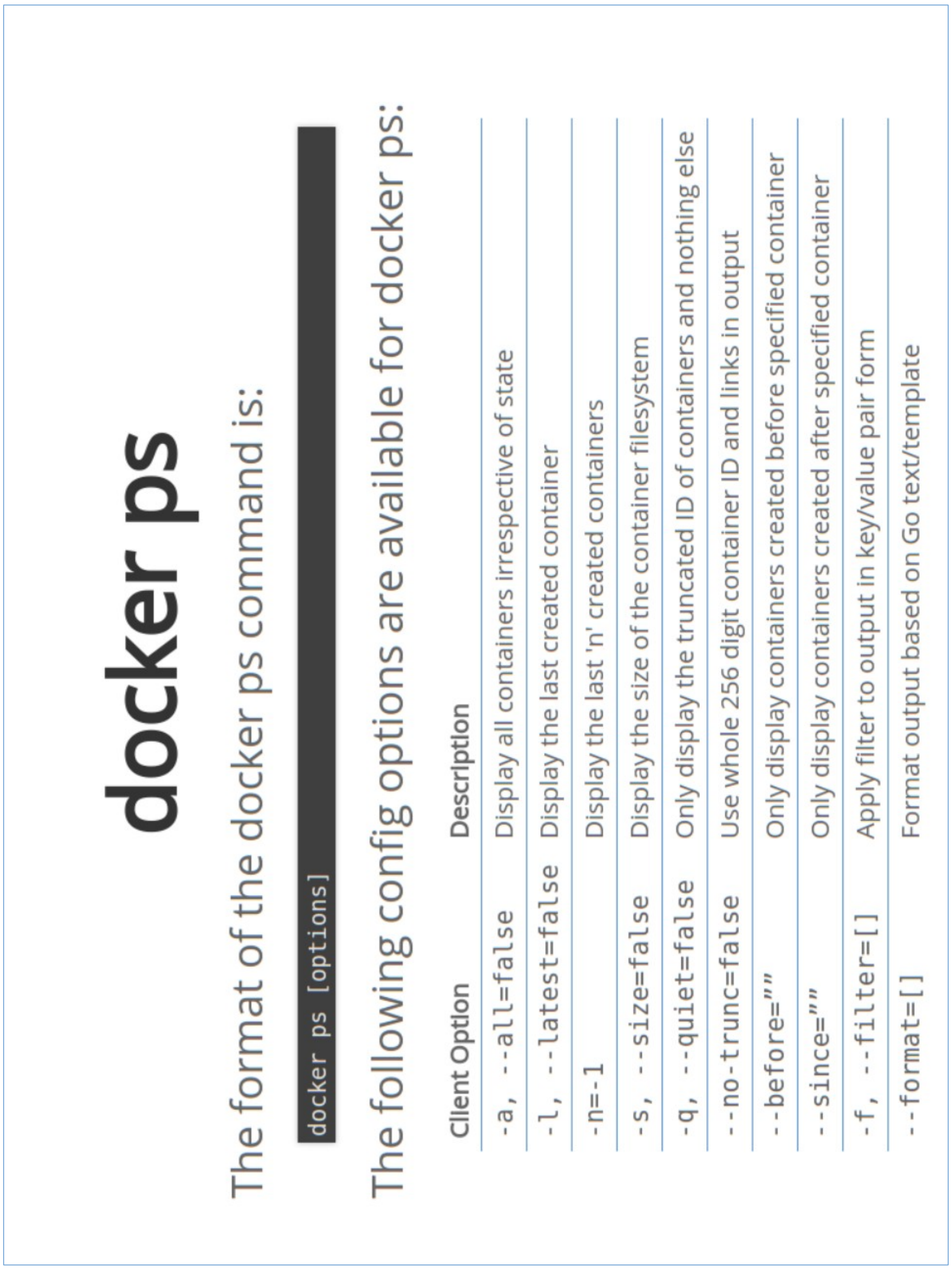
- The `docker wait` command blocks until the specified container(s) exits, where upon it prints the exit code
- The format of the `docker wait` command is:

```
docker wait container [container ...]
```

