Docker Cheat Sheet

This document will cover a small set of Docker CLI commands that you can use, and that may be necessary for the tasks that will be presented.

Docker Run

This is a command that we can use to instantiate a running instance of an image. This essentially starts a "program". There are quite a few flags and options that we can set to change it's behavior.

Detached Images

```
$ docker run -d <container_image>
```

> We use the **-d** flag to put the process that is created in the background. We often do not need or want to see the output generated by the container

Interactive Container

```
$ docker run -it <container_image>
```

> In the case we **do want** to see and interact with the container we need to include the **-it** flag(s). This allows us to interact with the "terminal" of the container

Port Mapping

```
$ docker run -d -p <host_port>:<container_port> <container_image>

# Example
# This will direct all traffic to the Host system's port 80 to the container's
port 80
# Additionally the container is detached this is not necessary
$ docker run -d -p 80:80 nginx
```

> We can use to **-p <host_port>:<container_port>** to direct all traffic to the host_port on the **host system** to the container_port on the container

Docker Volumes

```
$ docker run -d -p <host_port>:<container_port> -v <path_on_host>:
<path_in_container> <container_image>
```

```
# Example
$ docker run -d -p 80:80 -v ./proxy.conf:/etc/nginx/sites-available/default nginx
```

> We can use volumes to add persistance to containers. There are multiple types, but for this you should only need the **bind** mounts. For these mounts we specify a file or directory on the host and mount it (give access) to the container.

Docker Network Attaching

```
$ docker run -d -p <host_port>:<container_port> -v <path_on_host>:
<path_in_container> --network <network_name> <container_image>

# Example
$ docker run -d -p 80:80 -v ./proxy.conf:/etc/nginx/sites-available/default --
network web_proxy nginx
```

> This flag allows us to change the network a container will attach itself to, the default is the bridge network.

Docker Hostname

```
$ docker run -d -p <host_port>:<container_port> -v <path_on_host>:
<path_in_container> --network <network_name>  --hostname <hostname>
<container_image>

# Example
$ docker run -d -p 80:80 -v ./proxy.conf:/etc/nginx/sites-available/default --
network web_proxy --hostname proxy_test nginx
```

> The hostname flag allows us to change the hostname of the container, it defaults to the container ID

Docker Name

```
$ docker run -d -p <host_port>:<container_port> -v <path_on_host>:
<path_in_container> --network <network_name>  --hostname <hostname> --name <name>
<container_image>

# Example
$ docker run -d -p 80:80 -v ./proxy.conf:/etc/nginx/sites-available/default --
network web_proxy --hostname proxy_test --name proxy nginx
```

> The name is important for two reasons, it allows us to more easily interact with the container, and it is used to determine the internal DNS resolution for docker. We can internally refer to the example as "proxy" from another container.

Docker Network Management

Docker Network Create

```
$ docker network create <network_name>
```

> This is a command we can use to create docker networks. There are additional flags that are not covered to change the driver type, and other configurations. These other configurations can make the network internal, or attachable to non-swarm containers in the case of an overlay network.

Information Gathering

Docker Logs

```
$ docker logs <container_name/ID>

# Example
$ docker logs proxy
```

> This command will output logs (Information printed to stdout and stderr) generated by the container. This is useful for finding out why a container will not start as the internal programs will often print error messages.

Docker Inspect

```
# This container_id can also be some other object's ID
$ docker inspect <container_name/ID>

# Example
$ docker inspect proxy
```

> This allows us to see more information about the container or object we specify

Docker Exec

```
$ docker exec <container_name/ID> <command>

# Example
$ docker exec -it proxy /bin/bash
```

> This allows us to execute a command on a container. We can use this as described in the example to gain shell access to the container. This can be useful when we want to see if a file has been copied correctly or if the correct files have been edited or changed.

Docker Processes

```
$ docker ps -a
```

> This command will output all containers on the system, we can print only containers that are running by omitting the **-a**.

Container Management

Docker Start/Stop

```
$ docker <start/stop> <container_name>
```

> Start or stop **Only use one of the start or stop words** a container

Docker Remove

```
$ docker rm -f <container_name>

$ docker image rm -f <image_name>
```

> This will forcefully remove a container (If you want it to not remove running containers omit the **-f**)

Docker Pull

```
$ docker pull <image_name>
```

> This will download the container image onto the system

Additional References

- Official Docker: https://docs.docker.com/get-started/docker_cheatsheet.pdf
- Lots of Info: https://dockerlabs.collabnix.com/docker/cheatsheet/
- Portainer: https://docs.portainer.io/start/intro