Docker - Cheatsheet

Main

Docker Cheatsheet

Installation

Ubuntu

```
install_docker(){
    sudo apt-get update
    sudo apt-get install -y \
        ca-certificates \
        curl \
        gnupg \
        lsb-release
    # Add docker's official GPG key
    sudo mkdir -p /etc/apt/keyrings
    curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/etc/apt/keyrings/docker.gpg
    echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) \
    stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
    sudo apt-get update
    sudo apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
}

set_docker_user(){
    sudo usermod -aG docker ubuntu
}

install_docker
set_docker_user
```

Resource: https://docs.docker.com/engine/install/ubuntu/

Docker run params

`-d`: Run as a daemonized process
`--rm`: When a container stops, it's deleted
`-i`: Use STDIN
`-t`: Allocates sudo terminal to interact with the terminal
`--add-host 'hostname:ip address'`: Add hostname and ip address to `/etc/hosts` in running container

Resources: https://stackoverflow.com/questions/38302867/how-to-update-etc-hosts-file-in-docker-image-during-docker-build

Restart stopped docker container

```
docker start -ai <container name>
```

Push an image to the Docker repo

```
docker push <your name>/<image name>
```

Pull an image from the Docker repo

```
docker pull <name of image>
```

Build docker image

Run this command from the same directory as your `Dockerfile`.

```
NAME=myname
IMAGE_NAME=ubuntu-vnc
docker build . -t $NAME/$IMAGE_NAME
```

Rebuild all cached layers (useful when debugging):

```
docker build . -t $NAME/$IMAGE_NAME --no-cache
```

Resource: https://stackoverflow.com/questions/35594987/how-to-force-docker-for-a-clean-build-of-an-image

---

Debugging

If you want to start debugging a Dockerfile you're working on, do the following:

1. Add this to the bottom of your Dockerfile:

```
ENTRYPOINT tail -F /etc/passwd
```

2. Build and run the file:

```
docker build --squash -t <your name>/<image name> && docker run -d --name=<container name> <your name>/<image name> watch "echo 'test' >> /var/log/test.log"
```

3. Attach to it:

```
docker exec -it <name of running container> bash ##/bin/sh for alpine
```

Makefile template

```
build:
 docker build . -t <your name>/<image name>

run:
 docker run -d --name=<container name> --rm -it <your name>/<image name> watch "echo 'test' >> /var/log/test.log"

destroy:
 docker stop <container name>
```

Resource: https://medium.com/@betz.mark/ten-tips-for-debugging-docker-containers-cde4da841a1d

Export container to tar

```
docker save <container name or id> > <name of tar>.tar
```

Import docker tar

```
docker load < <name of tar>.tar
```

Stop & Remove docker containers

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

Remove docker image

```
docker rmi <image name or id>
```

Get shell on running docker container

```
docker exec -it <container name or id> /bin/bash
```

Run multiple commands

```
docker run -d --name=<container name> <your name>/<image name> /bin/bash -c "cd
/path/to/somewhere; python a.py"
```

Essentially, use `/bin/bash -c "commands"` and separate each command with a semicolon.

Resource: https://stackoverflow.com/questions/28490874/docker-run-image-multiple-commands

Start container and get shell (will stop container if you stop or exit terminal)

```
docker start -ai <container name or id>
```

Start container

```
docker start <container name or id>
```

List docker volumes

```
docker volume ls
```

Get mount point of a docker volume

```
docker volume inspect --format '{{ .Mountpoint }}' <volume name>
```

Delete unused volumes

```
docker volume prune
```

Copy file from running container to host

```
docker cp <container name>:<path to file> <location on host machine to copy file to>
```

Copy file form host to running container

```
docker cp <file> <container name>:<where you want file to go on container>
```

Pass proxy settings as params to docker run

```
-e http_proxy='<proxy settings>' \
-e https_proxy="<your proxy settings>" \
-e no_proxy="<your proxy settings>" \
-e HTTP_PROXY="<your proxy settings>" \
-e HTTPS_PROXY="<your proxy settings>" \
-e NO_PROXY="<your proxy settings>"
```

Resource: https://stackoverflow.com/questions/30494050/how-do-i-pass-environment-variables-to-docker-containers

Use proxy settings with a Docker Image

If you need proxy action in your Docker containers, add the following to your Dockerfile:

```
ENV http_proxy="<your proxy settings>" \
    https_proxy="<your proxy settings>" \
    no_proxy="<your proxy settings>" \
    HTTP_PROXY="<your proxy settings>" \
    HTTPS_PROXY="<your proxy settings>" \
    NO_PROXY="<your proxy settings>"
```