- The `docker wait` command has no options
- A `docker wait` example:

```
$ docker wait nginx redis node1 node2 node3 94580bf8e0e8
0
0
0
0
0
130
```

Notes



docker ps

The format of the docker ps command is:

```
docker ps [options]
```

The following config options are available for docker ps:

Client Option	Description
`-a, --all=false`	Display all containers irrespective of state
`-l, --latest=false`	Display the last created container
`-n=-1`	Display the last 'n' created containers
`-s, --size=false`	Display the size of the container filesystem
`-q, --quiet=false`	Only display the truncated ID of containers and nothing else
`--no-trunc=false`	Use whole 256 digit container ID and links in output
`--before=""`	Only display containers created before specified container
`--since=""`	Only display containers created after specified container
`-f, --filter=[]`	Apply filter to output in key/value pair form
`--format=[]`	Format output based on Go text/template

docker ps

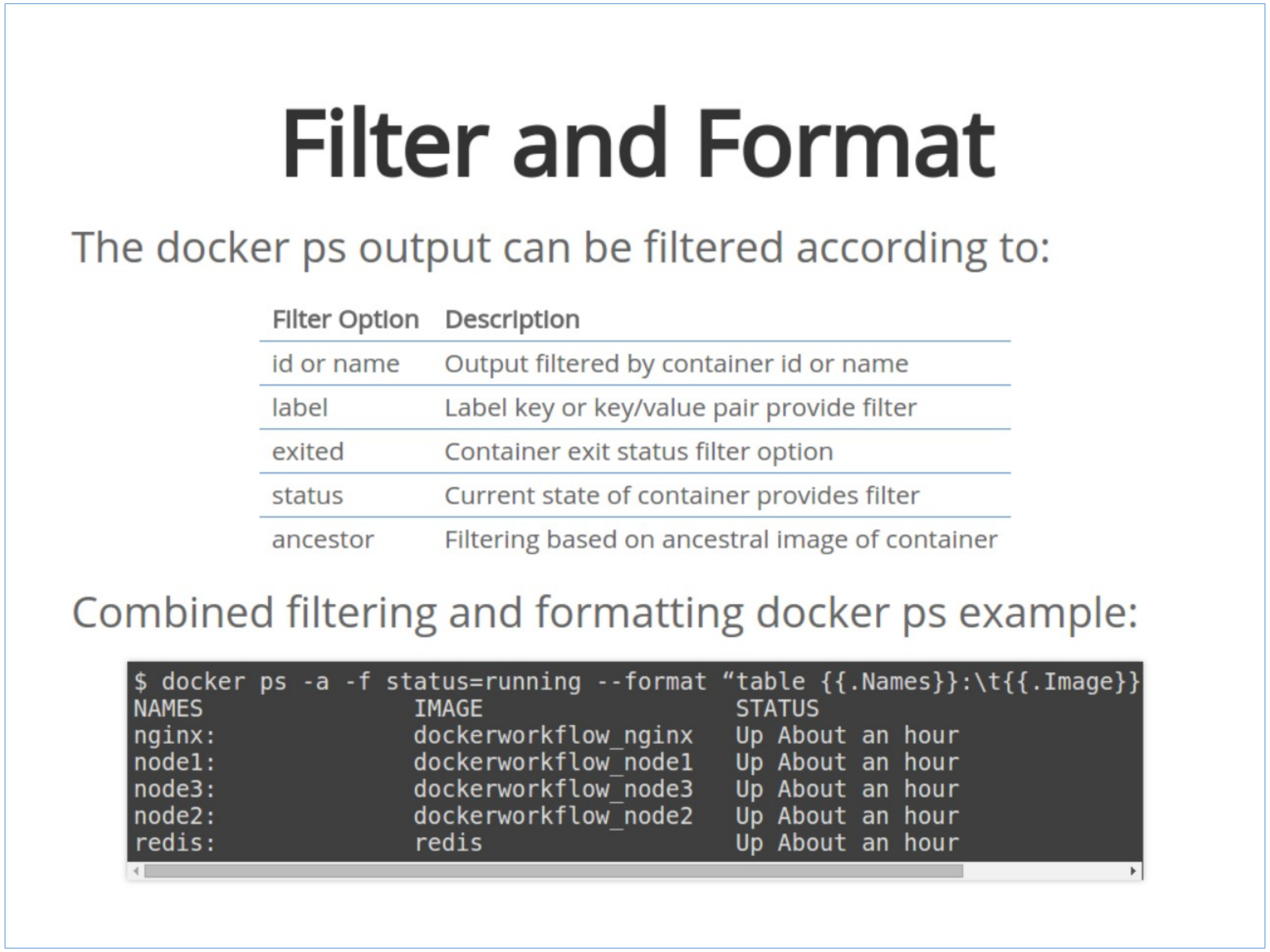
- The `docker ps` command is analogous to the 'nix `ps` command, in that it provides a listing of existing containers, and various attributes associated with those containers
- The `docker ps` command is probably the most often used of the utility commands, and can be used as input to other Docker commands
- The format of the `docker ps` command is:

```
        docker ps [options]
```

