Working with containers

Starting containers

To see the running containers (which are just processes) we use:

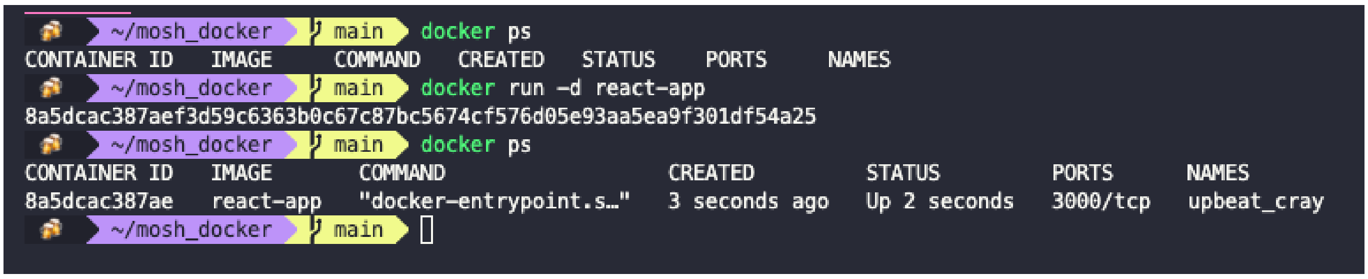
```
docker ps
```

To run a new container we use:

```
docker run IMAGE
```

To run a container in the detached mode (in the background) we use:

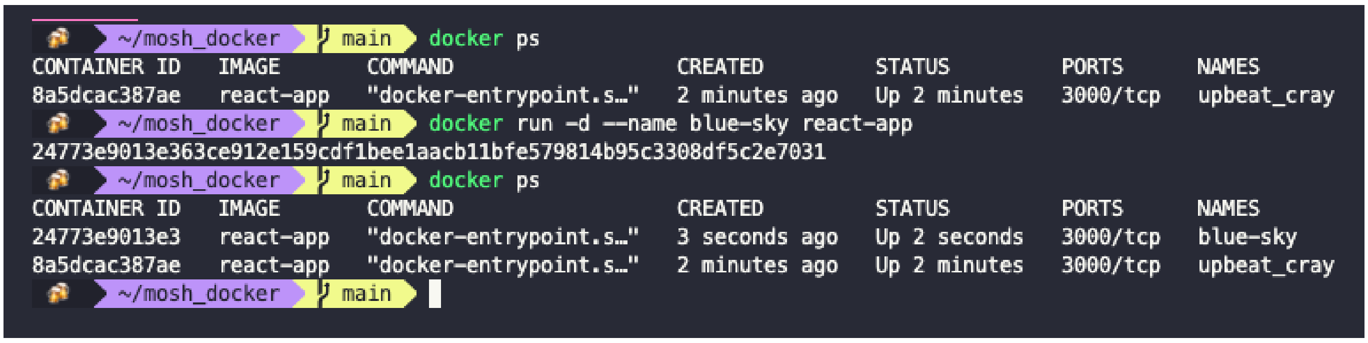
```
docker -d run IMAGE
```



```
 ~/mosh_docker  main  docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
 ~/mosh_docker  main  docker run -d react-app
8a5dcac387aef3d59c6363b0c67c87bc5674cf576d05e93aa5ea9f301df54a25
 ~/mosh_docker  main  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS      NAMES
8a5dcac387ae   react-app   "docker-entrypoint.s…"   3 seconds ago   Up 2 seconds   3000/tcp   upbeat_cray
 ~/mosh_docker  main 
```

Docker automatically assigns each container a random name. We can give it a name by running:

```
docker run -d --name NAME IMAGE
```



