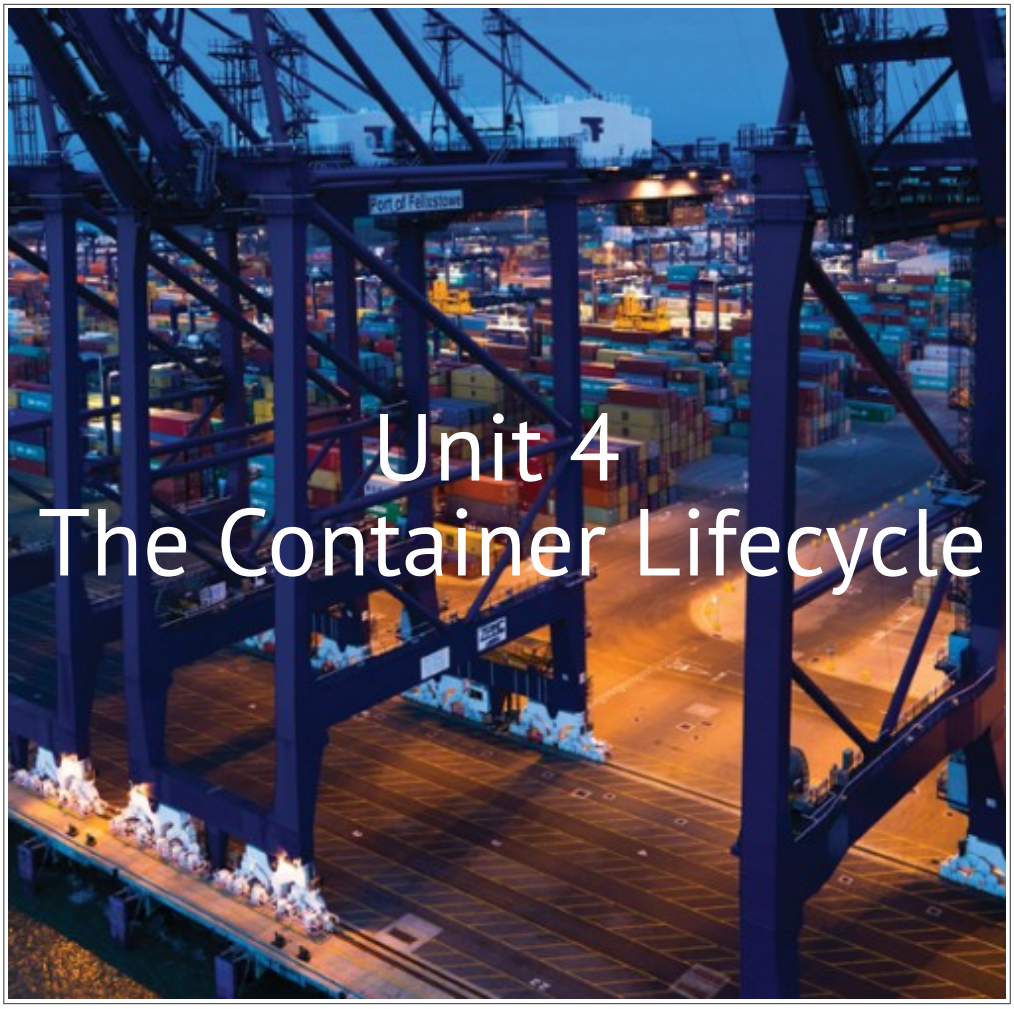


Unit 4
The Container Lifecycle

Objectives

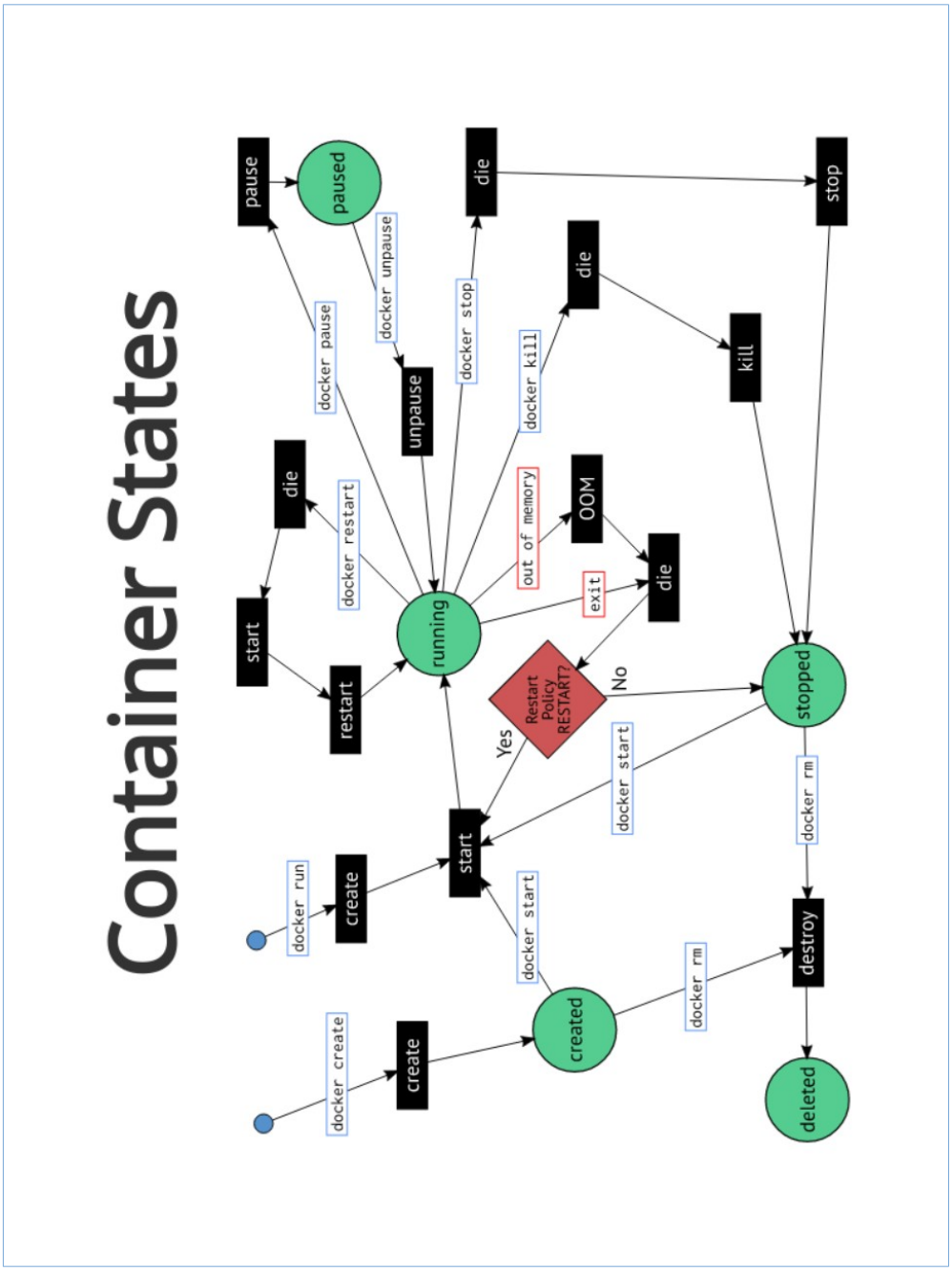
Upon completion of this unit, you will be able to:

- Describe the Docker CLI commands available for managing the lifecycle of a container
- Detail the various configuration options available for fine tuning the operation of containers
- Explain how it is possible to attach to, or execute ad hoc commands in, containers that are already running

Agenda

The following topics will be covered in this unit:

- Container States
- docker create and docker run
- docker start
- docker stop and docker restart
- docker attach
- docker pause and docker unpause
- docker kill
- docker rm
- docker exec
- Lab exercise: Managing the Container Lifecycle


Container States
pause
paused
die
stop
docker unpause
docker pause
docker stop
docker kill
die
kill
unpause
die
start
docker restart
out of memory
OOM
exit
running
die
restart
Restart Policy RESTART?
No
stopped
Yes
docker start
docker rm
start
docker run
create
destroy
docker start
docker rm
docker create
create
created
deleted

Container States

- Containers are ephemeral in nature, and should be treated as immutable objects (pets vs. cattle)
- Their life depends very much on the purpose of the primary process that runs inside the container; is it a long running service, or is it a one-time task that produces an output for some other purpose?
- With Docker, then, containers can exist in numerous states, and often cycle between states
- Containers transition between states based on events, which are prompted by Docker CLI commands, Remote API calls or system-related events
- The event categories are defined in Docker's codebase, are recorded during the daemon's execution, and can be queried by the Docker CLI and Remote API
- The diagram shows how container's come into existence, how they transition between the states (created, running, paused, stopped, deleted), and how they eventually pass into oblivion
- Up until Docker 1.3, creating and starting a container was an atomic step as there was no `docker create` command, but now it is possible to create a container before it is started, and enters a running state
- Containers remain registered with the Docker host until such time as they are deliberately removed from the system
- Containers can cycle between the running, stopped and paused states based on numerous different externally induced events

Notes

docker create/run

The format of the docker create command is:

```
docker create [options] image [command] [args ...]
```

The format of the docker run command is:

```
docker run [options] image [command] [args ...]
```

Both commands share majority of config options

The following config options are exclusive to docker run:

Client Option	Description
`-d, --detached=false`	Run the container in the background
`--rm=false`	Remove container immediately after the container process exits

docker create and docker run

- There are two separate Docker client commands that are used to establish a container from an image
- The `docker create` command enables the creation of a container without actually starting it, allowing for the preparation of a container's configuration ahead of its intended use
- The format of the `docker create` command is:

```
docker create [options] image [command] [args ...]
```

- The `docker run` command encapsulates the capabilities of `docker create` and `docker start`, by establishing a running container from scratch
- The format of the `docker run` command is:

```
docker run [options] image [command] [args ...]
```

- Hence, the library/hello-world image can be run by simply issuing the command:

```
$ docker run hello-world
```

- There are two command line options that are exclusive to `docker run`:

`-d, --detach=false`: specifies that the container is to run in detached mode, which means that it runs in the background (like a daemon process); the default is to run attached in the foreground
`--rm=false`: using this option will result in the container being automatically removed when the container exits

- The `-d`, `--detach` and `--rm` command line options are mutually exclusive

Notes

Runtime Config Options

The following config options influence container runtime:

Client Option	Description
`--addhost=[]`	Adds a hostname/IP address entry to container's /etc/hosts
`--device=[]`	Adds a device from the host into the container
`--entrypoint=""`	Overrides the default binary or command to run in the container
`-e, --env=[]`	Adds environment variable into container runtime environment
`--env-file=[]`	File with multiple environment variables for container's runtime
`--group-add=[]`	Adds additional group for user that owns the container's process
`-h, --hostname=""`	Specifies a specific hostname for the container

docker create and docker run – Common Runtime Configuration Options

The commands `docker create` and `docker run` share a number of configuration options which serve to define a container's runtime environment:

`--addhost=[]`: adds the specified hostname and IP address pair (in the form `hostname:IP`) to the container's private copy of `/etc/hosts` (e.g. `--addhost skut:192.168.1.69`)
`--device=[]`: adds the specified host device to the container (in the form `source:dest:mode` e.g. `--device=/dev/snd:/dev/snd`), where the optional mode represents the cgroup permissions for the device
`--entrypoint=""`: overrides the entrypoint of the image from which the container is created, which is the executable or shell script that is invoked as the container's process (e.g. `--entrypoint=/bin/sh`)
`-e, --env=[]`: adds an environment variable into the container's process runtime environment (e.g. `-e GOPATH=/src/go`)
`--env-file=[]`: rather than specifying multiple environment variables with multiple use of the `-e` command line option, a file can be specified which contains the definition of multiple environment variables, delimited by a newline (e.g. `--entryfile=/usr/share/tango/envs`)
`--group-add=[]`: add group(s) for the owner of the container's process to belong to; the group must exist otherwise the container will not be created
`-h, --hostname=""`: this allows for overriding the default hostname (a shortened version of the container's ID) provided by Docker for the container at runtime (e.g. `-h skut`)

Notes

Runtime Config Options

The following config options influence container runtime:

Client Option	Description
`--log-driver=""`	Specifies container's log driver (default: json-file)
`--log-opt=[]`	Provides logging driver specific options
`--mac-address=""`	Allows setting a specific MAC address
`--oom-kill-disable=false`	Disables the OOM killer for the container's process
`--sig-proxy=true`	Signals are proxied to container's process when no TTY
`--stop-signal="SIGTERM"`	Changes the signal sent to container's process to stop it
`-u, --user=""`	Specifies the user for the container's process
`-w, --workdir=""`	Specifies the working directory for the container's process

docker create and docker run – Common Runtime Configuration Options

Continuing the runtime configuration options that are common to the `docker create` and `docker run` commands:

`--log-driver=""`: specifies an alternative logging driver to the Docker daemon for the container's logs (`none`, `json-file`, `syslog`, `journald`, `gelf`, `fluentd`, `awslogs`); the default is `json-file`
`--log-opt=[]`: provides options to configure logging for the driver specified with `--log-driver`, which are specific to each driver
`--mac-address=""`: overrides the default MAC address provided by Docker for the primary network interface, `eth0` (e.g. `--mac-address=02:1b:f9:cd:e9:e3`); this must be a unicast, locally administered address
`--oom-kill-disable=false`: specifies whether to disable the OOM killer for the container's process
`--sig-proxy=true`: in non-TTY mode, specifies that all received signals are proxied to the container's process
`--stop-signal="SIGTERM"`: specifies the signal to use to stop a running container, which can be given as a number or a name
`-u, --user=""`: specifies the user for the process inside the container, which can be a username or a UID, but must exist within the image from which the container is derived, or must be created by the command that is executed when the container starts (e.g. `-u jbloggs`)
`-w, --workdir=""`: specifies the container's working directory for the process started inside the container (e.g. -w /data/owncloud)

- The daemon can be started with the `--log-driver` and corresponding `--log-opt` options, to specify a default logging arrangement for all containers, for which the container runtime options override

Notes

Constraint Config Options

The following config options constrain the container:

Client Option	Description
`--blkio-weight=0`	Specifies relative weight for block I/) activity
`--cgroup-parent=""`	Path to cgroup parent outside Docker for container
`-c, --cpu-shares=0`	Relative weight for container's access to CPU shares
`--cpu-period=0`	Time period over which to apply quota for container
`--cpuset-cpus=""`	Specifies which CPUs the container's process can run on
`--cpuset-mems=""`	Specifies which mem. nodes container's process can run on
`--cpu-quota=0`	Specifies quota to apply for access to CPU over time period
`--kernel-memory=""`	Assigns limit to kernel memory used by container
`-m, --memory=""`	Assigns a limit to the physical memory used by container
`--memory-reservation=""`	A soft limit for memory used by the container
`--memory-swap=""`	A limit to the swap used by the container
`--memory-swappiness=""`	Sets how aggressively the kernel swaps pages

docker create and docker run – Common Constraint Configuration Options

The commands `docker create` and `docker run` share a number of configuration options, which are controlled by cgroups, and serve to constrain a container's runtime environment:

`--blkio-weight=0`: specifies the cgroup relative weight given to a container's process for block I/O activity; the default option value is 0, which provides a weight of 500, but can be set between 10 and 1000
`--cgroup-parent=""`: specifies a cgroup parent pathname for a cgroup created outside of Docker, under which the container's cgroups will be formed
`-c, --cpu-shares=0`: specifies the cgroup setting for the container's access to CPU resources or CPU shares, is set to 1024 by default, and is a relative weight (e.g. `-c 2048`)
`--cpu-period=0`: specifies the cgroup period (in microseconds) over which to apply the corresponding quota for the container's process
`--cpuset-cpus=""`: specifies which CPU(s) the container can run on (e.g. `--cpuset-cpus="0-1"`)
`--cpuset-mems=""`: specifies which memory node(s) the container can run on (e.g. `--cpuset-mems="1"`)
`--cpu-quota=0`: specifies the cgroup quota (in microseconds) to apply over the corresponding period for the container's process
`--kernel-memory=""`: assigns a limit to the kernel memory that a container will use whilst running, in the form <number><unit>, with the unit being optional (default is b) and having the following values: b, k, m, g (e.g. 512m)
`-m, --memory=""`: assigns a limit to the physical memory that a container will use whilst running, in the form `<number><unit>`, same format as above
`--memory-reservation=""`: specifies memory soft limit, same format as above, should be set less than `-m`, `--memory`
`--memory-swap=""`: assigns a limit to the swap memory that a container can use, takes the same form as above, and is implicitly set to the same value of that set by the `-m, --memory` configuration option when it is used, unless it is explicitly turned off with `–memory-swap=-1`
`--memory-swappiness=""`: specifies how aggressively the kernel swaps pages of memory for the container (0-100); inherits value from parent cgroup

Notes

Namespace Config Options

The following config options alter default namespaces for the container:

Client Option	Description
`--ipc=""`	Container belongs to IPC namespace of host or other container
`--pid=""`	Container belongs to PID namespace of host
`--uts=""`	Container belongs to UTS namespace of host

docker create and docker run – Common Namespace Configuration Options

The commands `docker create` and `docker run` share several configuration options that alter the default namespace configuration of a container's runtime environment:

`--ipc=""`: enables the container to share the same IPC namespace as the host, or another container (e.g. `--ipc=host`, or `–ipc=container:a8236b38ed4f`)
`--pid=""`: allows the container to share the same PID namespace as the Docker host (e.g. `--pid=host`)
`--uts=""`: allows the container to share the same UTS namespace as the Docker host (e.g. `--uts=host`); the container will have the same hostname as the host, and the ability to change the host's hostname

- Docker also allows you to define the NET namespace of a container, but this is a detailed topic covered later

Notes

Security Config Options

The following config options enhance security for the container:

Client Option	Description
`--cap-add=[]`	Add Linux capabilities for the container's process owner
`--cap-drop=[]`	Drop Linux capabilities for the container's process owner
`--privileged=false`	Provide enhanced privileges for the container
`--read-only=false`	Make the container's filesystem read only
`--security-opt=[]`	Apply SELinux or AppArmor security to container

docker create and docker run – Common Security Configuration Options

The commands `docker create` and `docker run` share several configuration options that help to secure the container's runtime environment:

`--cap-add=[]`: specifies a Linux capability to add to the whitelist* of capabilities given to the user of the process in the container (e.g. `--cap-add=KILL` adds the `CAP_KILL` capability for the container's user, whereas `--cap-add=ALL` adds all capabilities for the user)
`--cap-drop=[]`: specifies a Linux capability to drop from the whitelist* of capabilities given to the user of the process in the container (e.g. `--cap-drop=CHOWN` drops the `CAP_CHOWN` capability for the container's user, whereas `--cap-drop=ALL` drops all capabilities for the user)
`--privileged=false`: gives the process in the container extended privileges, particularly to the host's devices, and should be used with extreme caution
`--read-only=false`: makes the container's root filesystem read-only instead of read-write
`--security-opt=[]`: specifies a custom SELinux label or AppArmor profile to apply to the container (e.g. `--security-opt apparmor:docker-default`)