- Without any command line options, `docker ps` lists all of the currently running containers
- The configuration options available for the `docker ps` command are:

`-a, --all=false`: list all existing containers, whether in the running state or stopped
`-l, --latest=false`: lists last created container, irrespective of its state; running or stopped
`-n=-1`: lists the last 'n' created containers, irrespective of their state; running or otherwise
`-s, --size=false`: augments the output with the total size of the container, and the size of the final layer of the container's union filesystem (which will initially be 0 bytes)
`-q, --quiet=false`: specifies that only the ID of the containers are listed, which is frequently used in conjunction with other Docker commands (e.g. `docker rm`)
`--no-trunc=false`: specifies that `docker ps` outputs the entire ID of a container, instead of a truncated ID, as well as the name and alias of any links associated with a container listed
`--before=””`: only list containers that were created before the container provided, irrespective of its state; running or stopped (e.g. `-before 2ccea01f6208`)
`--since=””`: only list containers that were created since the container provided, irrespective of its state; running or stopped (e.g. `-since 2ccea01f6208`)
`-f, --filter=[]`: specifies a filter to apply to the list as a key/value pair, and the available filters are: id, label, name, exit code, status and image ancestor
`--format=[]`: specifies use of a Go text/template to pretty print the output

Notes



Filter and Format

The docker ps output can be filtered according to:

Filter Option	Description
id or name	Output filtered by container id or name
label	Label key or key/value pair provide filter
exited	Container exit status filter option
status	Current state of container provides filter
ancestor	Filtering based on ancestral image of container

Combined filtering and formatting docker ps example:

```
$ docker ps -a -f status=running --format "table {{.Names}}:\t{{.Image}}
NAMES                IMAGE                  STATUS
nginx:               dockerworkflow_nginx   Up About an hour
node1:               dockerworkflow_node1   Up About an hour
node3:               dockerworkflow_node3   Up About an hour
node2:               dockerworkflow_node2   Up About an hour
redis:               redis                  Up About an hour
```