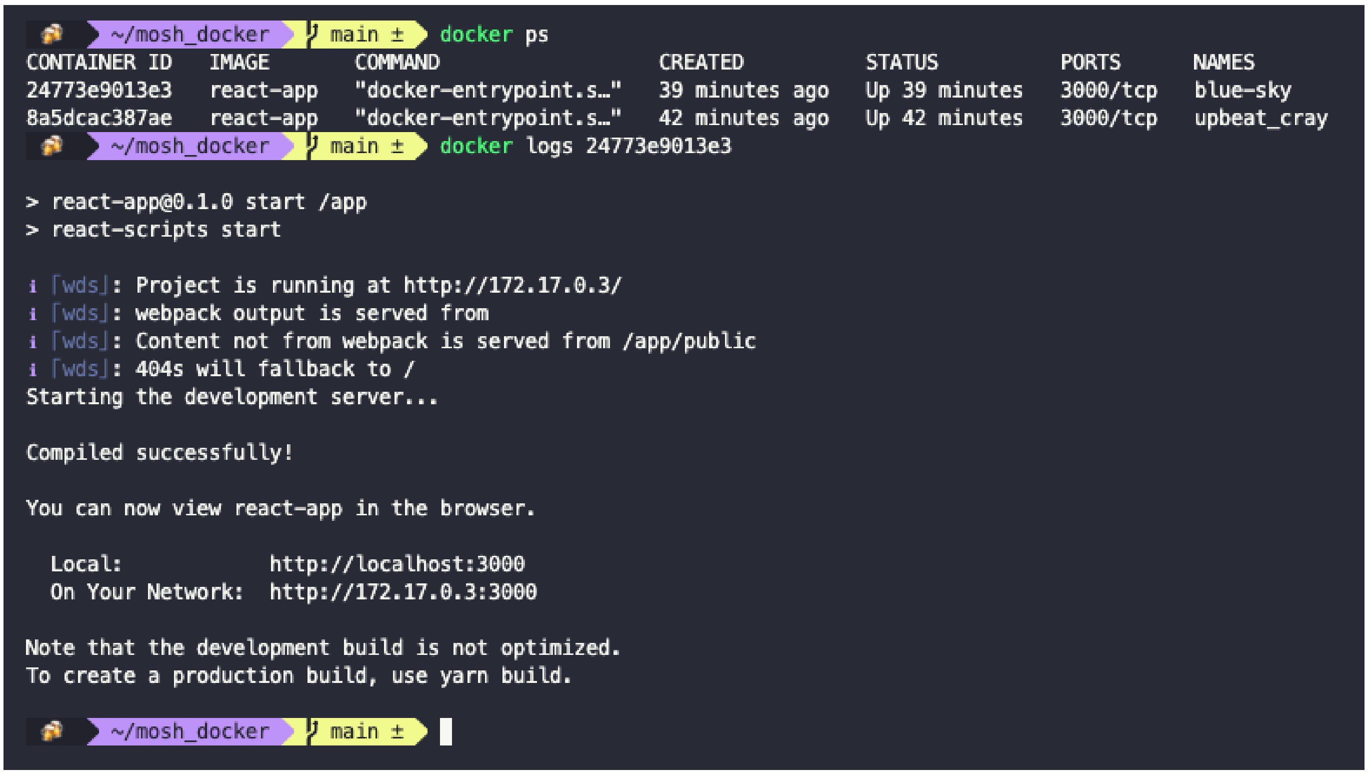
```
 ~/mosh_docker  main  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS      NAMES
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   3000/tcp   upbeat_cray
 ~/mosh_docker  main  docker run -d --name blue-sky react-app
24773e9013e363ce912e159cdf1bee1aacb11bfe579814b95c3308df5c2e7031
 ~/mosh_docker  main  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED         STATUS         PORTS      NAMES
24773e9013e3   react-app   "docker-entrypoint.s…"   3 seconds ago   Up 2 seconds   3000/tcp   blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 minutes ago   Up 2 minutes   3000/tcp   upbeat_cray
 ~/mosh_docker  main 
```

v

Viewing logs

To view the logs of a container we use:

```
docker logs CONTAINER_ID
```



```
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS      NAMES
24773e9013e3   react-app   "docker-entrypoint.s…"   39 minutes ago   Up 39 minutes   3000/tcp   blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   42 minutes ago   Up 42 minutes   3000/tcp   upbeat_cray
 ~/mosh_docker  main ±  docker logs 24773e9013e3

> react-app@0.1.0 start /app
> react-scripts start

ℹ ｢wds｣: Project is running at http://172.17.0.3/
ℹ ｢wds｣: webpack output is served from
ℹ ｢wds｣: Content not from webpack is served from /app/public
ℹ ｢wds｣: 404s will fallback to /
Starting the development server...

Compiled successfully!

You can now view react-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.17.0.3:3000

Note that the development build is not optimized.
To create a production build, use yarn build.

 ~/mosh_docker  main ± 
```