* CAP_CHOWN, CAP_DAC_OVERRIDE, CAP_FSETID, CAP_FOWNER, CAP_MKNOD, CAP_NET_RAW, CAP_SETGID, CAP_SETUID, CAP_SETFCAP, CAP_SETPCAP, CAP_NET_BIND_SERVICE, CAP_SYS_CHROOT, CAP_KILL, CAP_AUDIT_WRITE

Notes

I/O Config Options

The following config options enable client I/O with the container:

Client Option	Description
`-a, --attach=[]`	Attach to a container's stdin, and/or stdout and/or stderr
`-i, --interactive=false`	Make interactive connection by attaching to stdin
`-t, --tty=false`	Assign a pseudo-TTY terminal to the container

docker create and docker run – Common I/O Configuration Options

The commands `docker create` and `docker run` share several configuration options that enable I/O between the Docker client and the container's runtime environment:

`-a, --attach=[]`: attach the client to the container's `stdin`, `stdout` and `stderr`; `stdout` and `stderr` are attached by default, and `stdin` is attached if the `-i` command line option is specified
`-i, --interactive=false`: specifies that the container is interactive; i.e. `stdin` is attached
`-t, --tty=false`: specifies that a pseudo-TTY terminal is assigned to the container

- The `-a, --attach` configuration option should only be used for the use case where the client is to be selectively attached to a sub-set of the streams
- In most situations where I/O with the container is expected, it is normal practice to specify both `-i` and `-t` options, which can be combined as can all versions of the short command that don't require additional parameters; e.g. `-it`

Notes

Other Config Options

The following miscellaneous config options can be applied to the container:

Client Option	Description
`--cidfile=""`	Write the container's ID to the specified file
`--disable-content-trust=true`	Image tag content trust for Docker Hub is disabled
`-l, --label=[]`	Define metadata label for container as key/value pair
`--label-file=[]`	Provide a file containing labels for a container
`--lxc-conf=[]`	Add LXC specific options when using LXC exec driver
`--name=""`	Override daemon provided container name
`--restart=""`	Defines the restart policy to apply to container
`--ulimit=[]`	Specifies ulimits to apply to the container

docker create and docker run – Other Common Configuration Options

The commands `docker create` and `docker run` also share a few miscellaneous configuration options:

`--cidfile=""`: specifies a file on the host system, where the container's ID is written (e.g. `--cidfile=/var/run/docker/container.id`)
`--disable-content-trust=true`: specifies that the Docker Hub content trust capability for signing image tags is disabled
`-l, --label=[]`: defines metadata associated with a container in the form of a key/value pair (e.g. `-l environment=prod`)
`--label-file=[]`: provides a file containing labels to be applied as the container's metadata
`--lxc-conf=[]`: adds LXC specific configuration options where the LXC driver is used ahead of Docker's native execution driver
`--name=""`: specifies a name to assign to a container in place of the auto generated name provided by Docker
`--restart=""`: defines the container's restart policy in the event that the container exits, and can be one of `no` (the default), `on-failure[:max-retry]`, `always`, `unless-stopped`
`--ulimit=[]`: specifies a list of ulimit parameters to apply to the container (e.g. `nofile`, `noproc`)

- The `--restart` command line option cannot be used in conjunction with the `--rm` command line option with `docker run`
- The `docker rename` command can be used to change a container's existing name:

```
docker rename oldName newName
```

- The ulimit parameters are set with the following format, `type=soft[:hard]` (e.g. `msgqueue=204800:409600`. If the `hard` value is omitted, it takes the same as the `soft` value.
- The daemon can be started with one or more `--default-ulimit` configuration options, which provide ulimit defaults, which are overridden by equivalent container runtime configuration options

Notes

docker start

The format of the docker start command is:

```
docker start [options] container [container ...]
```

The following config options are available for docker start:

Client Option	Description
`-a, --attach=false`	Attach to container's stdout and stderr, and forward signals
`-i, --interactive=false`	Make container interactive by connecting to stdin

docker start

- The `docker start` command allows for the instantiation of a previously configured container instance, resulting in a running process(es) isolated in each of a MNT, NET, UTS, PID and IPC namespace
- The format of the `docker start` command is:

```
docker start [options] container [container ...]
```

- The `docker start` command has two available configuration options:

`-a, --attach=false`: specifies that the container's `stdout` and `stderr` are connected to the client, and that signals originating in the client are forwarded to the container's main process; the default setting is false
`-i, --interactive=false`: specifies that the container's `stdin` is attached to the client

Notes

docker stop/restart

The format of the docker stop command is:

```
docker stop [option] container [container ...]
```

Default signal is SIGTERM, which can be overridden using --stop-signal config option on creation

The format of the docker restart command is:

```
docker restart [option] container [container ...]
```

The following config option is available for both commands:

Client Option	Description
`-t, --time=10`	Grace period before sending SIGKILL to stop container's process

docker stop and docker restart

- The `docker stop` command simply stops a running container by sending a `SIGTERM` signal, and if that fails it is followed with a `SIGKILL` signal after a grace period (default 10 seconds)
- If the `--stop-signal` was specified when the container was created, as a configuration option to `docker create` or `docker run`, this signal is sent instead of `SIGTERM`
- The format of the `docker stop` command is:

```
docker stop [option] container [container ...]
```

- The `docker restart` command stops a running container, and then attempts to restart it, and the format of the command is:

```
docker restart [option] container [container ...]
```

- Both commands have the same, single, optional configuration option:

`-t, --time=10`: specifies the grace period for `SIGTERM` (or other configured signal) to take effect before sending `SIGKILL`

Notes

docker attach

The format of the docker attach command is:

```
docker attach [options] container
```

To disconnect from a container, use the CTL-P CTL-Q sequence or CTL-C if --sig-proxy=false

The following config options are available for docker attach:

Client Option	Description
`--no-stdin=false`	Do not attach to container's stdin
`--sig-proxy=true`	In non-TTY mode, proxy all received signals to the container

docker attach

- The `docker attach` command allows you to connect to a running container, which was started in detached mode
- It provides access to the `stdin`, `stdout` and `stderr` of the container's main process (PID 1)
- It's possible to attach to the container multiple, simultaneous times (even from different hosts), with the same output being displayed on each 'attachment', and each 'attachment' being able to provide input to the container
- When attached to a container, It's possible to detach from it by issuing the sequence, `CTL-P CTL-Q`, or by using `CTL-C` if `--sig-proxy` is set `false`
- Detaching from the container using `CTL-P CTL-Q` will only work, however, if the container is started in interactive mode, and has a pseudo-TTY
- Whilst the default behaviour is to attach to the container's `stdin`, it's possible to override this by using the `docker attach` command line option `--no-stdin=true`, which means a SIGKILL needs to be sent to the client process in order to detach it from the container
- By default, in non-TTY mode, signals received by the attached client are proxyed to the container's process (excluding `SIGCHLD`, `SIGKILL`, `SIGSTOP`), but this behaviour can be overridden using the command line option `--sig-proxy=false`
- Care should be taken when disconnecting from the container, to ensure that the container's process is not inadvertently terminated
- The format of the `docker attach` command is:

```
docker attach [options] container
```

`--no-stdin=false`: specifies that when attaching to the container, do not attach the client to the container's `stdin`
`--sig-proxy=true`: in non-TTY mode, specifies that all signals received by the client are proxied to the container