Filter and Format

Filter

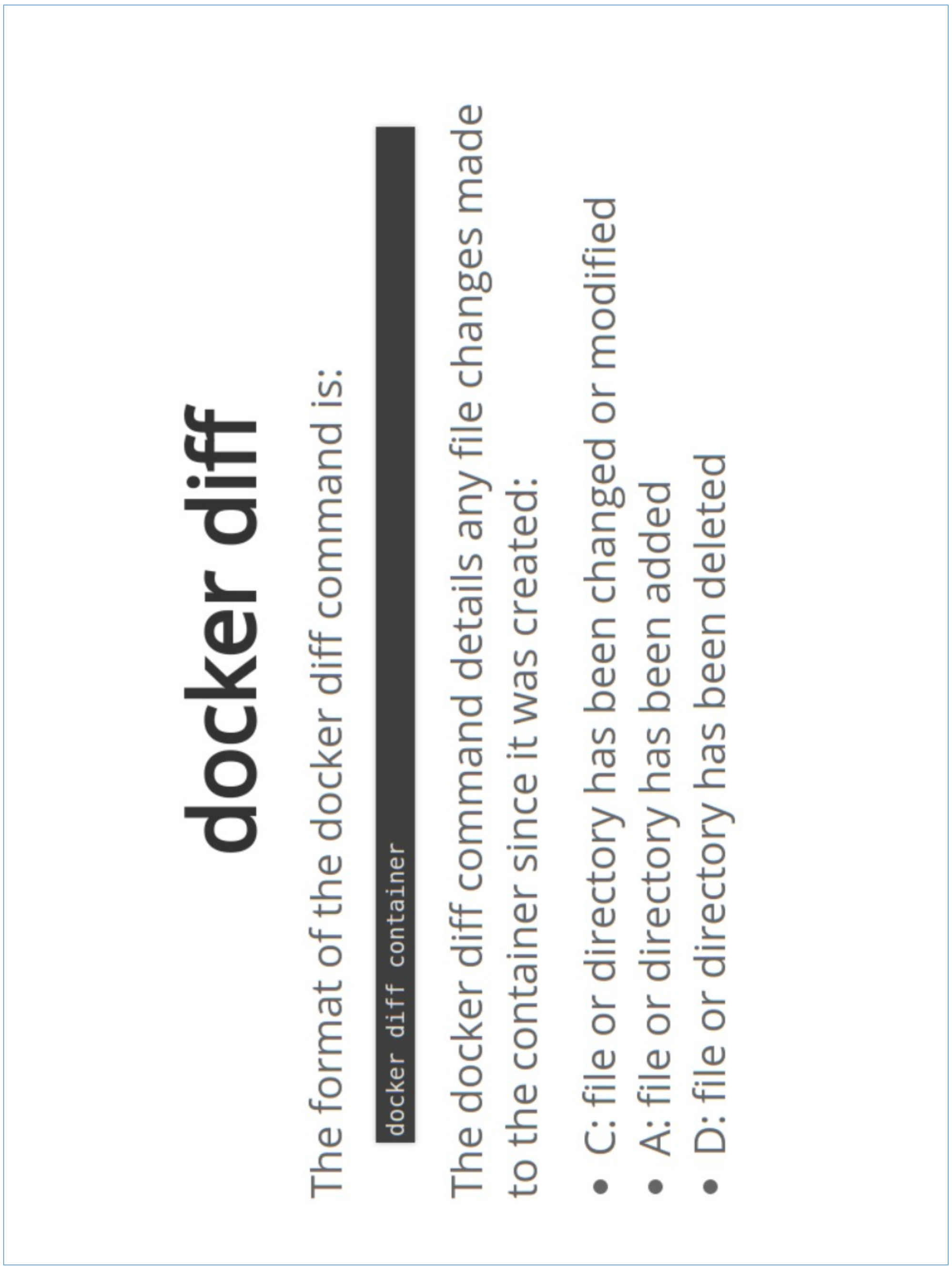
- The `-f`, `--filter` configuration option can be applied to filter the `docker ps` output for a specific container by id or name, e.g. `docker ps -f 9738e7c7f277` or `docker ps -f stoic_albattani`
- When a filter is specfied using the `exited` key, the `docker ps` command also requires the `-a` configuration option, e.g. `docker ps -a -f “exited=0”`
- Filtering `docker ps` output based on labels can take one of two forms, one specifying just the key, the other specifying the key/value pair, e.g. `docker ps -f “label=env”` or `docker ps -f “label=env=qa”`
- State filtering can be achieved using the `status` key, which can take one of the following values: `created`, `restarting`, `running`, `paused`, `exited`
- The `ancestor` key provides a means of filtering based on whether containers have an ancestral image defined by the associated value, which can take one of several forms: `image`, `image:tag`, `image:tag@digest`, `short-id`, `long-id`; e.g. `docker ps -f ancestor=node:5.2.0-onbuild`