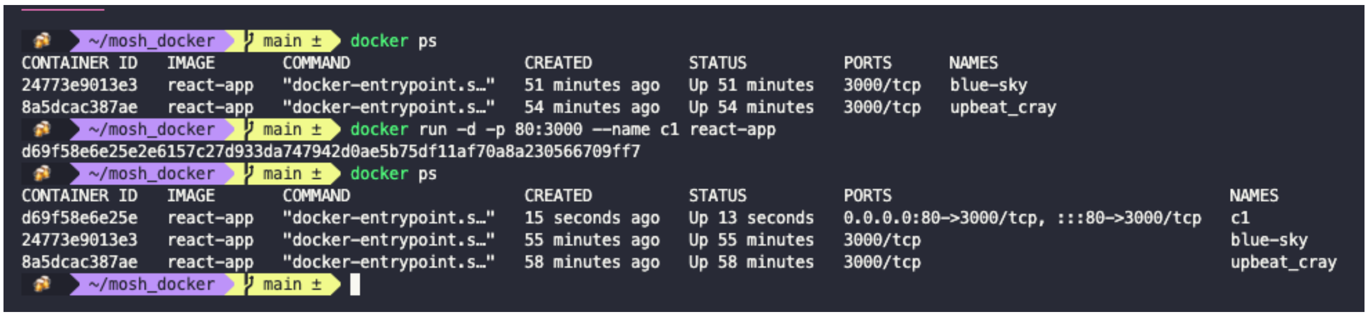
This command has some additional options.

- With the `-f` flag we can follow the log. This will allow us to see on the terminal whatever is written to the log.
- With the `-n` flag we can get just the last `n` lines of the log
- With the `-t` flag we can add timestamps to the log entries.

Publishing ports

To send traffic from the container to the local host we need to publish a port. As you can see above, containers are not published by default (their value is `3000/tcp` because we specified that on the `Dockerfile`). To publish a port we use the `-p` flag to publish a port in on the host to a port in the container.

```
docker run -d -p LOCAL_PORT:CONTAINER_PORT --name NAME IMAGE
```



```
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS      NAMES
24773e9013e3   react-app   "docker-entrypoint.s…"   51 minutes ago   Up 51 minutes   3000/tcp   blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   54 minutes ago   Up 54 minutes   3000/tcp   upbeat_cray
 ~/mosh_docker  main ±  docker run -d -p 80:3000 --name c1 react-app
d69f58e6e25e2e6157c27d933da747942d0ae5b75df11af70a8a230566709ff7
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                 NAMES
d69f58e6e25e   react-app   "docker-entrypoint.s…"   15 seconds ago   Up 13 seconds   0.0.0.0:80->3000/tcp, :::80->3000/tcp   c1
24773e9013e3   react-app   "docker-entrypoint.s…"   55 minutes ago   Up 55 minutes   3000/tcp                              blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   58 minutes ago   Up 58 minutes   3000/tcp                              upbeat_cray
 ~/mosh_docker  main ± 
```

Executing commands in running containers

When we start a container it executes the default command that we specify in the `Dockerfile`. If we need to run commands on a running container we use:

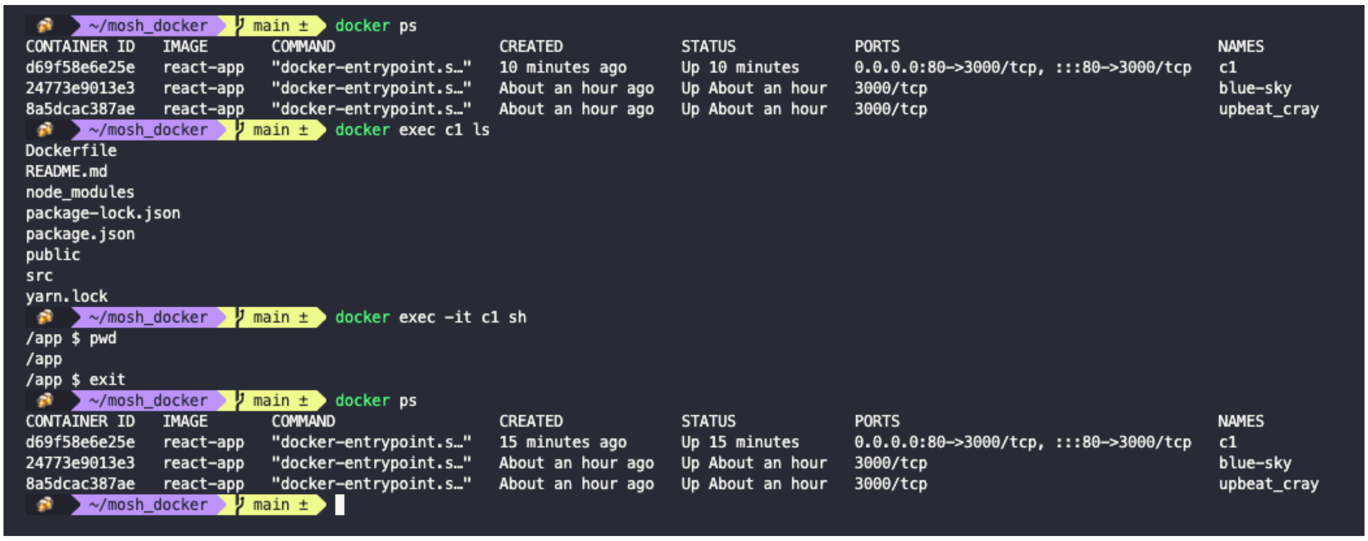
```
docker exec CONTAINER_NAME_OR_ID <command>
```

Commands need to be of the same OS as the base image (Linux commands for when using a Linux distro, Windows for when using Windows, and so on). This commands will be run in the default directory, which we specified in the `Dockerfile` with the `WORKDIR` instruction.

If we want to start a shell session in the container we run:

```
docker exec -it CONTAINER_NAME_OR_ID sh
```

When we are done we can run the `exit` command, and that won't affect the state of the container.

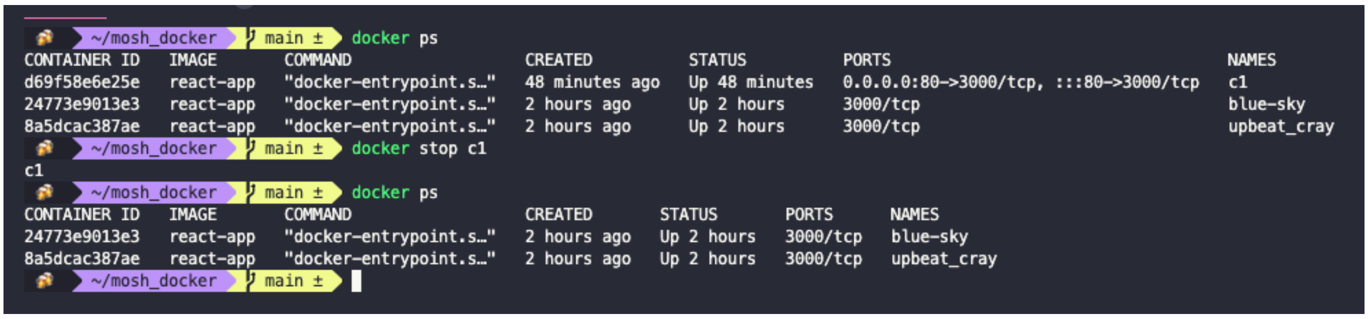


```
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED            STATUS           PORTS                                   NAMES
d69f58e6e25e   react-app   "docker-entrypoint.s…"   10 minutes ago     Up 10 minutes    0.0.0.0:80->3000/tcp, :::80->3000/tcp   c1
24773e9013e3   react-app   "docker-entrypoint.s…"   About an hour ago   Up About an hour   3000/tcp                              blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   About an hour ago   Up About an hour   3000/tcp                              upbeat_cray
 ~/mosh_docker  main ±  docker exec c1 ls
Dockerfile
README.md
node_modules
package-lock.json
package.json
public
src
yarn.lock
 ~/mosh_docker  main ±  docker exec -it c1 sh
/app $ pwd
/app
/app $ exit
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED            STATUS           PORTS                                   NAMES
d69f58e6e25e   react-app   "docker-entrypoint.s…"   15 minutes ago     Up 15 minutes    0.0.0.0:80->3000/tcp, :::80->3000/tcp   c1
24773e9013e3   react-app   "docker-entrypoint.s…"   About an hour ago   Up About an hour   3000/tcp                              blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   About an hour ago   Up About an hour   3000/tcp                              upbeat_cray
 ~/mosh_docker  main ± 
```

