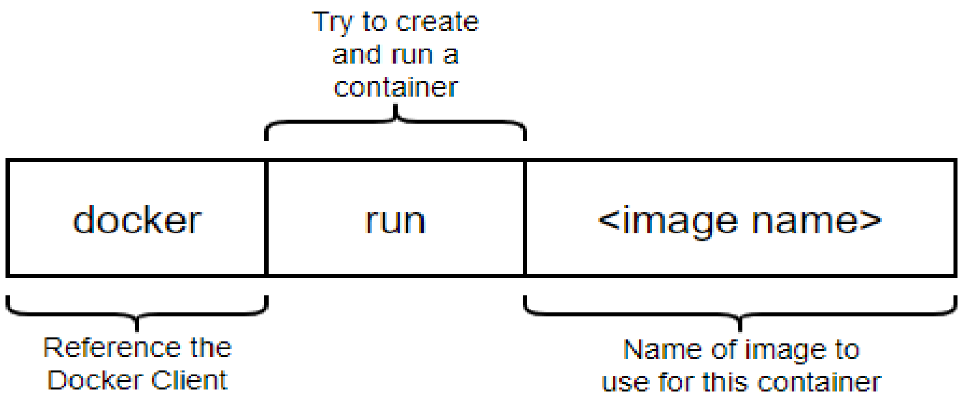
Docker Basic Commands

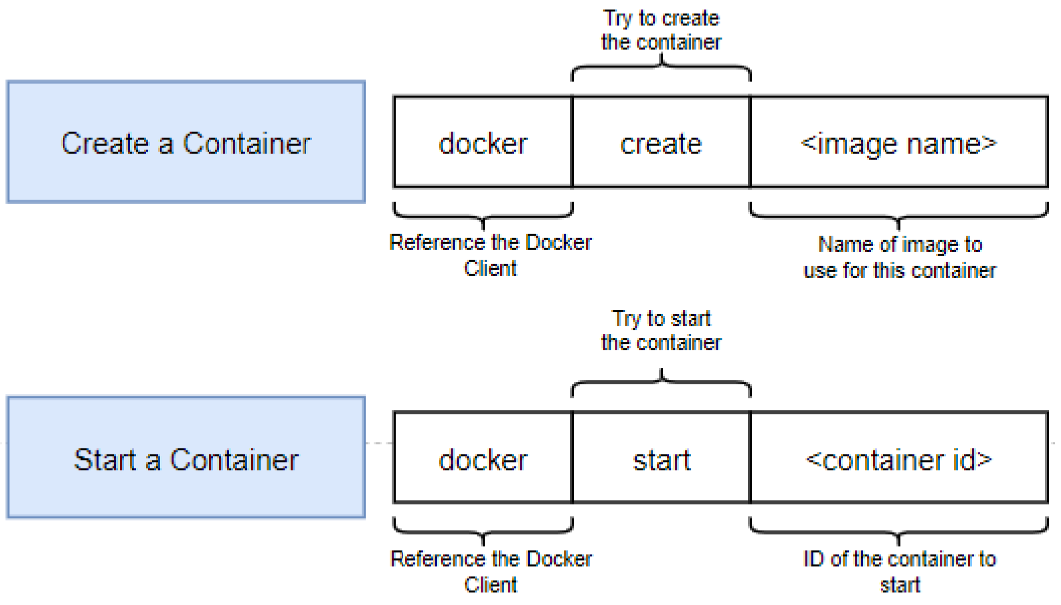
1. Container life cycle

1.1 docker run


Try to create and run a container
docker
run
<image name>
Reference the Docker Client
Name of image to use for this container

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

1.2 Creating and Running a Container from an Image


Try to create the container
Create a Container
docker
create
<image name>
Reference the Docker Client
Name of image to use for this container
Try to start the container
Start a Container
docker
start
<container id>
Reference the Docker Client
ID of the container to start

Docker create:

```
$ docker create hello-world
8e6d758510631a023582396f92c1f749dd08a163a627c2cfeee3a5f5c1ab374e
```

Docker start:

```
$docker start 8e6d758510631a023582396f92c1f749dd08a163a627c2cfeee3a5f5c1ab374e
8e6d758510631a023582396f92c1f749dd08a163a627c2cfeee3a5f5c1ab374e
```

Docker start with -a:

-a attaches the container output to console

```
$docker start -a 402c5a4611068fe4e0a14372b44f76a84430c5af300a71fdb15e68e8297bbc3f

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.
```

```
To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

2. Overriding the default command of image

- Syntax:

  docker run <image_name> <cmd>

  Example: docker run busybox ls

- We can override only those cmd which file system of image support.
  For example we cannot use ls cmd for hello world image
- We can override the command only when we are creating a container. We cannot override the default command when we are restarting the container.

3. Listing containers

docker ps [OPTIONS]

Options

Name, shorthand	Default	Description
--all , -a		Show all containers (default shows just running)
--filter , -f		Filter output based on conditions provided
--format		Pretty-print containers using a Go template
--last , -n	-1	Show n last created containers (includes all states)
--latest , -l		Show the latest created container (includes all states)
--no-trunc		Don't truncate output
--quiet , -q		Only display container IDs
--size , -s		Display total file sizes

```
$ docker ps
CONTAINER ID   IMAGE    COMMAND   CREATED   STATUS    PORTS     NAMES

PS C:\Users\Sneha Y V> docker ps -all
CONTAINER ID   IMAGE        COMMAND    CREATED            STATUS                      PORTS     NAMES
f8dde35c1c13   hello-world   "/hello"   About a minute ago   Exited (0) About a minute ago            elated_sinoussi


PS C:\Users\Sneha Y V> docker ps -a
CONTAINER ID   IMAGE                        COMMAND             CREATED            STATUS                      PORTS        NAMES
 f8dde35c1c13   hello-world                    "/hello"             About a minute ago   Exited (0) About a minute ago              elated_sinoussi
```

```
5e3c29cdce05   redis                    "docker-entrypoint.s…"   5 days ago         Exited (0) 5 days ago                quirky_panini
```

For more refer : https://docs.docker.com/engine/reference/commandline/ps/

4. Restarting the container

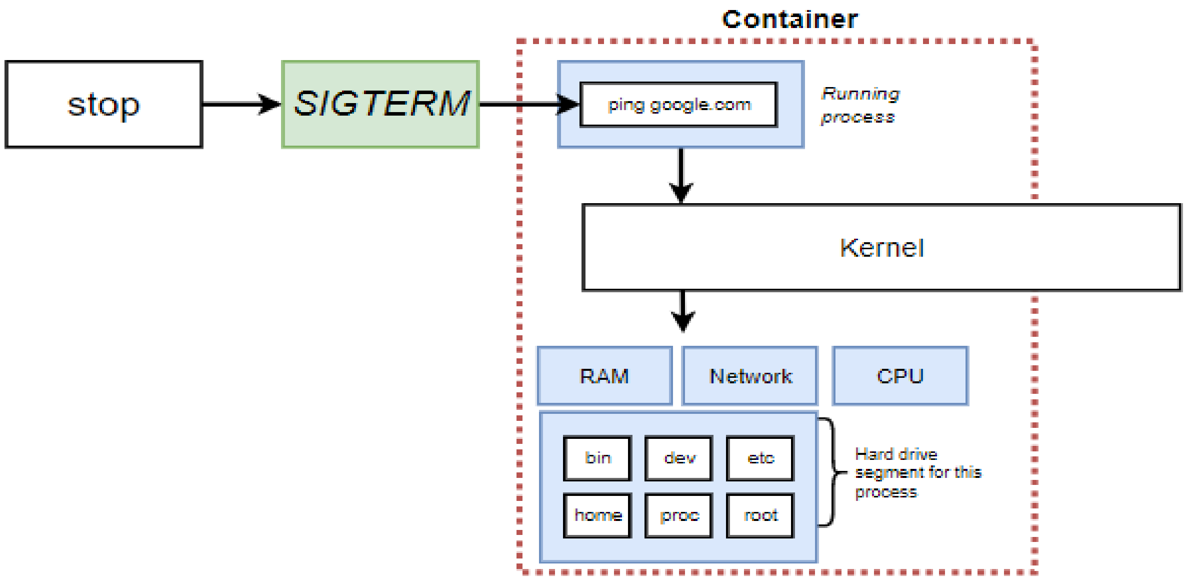
docker start [options] <containerId>
We cannot override default command here

5. Stopping the container

5.1 Stopping container

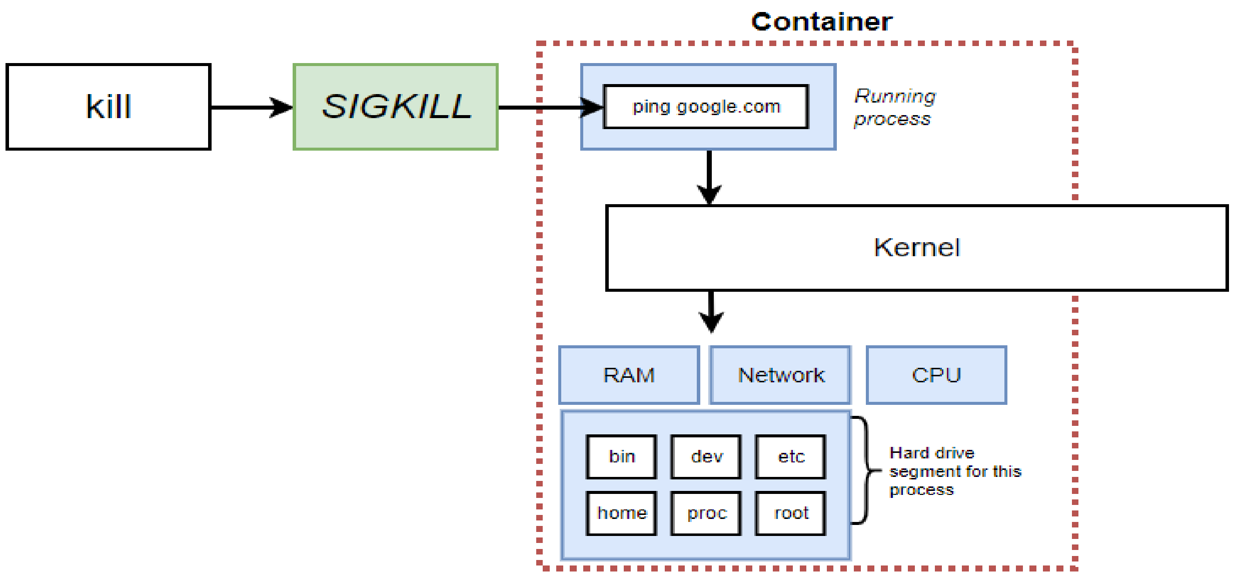
It will wait for some time to clean up.
If within 10 sec it is not done then kill command is called.

- $docker stop <containerId>



5.2 Killing container

- $docker kill <containerId>



6. Retrieving Log Outputs

$docker logs <containerId>