Change directory in a docker image

```
WORKDIR /path/to/cd/to
```

Mount directory using relative path

For this example, let’s imagine we have a container which takes an argument, and the output from running the container should go into a shared folder:

```
docker run -v $(PWD)/<folder to share w/ container>:/<location of folder to share in container>/<folder to share> <your name>/<image name> <argument container takes>
```

Get bash shell to container as a specific user

```
docker exec -it -u <user> <container name> /bin/bash
```

For example:

```
docker exec -it -u root jovial_mclean /bin/bash
```

Docker shortcuts in zsh

Under your ~/.zshrc, look for this field:

```
plugins=(<whatever>)
```

Once you’ve located it, change it to look something like this:

```
plugins=(git docker docker-compose nmap)
```

Welcome to the world of tab completion for your docker commands. You’ve leveled up. Bonus points for nmap and git shortcuts, which (of course) are completely unrelated to docker.

Install Docker on Kali

https://www.ptrace-security.com/2017/06/14/how-to-install-docker-on-kali-linux-2017-1/

Use entrypoint script

Create a file, docker-entrypoint.sh. Make sure it has the following in it:

```
#!/bin/bash
set -e

commands to run

exec "$@"
```

Be sure to run `chmod +x docker-entrypoint.sh` on the host system.

At the bottom of the Dockerfile:

```
COPY docker-entrypoint.sh /

ENTRYPOINT ["/docker-entrypoint.sh"]
```

Resource: https://success.docker.com/article/use-a-script-to-initialize-stateful-container-data

---

Docker Compose

Installation on Ubuntu

```
install_compose() {
    COMPOSE_VERSION=$(git ls-remote https://github.com/docker/compose | \
        grep refs/tags | \
        grep -oE "v[0-9]\.[0-9]\.[0-9]$" | sort --version-sort | tail -n 1)
    sudo curl -L \
        "https://github.com/docker/compose/releases/download/${COMPOSE_VERSION}/docker-
compose-$(uname -s)-$(uname -m)" \
        -o /usr/local/bin/docker-compose
    sudo chmod +x /usr/local/bin/docker-compose
```

```
    # Add command completion
    sudo curl -L \

"https://raw.githubusercontent.com/docker/compose/${COMPOSE_VERSION}/contrib/completion/bash/d
ocker-compose" \
        -o /etc/bash_completion.d/docker-compose
}

install_compose

# To test:
docker-compose --version
```

Resources: https://gist.github.com/wdullaer/f1af16bd7e970389bad3

https://docs.docker.com/compose/install/

Run bash script in container

Add this line at end the of the bash script:

```
exec bash
```

Resource: https://github.com/docker-library/wordpress/issues/205

Remove named containers

```
docker-compose down -v
```

Resource: https://stackoverflow.com/questions/45511956/remove-a-named-volume-with-docker-compose

Exec into container

```
docker-compose exec <container name> bash
```

Debug container

Add this to the `docker-compose.yml` file to keep a container running that exits due to an error:

```
command: tail -F anything
```

Port mapping to localhost

```
ports:
  - "127.0.0.1:80:80"
```

Watch logs

This is the equivalent of running `tail -f` on all of the containers in compose. Timestamps are included in the output through including the `-t` parameter.

```
docker-compose logs -t --tail="all" -f
```

Watch logs for specific service

You can get the service name from the `docker-compose.yml` file.

```
docker-compose logs -t --tail="all" -f $SERVICE_NAME
```

Resource: https://stackoverflow.com/questions/37195222/how-to-view-log-output-using-docker-compose-run

Static IP addresses

Add this to the bottom of the `docker-compose.yml` file:

```
networks:
  testing_net:
    ipam:
      driver: default
      config:
        - subnet: 172.18.0.0/24
```

For each container, add this line and change the `ipv4_address`:

```
networks:
  testing_net:
    ipv4_address: 172.18.0.4
```

Resource: https://blog.alejandrocelaya.com/2017/04/21/set-specific-ip-addresses-to-docker-containers-created-with-docker-compose/