Stopping and starting containers

We can stop a running container with

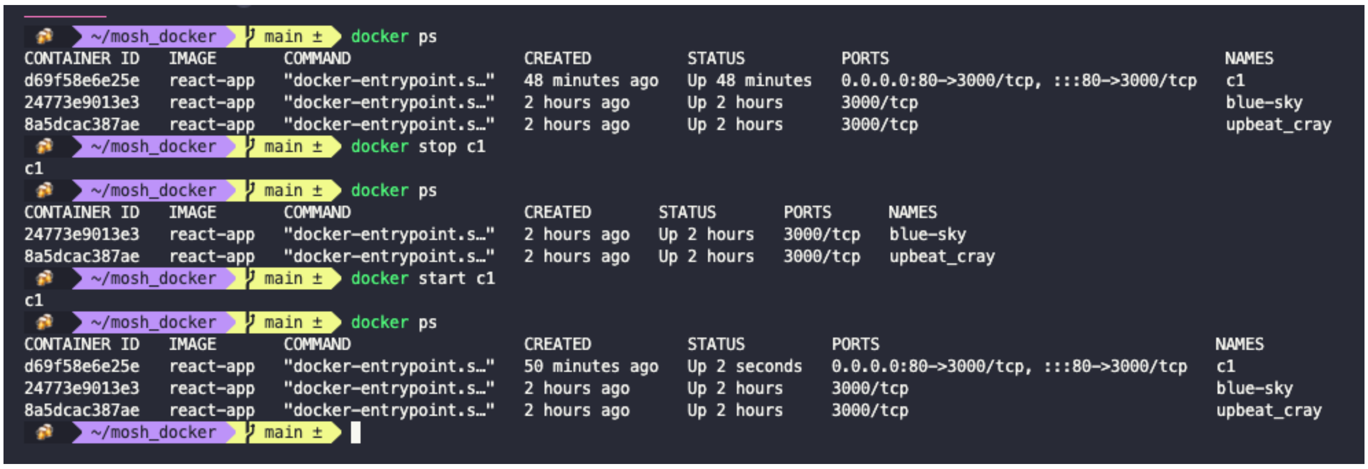
```
docker stop CONTAINER_NAME_OR_ID
```



```
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                   NAMES
d69f58e6e25e   react-app   "docker-entrypoint.s…"   48 minutes ago   Up 48 minutes   0.0.0.0:80->3000/tcp, :::80->3000/tcp   c1
24773e9013e3   react-app   "docker-entrypoint.s…"   2 hours ago      Up 2 hours      3000/tcp                                blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 hours ago      Up 2 hours      3000/tcp                                upbeat_cray
 ~/mosh_docker  main ±  docker stop c1
c1
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED       STATUS       PORTS      NAMES
24773e9013e3   react-app   "docker-entrypoint.s…"   2 hours ago   Up 2 hours   3000/tcp   blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 hours ago   Up 2 hours   3000/tcp   upbeat_cray
 ~/mosh_docker  main ± 
```