Notes

docker pause/unpause

The format of the docker pause/unpause commands are:

```
docker pause container
```

```
docker unpause container
```

docker pause and docker unpause

- The `docker pause` command halts the execution of the container's process(es), preserving its state
- Temporarily pausing a container's execution can be useful for establishing control over system resources
- Docker uses the cgroups 'freezer' controller to freeze the tasks in the cgroup hierarchy
- This method is considered superior to using the traditional `SIGSTOP` and `SIGCONT` signals, as processes can be made aware of these signals, which may result in unsolicited behaviour
- The `docker unpause` command simply returns the container's process(es) to a running state
- There are no configuration options for either `docker pause` or `docker unpause`, and the format of the command is:

```
docker pause container
docker unpause container
```

Notes

docker kill

The format of the docker kill command is:

```
docker kill [option] container [container ...]
```

The following config option is available for docker kill:

Client Option	Description
`-s, --signal=""`	Send specified signal to container (default: SIGKILL)

docker kill

- The `docker kill` command sends a specified signal to the container's main process
- The format of the `docker kill` command is:

```
docker kill [option] container [container ...]
```

- The single configuration option available for the command is:

`-s, --signal=""`: specifies the signal to send to the container's main process (the default is `SIGKILL`)

- The value of the command line option can be an abbreviation of the signal string, i.e. `STOP` instead of `SIGSTOP`

Notes

docker rm

The format of the docker rm command is:

```
docker rm [options] container [container ...]
```

Destroys the container, and unregisters from the daemon

The following config options are available for docker rm:

Client Option	Description
`-f, --force=false`	Forcefully remove a container even though it is running
`-l, --link=false`	Remove the link that exists between two containers
`-v, --volumes=false`	Remove any volumes uniquely referenced by container

docker rm

- By the very nature of Docker, the containers that are created to perform tasks are often transient, and unless they are invoked with the `--rm` configuration option of `docker run`, stopped containers will remain registered with Docker's daemon, consuming disk space on the host
- Docker provides the `docker rm` command for removing spent containers that no longer serve a purpose
- The format of the `docker rm` command is:

```
docker rm [options] container [container ...]
```

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

`-f, --force=false`: ordinarily, a container needs to be in a stopped state before it can be removed, but a running container can be removed if this option is specified (container is forcibly stopped with `SIGKILL` signal)
`-l, --link=false`: specifies that the link is removed rather than the underlying container
`-v, --volumes=false`: specifies that any volumes associated with the container are also removed (this will fail if any other containers currently reference any volumes referenced by the container being removed)

Notes

docker exec

The format of the docker exec command is:

```
docker exec [options] container command [arg ...]
```

The following config options are available for docker exec:

Client Option	Description
`-d, --detach=false`	Execute command in detached mode
`-i, --interactive=false`	Make the new process in the container interactive
`--privileged=false`	Raise the capabilities provided for the new command
`-t, --tty=false`	Assign a pseudo-TTY to the new process
`-u, --user=""`	Execute command as user with optional group membership

docker exec

- The `docker exec` command allows for the client to establish a process inside an already running container
- Introduced with Docker 1.3, this command satisfies a need that has existed from the early days of Docker, namely, "how can I get inside my running container?"
- The command can be run in detached mode, which is akin to injecting a long -running process into the container
- It can also be run interactively, perhaps for the purposes of administration or troubleshooting
- If a process is established inside a container with the `docker exec` command, and the container subsequently restarts, the process will not get restarted
- The format of the `docker exec` command is:

```
docker exec [options] container command [arg ...]
```

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

`-d, --detach=false`: specifies that the command is run in the background, which is false by default
`-i, --interactive=false`: specifies that the new process is interactive; i.e. `stdin` is attached
`--privileged=false`: provide extended privileges for the command executed in the container
`-t, --tty=false`: specifies that a pseudo-TTY terminal is assigned to the new process
`-u, --user=""`: run command as specified user and optional group

- The format for the `-u, --user` configuration option can be any combination of `user[:group]`, where the user element is one of user or uid, and the optional group element is one of group or gid (e.g. `jbloggs:adm`, `jbloggs:4`, `1000:adm`, `1000:4`, `jbloggs`, `1000`)

Notes