```
$docker create busybox echo hi everyone
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
2c39bef88607: Pull complete
Digest:
sha256:20142e89dab967c01765b0aea3be4cec3a5957cc330f061e5503ef6168ae6613
Status: Downloaded newer image for busybox:latest
324f78539aacfcff295ab40701e859b4b564d63fe3fa1be5681653d6a8f2c3d9


$ docker start 324f78539aacfcff295ab4
324f78539aacfcff295ab4

$ docker logs 324f78539aacfcff295ab4
hi everyone
```

7. Removing stopped container

$docker system prune

This will remove all the containers, even image caches. After running this command we need to redownload the image if we want to run the container again

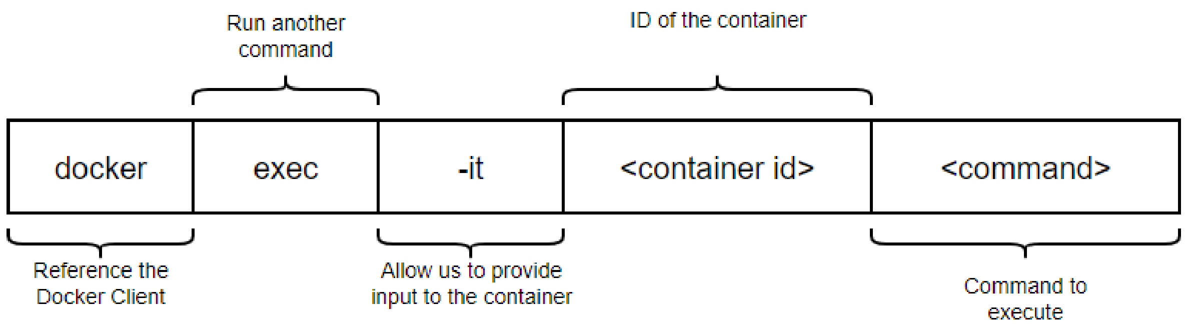
```
PS C:\Users\Sneha Y V> docker system prune
```

```
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all dangling images
  - all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted Containers:
402c5a4611068fe4e0a14372b44f76a84430c5af300a71fdb15e68e8297bbc3f
8e6d758510631a023582396f92c1f749dd08a163a627c2cfeee3a5f5c1ab374e