To re start it we use

```
docker start CONTAINER_NAME_OR_ID
```



```
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS          PORTS                                   NAMES
d69f58e6e25e   react-app   "docker-entrypoint.s…"   48 minutes ago   Up 48 minutes   0.0.0.0:80->3000/tcp, :::80->3000/tcp   c1
24773e9013e3   react-app   "docker-entrypoint.s…"   2 hours ago      Up 2 hours      3000/tcp                                blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 hours ago      Up 2 hours      3000/tcp                                upbeat_cray
 ~/mosh_docker  main ±  docker stop c1
c1
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED       STATUS       PORTS      NAMES
24773e9013e3   react-app   "docker-entrypoint.s…"   2 hours ago   Up 2 hours   3000/tcp   blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 hours ago   Up 2 hours   3000/tcp   upbeat_cray
 ~/mosh_docker  main ±  docker start c1
c1
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS         PORTS                                   NAMES
d69f58e6e25e   react-app   "docker-entrypoint.s…"   50 minutes ago   Up 2 seconds   0.0.0.0:80->3000/tcp, :::80->3000/tcp   c1
24773e9013e3   react-app   "docker-entrypoint.s…"   2 hours ago      Up 2 hours     3000/tcp                                blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 hours ago      Up 2 hours     3000/tcp                                upbeat_cray
 ~/mosh_docker  main ± 
```

Removing containers

There are two ways of removing a container:

```
docker container remove CONTAINER_NAME_OR_ID
```

or

```
docker rm CONTAINER_NAME_OR_ID
```

If the container is still running, we have two options. Stop the container and then remove it, or to use the `--force` option:

```
docker rm -f CONTAINER_NAME_OR_ID
```

If we have many container but only want to search for one, we can use the pipe (in Linux):

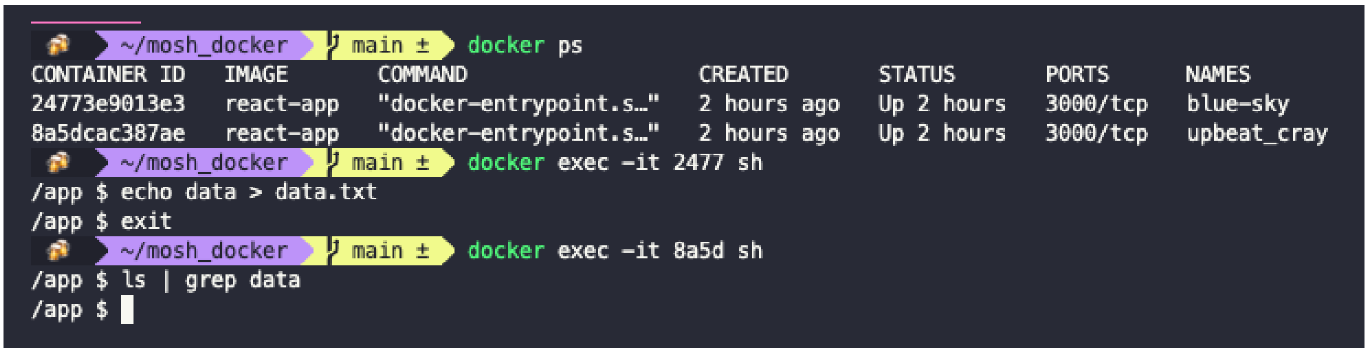
```
docker ps -a | grep CONTAINER_NAME
```

If we want to get rid of all stopped containers in one go, we run:

```
docker container prune
```

Containers file system

Each container has its own file system, which is invisible to other containers.



```
 ~/mosh_docker  main ±  docker ps
CONTAINER ID   IMAGE       COMMAND                  CREATED       STATUS      PORTS      NAMES
24773e9013e3   react-app   "docker-entrypoint.s…"   2 hours ago   Up 2 hours  3000/tcp   blue-sky
8a5dcac387ae   react-app   "docker-entrypoint.s…"   2 hours ago   Up 2 hours  3000/tcp   upbeat_cray
 ~/mosh_docker  main ±  docker exec -it 2477 sh
/app $ echo data > data.txt
/app $ exit
 ~/mosh_docker  main ±  docker exec -it 8a5d sh
/app $ ls | grep data
/app $
```

If we delete the container, the file system will be deleted with it and we will loose any extra data (for example, data generated during an interactive shell session). So we should never store data in the containers file system. To do that we use volumes.