Format

- The `--format` configuration option can take a number of 'placeholders' which are used to format the output: `.ID`, `.Image`, `.Command`, `.CreatedAt`, `.RunningFor`, `.Ports`, `.Status`, `.Size`, `.Names`, `.Labels`, `.Label` (specific label)
- The table keyword is required to provide placeholder headings:

```
$ docker ps -a --format “table {{.Names}}:\t{{.Image}}\t{{.Status}}”
NAMES               IMAGE                  STATUS
nginx:              dockerworkflow_nginx   Up About an hour
node1:              dockerworkflow_node1   Up About an hour
node3:              dockerworkflow_node3   Up About an hour
node2:              dockerworkflow_node2   Up About an hour
redis:              redis                  Up About an hour
sleepy_liskov:      debian                 Exited (130) 3 hours ago
```

Notes



docker diff

The format of the docker diff command is:

```
docker diff container
```

The docker diff command details any file changes made to the container since it was created:

- C: file or directory has been changed or modified
- A: file or directory has been added
- D: file or directory has been deleted

docker diff

- The `docker diff` command provides a detailed view of the changes made to a container's filesystem since its pristine creation state
- The format for the `docker diff` command is:

```
docker diff container
```

- Each file and directory that has altered since the container's creation is listed, along with a letter indicating the nature of the alteration
- The three altered states are:

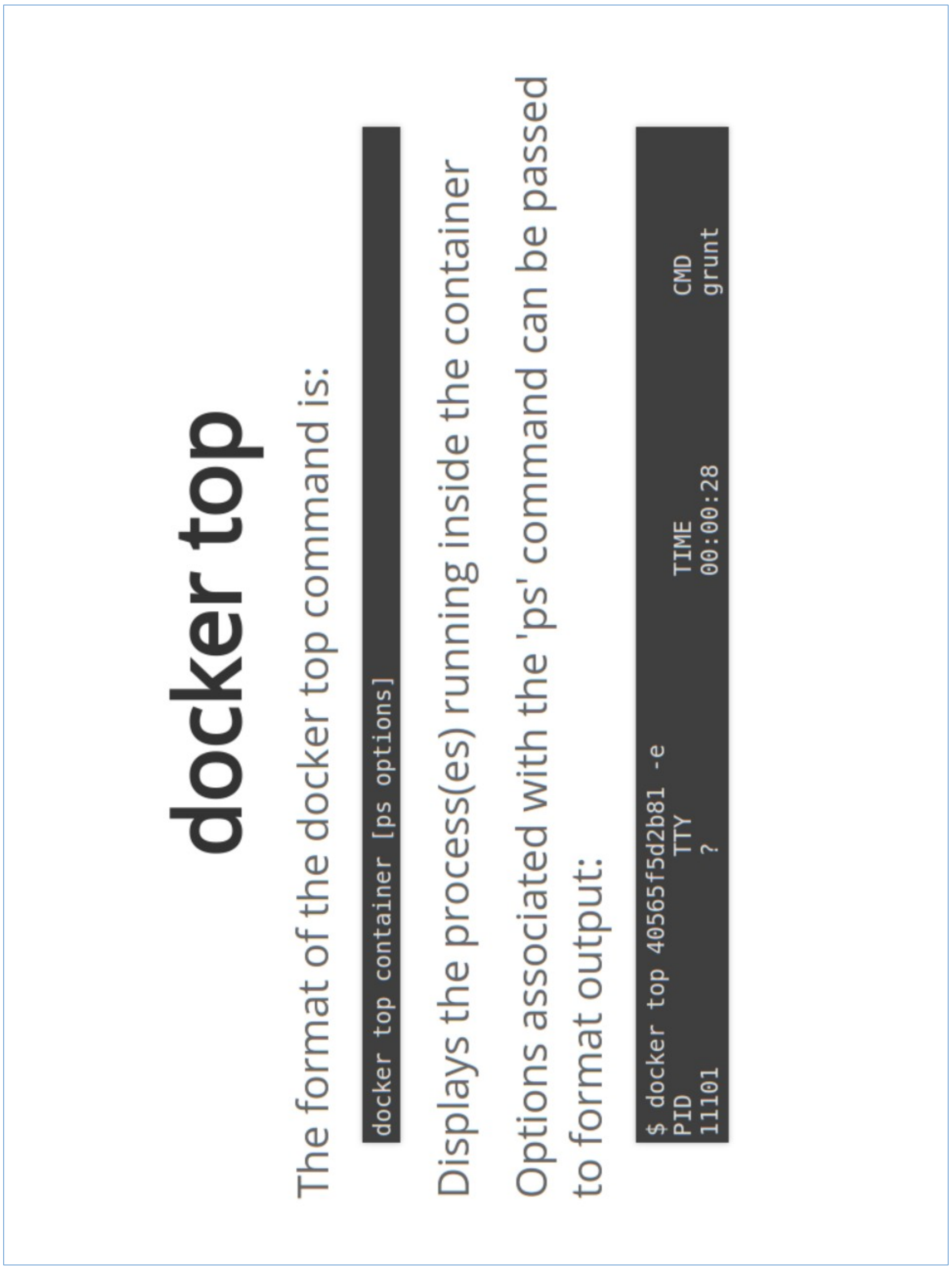
  C: file or directory has been changed or modified
  A: file or directory has been added
  D: file or directory has been deleted