Deleted Networks:
minikube

Total reclaimed space: 3.166MB
```

8. Run a command for running container

`$docker exec [OPTIONS] CONTAINER COMMAND [ARG...]`

Execute an additional command in a container



Example:

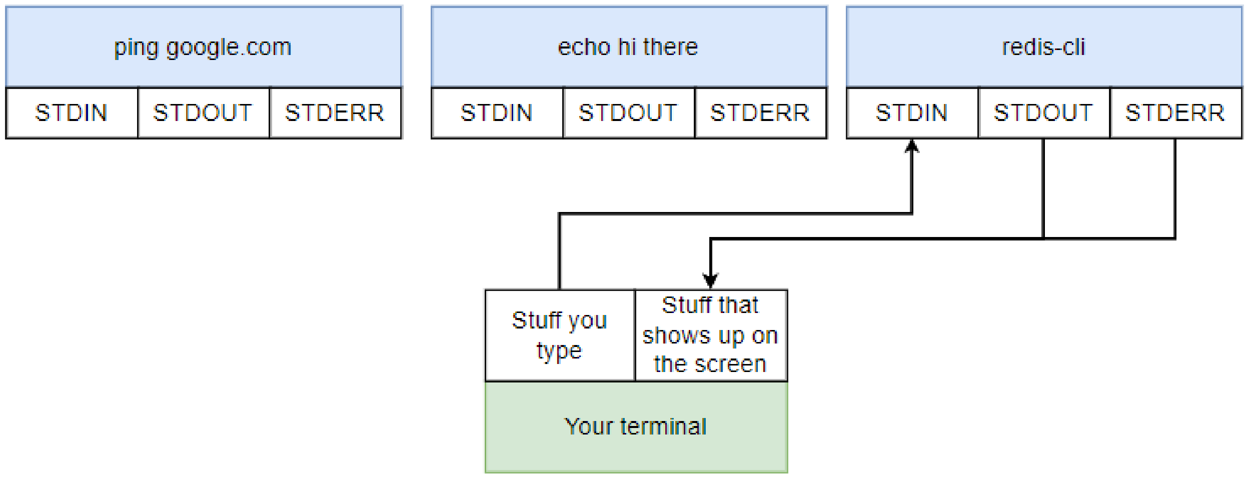
```
$docker run redis
1:C 04 Sep 2022 11:55:37.976 # oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
1:C 04 Sep 2022 11:55:37.976 # Redis version=7.0.4, bits=64, commit=00000000,
modified=0, pid=1, just started
1:C 04 Sep 2022 11:55:37.976 # Warning: no config file specified, using the
default config. In order to specify a config file use redis-server
/path/to/redis.conf
1:M 04 Sep 2022 11:55:37.976 * monotonic clock: POSIX clock_gettime
1:M 04 Sep 2022 11:55:37.980 * Running mode=standalone, port=6379.
1:M 04 Sep 2022 11:55:37.981 # Server initialized
1:M 04 Sep 2022 11:55:37.981 # WARNING overcommit_memory is set to 0!
Background save may fail under low memory condition. To fix this issue add
```

```
'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the
command 'sysctl vm.overcommit_memory=1' for this to take effect.
1:M 04 Sep 2022 11:55:37.982 * Ready to accept connections
```

```
$docker ps
CONTAINER ID   IMAGE    COMMAND                  CREATED          STATUS          PORTS      NAMES
6ae94d73858d   redis    "docker-entrypoint.s…"   40 seconds ago   Up 38 seconds   6379/tcp naughty_mayer

$docker exec -it 6ae94d73858d redis-cli
127.0.0.1:6379> set name sneha
OK
127.0.0.1:6379> get name
"sneha"
```

9. Interactive mode

- All the process in linux are connected to 3 type of signals



- -it contains 2 flags namely -i (to connect to STDIN signal) and -t(in general to format the displayed content in console)

```
1. Without -t flag : there is no auto filling
   $docker exec -i 6ae94d73858d redis-cli
   set name bob
   OK
   get name
   bob
```

10. Getting a command prompt in container

```
1. $docker exec -it 6ae94d73858d sh
```

```
# ls
# cd /
# ls
bin  boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp
usr  var
# redis-cli
127.0.0.1:6379> set test "Testing docker command prompt"
OK
127.0.0.1:6379> get test
"Testing docker command prompt"
```

2. $docker run -it busybox sh

```
/ # echo hi everyone
hi everyone
```

11. Container isolation

- Any command runned on a 1 container will not impact other
- Example: file created in 1 container will not be present in another

Terminal 1:

$docker run -it busybox sh

```
/ # touch terminal1
/ # ls
bin       dev       etc       home      proc      root      sys       terminal1  tmp       usr
var
```

Terminal 2:

$docker run -it busybox sh

```
/ # touch terminal1
/ # ls
bin       dev       etc       home      proc      root      sys       terminal2  tmp       usr
var
```

Terminal 3:

$docker ps

```
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS          PORTS     NAMES
9a87838869ba   busybox   "sh"      47 seconds ago   Up 45 seconds             great_ganguly
43155b0d1d71   busybox   "sh"      2 minutes ago    Up 2 minutes              sleepy_edison
```

Reference

https://docs.docker.com/engine/reference/commandline