Persisting data using volumes

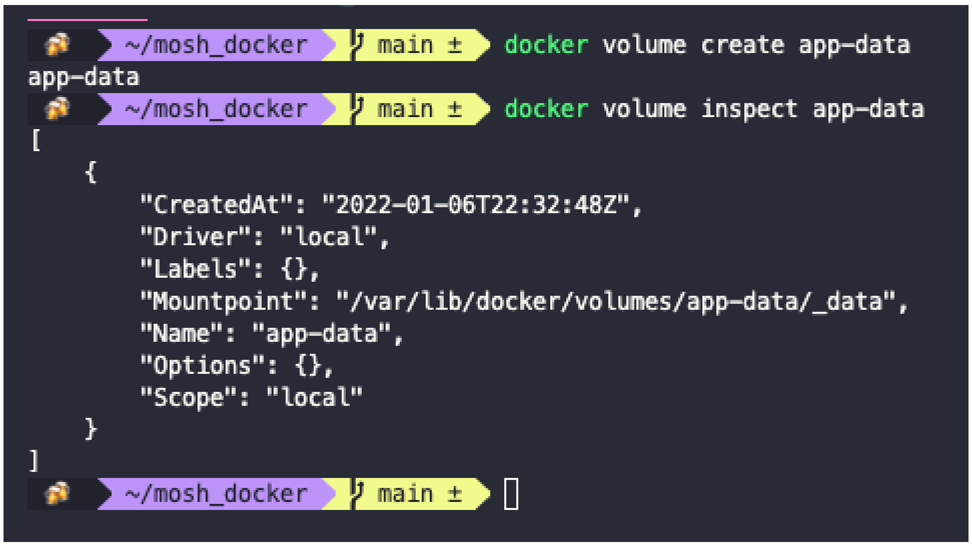
A volume is a storage outside of the container. It can be in the local host or in the cloud.

To create a volume we use the `create` command:

```
docker volume create VOLUME_NAME
```

To inspect the volume we use the `inspect` command:

```
docker volume inspect VOLUME_NAME
```



```
~/mosh_docker  main ±  docker volume create app-data
app-data
~/mosh_docker  main ±  docker volume inspect app-data
[
    {
        "CreatedAt": "2022-01-06T22:32:48Z",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/app-data/_data",
        "Name": "app-data",
        "Options": {},
        "Scope": "local"
    }
]
~/mosh_docker  main ± 
```

This example volume has `"Driver": "local"` because we are using local storage. If you want to create them in the cloud, you need to do some research in that cloud service for *how to create Docker volumes*. The `Mountpoint` key contains as its value the location of the volume. In the case of MacOS, the path is inside the virtual machine.

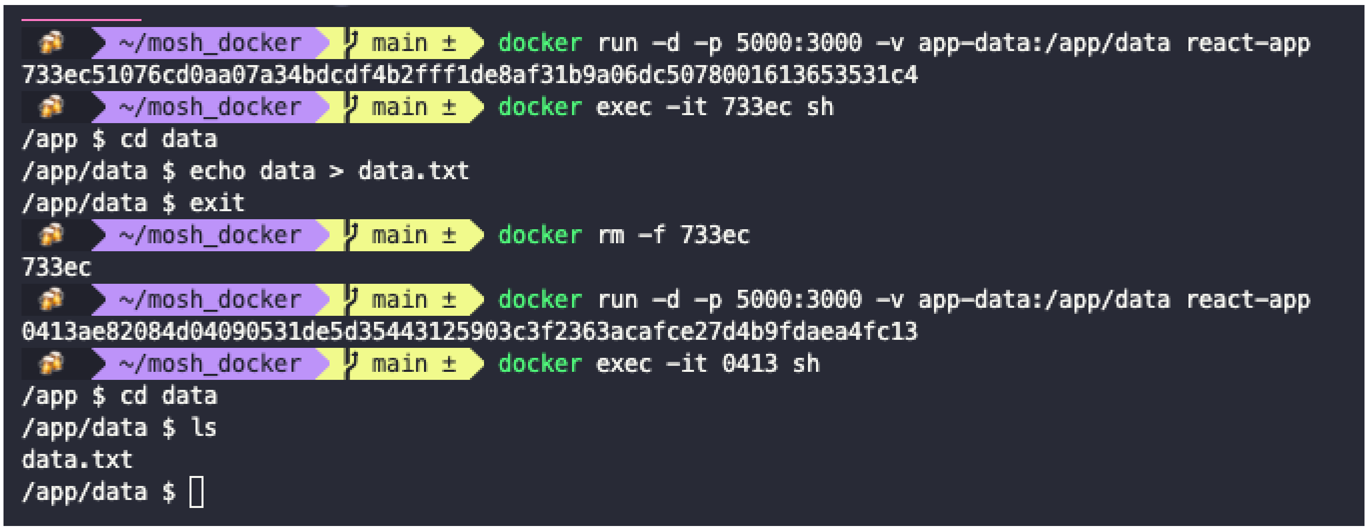
Now that we have a volume we can start a container and give it this volume for persistent data.

```
docker run -d -p LOCAL_PORT:CONTAINER_PORT -v VOLUME_NAME:/CONTAINER/ABSOLUTE/PATH
IMAGE
```