- Example:

- `$ docker diff 08c434e14d9b`

```
C /usr
C /usr/bin
A /usr/bin/editor
A /usr/bin/vi
C /usr/share
C /usr/share/doc
A /usr/share/doc/elvis-tiny
A /usr/share/doc/elvis-tiny/KNOWN.BUGS
A /usr/share/doc/elvis-tiny/changelog.Debian.gz
A /usr/share/doc/elvis-tiny/changelog.gz
A /usr/share/doc/elvis-tiny/copyright
C /usr/share/lintian
C /usr/share/lintian/overrides
A /usr/share/lintian/overrides/elvis-tiny
.
.
.
```

Notes



docker top

The format of the docker top command is:

```
docker top container [ps options]
```

Displays the process(es) running inside the container

Options associated with the 'ps' command can be passed to format output:

```
$ docker top 40565f5d2b81 -e
PID           TTY             TIME            CMD
11101         ?               00:00:28        grunt
```

docker top

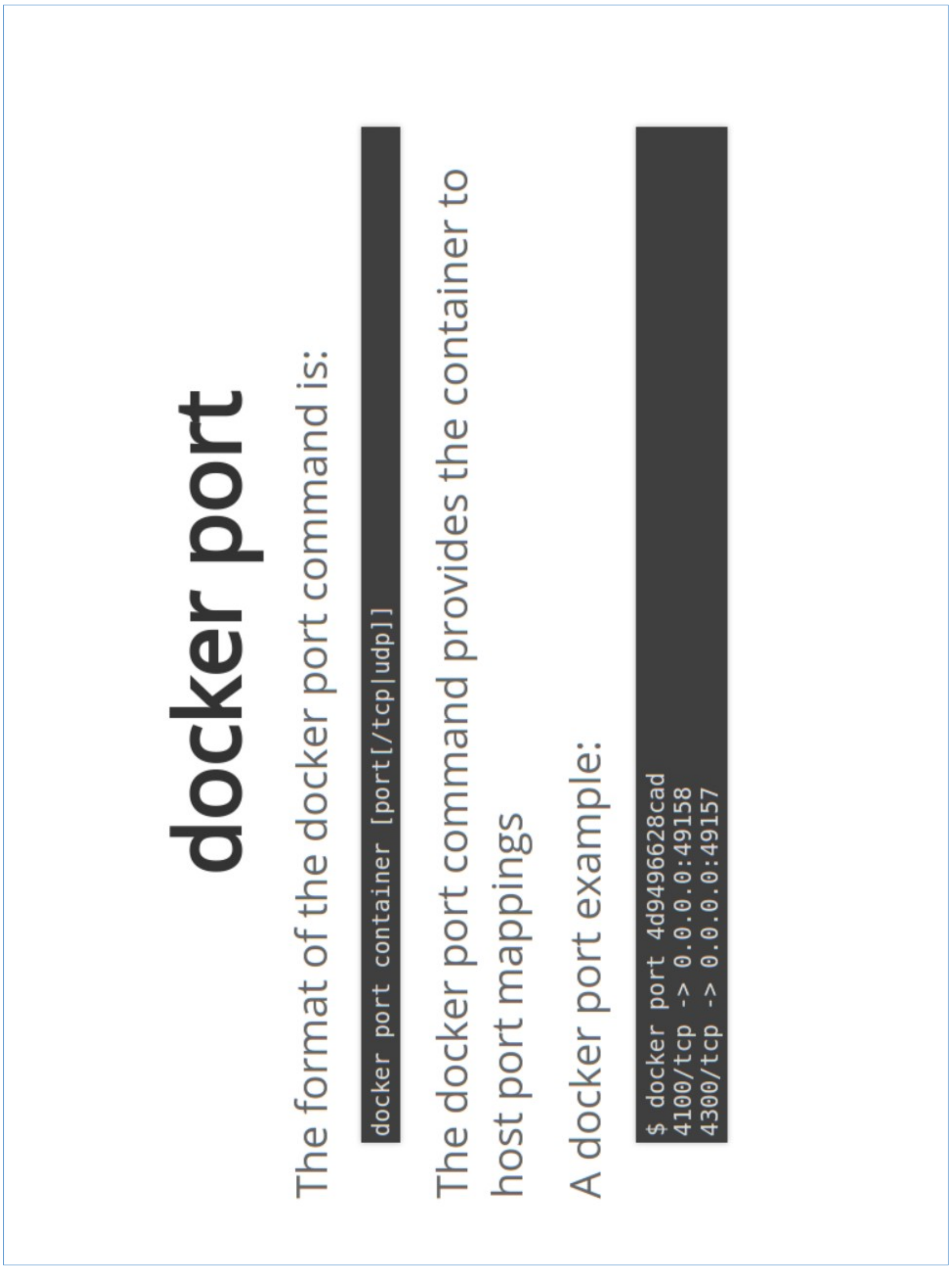
- The `docker top` command provides a listing of the process(es) running inside the specified container
- The `docker top` command is effectively a filtered version of the Linux `ps` command as if it were issued in the default PID namespace
- The format of the `docker top` command is:

```
docker top container [ps options]
```