We do not need to create the volume before running the container. If the volume doesn't exist, Docker will create it when we run the `docker run` command. The same holds true for the directory in the container. But if the directory does not exist, we might run into issues with permissions (since the directory will be created by the `root` user). To solve this, we are going to add the creation of the directory in the `Dockerfile`.

```
FROM node:14.16.0-alpine3.13
RUN addgroup app && adduser -S -G app app
USER app
WORKDIR /app
RUN mkdir data
COPY package*.json .
RUN npm install
COPY . .
ENV API_URL=http://api.myapp.com/
EXPOSE 3000
CMD npm start
```

Now we can start a container, create a file in it, remove the container, start another container, and the file will still be there. This is because the volume is stored on the local host and not inside the container.



```
~/mosh_docker  main ±  docker run -d -p 5000:3000 -v app-data:/app/data react-app
733ec51076cd0aa07a34bdcdf4b2fff1de8af31b9a06dc5078001613653531c4
~/mosh_docker  main ±  docker exec -it 733ec sh
/app $ cd data
/app/data $ echo data > data.txt
/app/data $ exit
~/mosh_docker  main ±  docker rm -f 733ec
733ec
~/mosh_docker  main ±  docker run -d -p 5000:3000 -v app-data:/app/data react-app
0413ae82084d04090531de5d35443125903c3f2363acafce27d4b9fdaea4fc13
~/mosh_docker  main ±  docker exec -it 0413 sh
/app $ cd data
/app/data $ ls
data.txt
/app/data $
```

We can also share the same volume between different containers.

Copying files between the host and containers

Sometimes we need to copy files between the host and a container. To do this we use the copy (`cp`) command. To it we need to pass a *source* and a *destination*. For the source we use the container ID, colon, the path to the file inside the container. The destination is a directory on the host, but a `.` signifies the current directory.

```
docker cp CONTAINER_ID:PATH/TO/FILE .
```

If we want to copy from the host to the container we use the same command but invert the source and destination:

```
docker cp path/to/local/file CONTAINER_ID:PATH/TO/DIRECTORY
```

Sharing source code with a container

Some times we are running our project locally in a container while doing some work. What happens if we change a file and we want to see the changes in action (for example, changes to a web page that we are building and we want to see them on the browser). We don't want to have to rebuild the entire image every time we make a change while working locally.

To solve this we can create a mapping between a directory on the host and a directory inside the container. This way, any changes we make locally will immediately be visible inside the container. To do so, we use the same syntax as the one we used to map volumes. We can still add volumes by adding more `-v` flags.

```
docker run -d -p HOST_PORT:CONTAINER_PORT -v $(pwd):/CONTAINER/ABSOLUTE/PATH -v
VOLUME_NAME:/CONTAINER/ABSOLUTE/PATH IMAGE
```

Here the syntax `$(pwd)` means the return value of the `pwd` (current directory) command. This command will be evaluated first, and its return value replaced in our command.