- The options mimic those of the Linux `ps` command, e.g.

```
$ docker top 40565f5d2b81 -e
PID                 TTY                 TIME                CMD
11101               ?                   00:00:28            grunt
```

Notes



docker port

The format of the docker port command is:

```
docker port container [port[/tcp|udp]]
```

The docker port command provides the container to host port mappings

A docker port example:

```
$ docker port 4d9496628cad
4100/tcp -> 0.0.0.0:49158
4300/tcp -> 0.0.0.0:49157
```

docker port

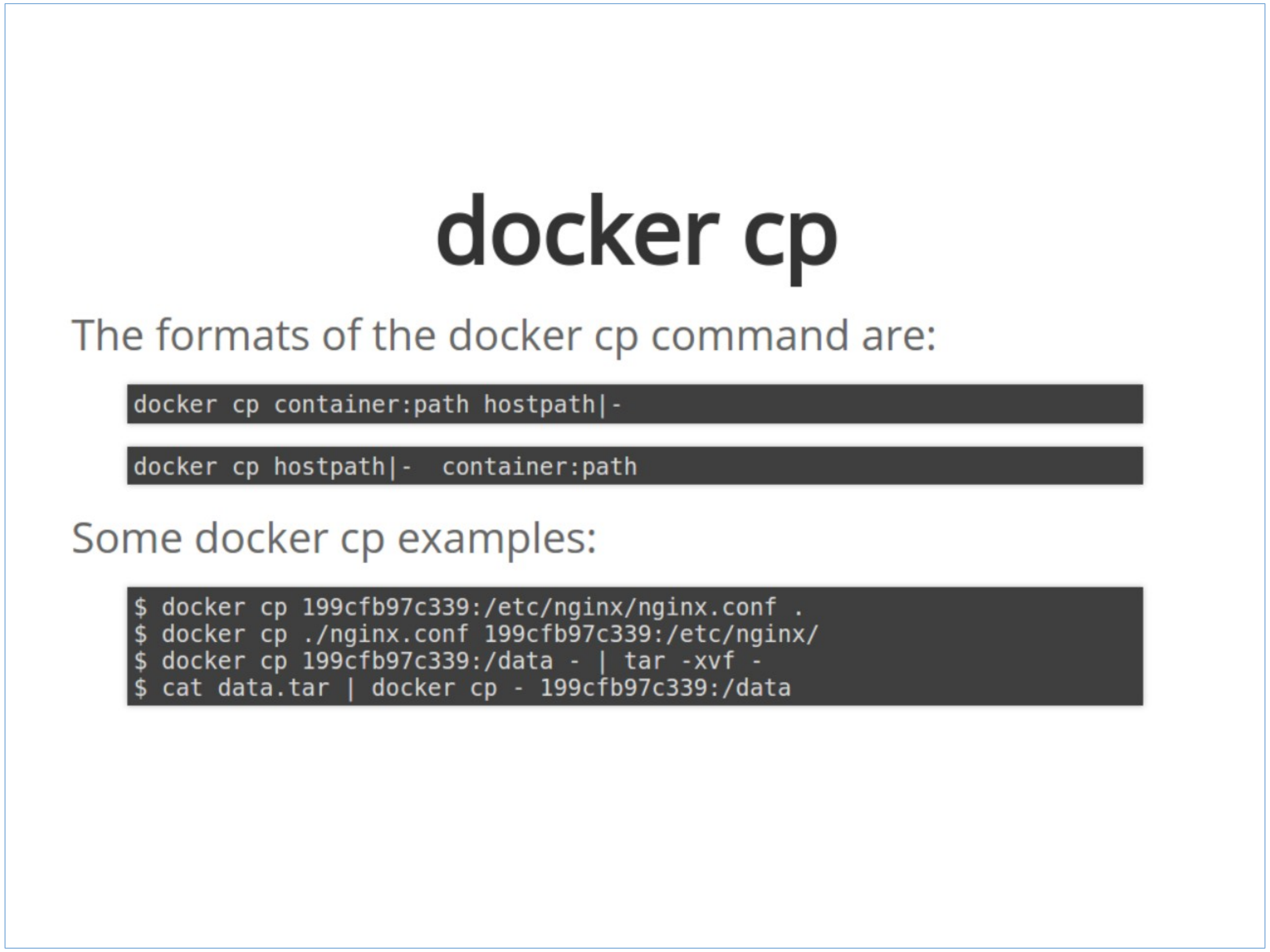
- The `docker port` command details information regarding the ports exposed by the container to the host
- The format for the `docker port` command is:

```
docker port container [port[/tcp|udp]]
```

- Without the port/protocol specified, the command provides all of the container’s port to host mappings, whilst specifying the container’s private port (and optionally, protocol), the command returns the mapping for the specified port
- Example:

```
# docker port 4d9496628cad
4100/tcp -> 0.0.0.0:49158
4300/tcp -> 0.0.0.0:49157
```

Notes



docker cp

The formats of the docker cp command are:

```
docker cp container:path hostpath|-
```

```
docker cp hostpath|-  container:path
```

Some docker cp examples:

```
$ docker cp 199cfb97c339:/etc/nginx/nginx.conf .
$ docker cp ./nginx.conf 199cfb97c339:/etc/nginx/
$ docker cp 199cfb97c339:/data - | tar -xvf -
$ cat data.tar | docker cp - 199cfb97c339:/data
```

docker cp

- The `docker cp` command is used to copy files and directories between the container and the Docker host
- The `docker cp` command takes two forms, depending on the direction of the copy operation:

```
docker cp container:path hostpath|-

docker cp hostpath|-  container:path
```

- In the first form, a file or directory (tree) is copied from the container to a location on the Docker host (or to `stdout` as a tar archive if – is specified)
- In the second form, a file or directory, or tar archive streamed on `stdin`, is copied to the container at the path specified
- The copy operation can be conducted on a stopped or running container
- The container path is interpreted as relative to `/`, whilst the host path can be relative to the current working directory, or absolute
- Files or directories copied from the container to the host are subsequently owned by the UID and primary GID of the user invoking the `docker cp` command
- Files or directories copied from the host to the container are subsequently owned by `root`
- The copy behaviour mimics that of the Linux cp command (with the `-a` option)

Notes