Force recreate deployment

```
docker-compose up -d --force-recreate --build
```

Clean up containers daily

macOS

```
sudo touch /Library/LaunchDaemons/DockerSystemPrune.plist
```

Add this content:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
          http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>DockerSystemPrue</string>
    <key>Program</key>
    <string>/usr/local/bin/docker system prune -f</string>
    <key>StartCalendarInterval</key>
    <dict>
        <key>Hour</key>
        <integer>3</integer>
        <key>Minute</key>
        <integer>0</integer>
    </dict>
</dict>
```

```
</plist>
```

Run this:

```
launchctl load -w /Library/LaunchDaemons/DockerSystemPrune.plist
```

Linux

Create a cronjob:

```
0 3 * * * /usr/bin/docker system prune -f
```

Resource: https://nickjanetakis.com/blog/docker-tip-32-automatically-clean-up-after-docker-daily

---

Assign static ips to docker containers

```
docker network create --subnet=172.18.0.0/16 mynet123
docker run --net mynet123 --ip 172.18.0.22 -it ubuntu bash
```

Resource: https://stackoverflow.com/questions/27937185/assign-static-ip-to-docker-container

Change password for container user

```
RUN echo "root:Docker!" | chpasswd
```

Resource: https://stackoverflow.com/questions/28721699/root-password-inside-a-docker-container

Connect to remote container via Docker Remote API

This will run `docker ps` on the api running on target.com:

```
docker -H tcp://target.com:2376 ps
```

To get a shell to a container:

```
docker -H tcp://target.com:2376 exec -it container_name bash
```

Resource: https://stackoverflow.com/questions/18038985/how-to-connect-to-docker-api-from-another-machine

---

API

List all containers

```
curl -i -s -X GET http://target.com:2376/containers/json
```

Prepare command to run

```
curl -i -s -X POST \
     -H "Content-Type: application/json" \
     --data-binary '{"AttachStdin": true,"AttachStdout": true,"AttachStderr": true,"Cmd":
["whoami"],"DetachKeys": "ctrl-p,ctrl-q","Privileged": true,"Tty": true}' \

http://target.com:2376/containers/450ba994c49ceff62f50f63vd50fbeb20903ca93dbbd165b771a12e3b1d0
4f90/exec
```

Take the id that is output and set it in the command line:

```
id=6d521160423abf9c35b8ll62d3a1dffb0cb9cd577b9cdde20a49d553f871257c
```

Run the command

```
curl -i -s -X POST \
     -H 'Content-Type: application/json' \
     --data-binary '{"Detach": false,"Tty": false}' \
     http://target.com:2376/exec/$id/start
```

Resource: https://dejandayoff.com/the-danger-of-exposing-docker.sock/

https://docs.docker.com/engine/api/v1.37/

Get Docker container's IP address

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
container_name_or_id
```

Resource: https://stackoverflow.com/questions/17157721/how-to-get-a-docker-containers-ip-address-from-the-host

Get specific image from running container

1. Use docker inspect on the specific image id, i.e.

```
docker inspect 7f60f7bfc58a
```

2. Pull the repo digest sha from the output and use that to do a docker pull:

```
docker pull
chef/chefdk@sha256:d65c2597802d4a7336cd43cb3ed706701fcc9bdde122e53eacdead8b21a13591
```

Attach running container to existing docker network

```
docker network connect your-network-name container-name
```

Resource: https://stackoverflow.com/questions/50721424/docker-how-to-add-containers-to-same-network/50721476

Detect and cleanup running container

```
NAME=container_name
if [ ! "$(docker ps -q -f name=$NAME)" ]; then
    if [ "$(docker ps -aq -f status=exited -f name=$NAME)" ]; then
        # cleanup
        docker rm $NAME
    fi
fi
```

Resource: https://stackoverflow.com/questions/38576337/how-to-execute-a-bash-command-only-if-a-docker-container-with-a-given-name-does

Running commands in a container as part of a pipeline

Just use `-i` and skip the `-t`.

Resource: https://stackoverflow.com/questions/43099116/error-the-input-device-is-not-a-tty

Remove all images that match a name

In this particular example, web_apps:

```
docker images |grep web_apps | cut -d' ' -f1 | xargs docker rmi
```

If you’re seeing weird discrepancies

This has bit me in the ass so many times. If functionality is not working properly in code that you’re running in a container and you’re using Docker for Mac or Windows, go ahead and increase the size of the VM before spending hours debugging your code that is working perfectly fine. ave yourself a lot of time and frustration.

Multiple authors in a Dockerfile

```
LABEL authors="first author,second author"
```

Resource: https://stackoverflow.com/questions/38899977/how-do-i-declare-multiple-maintainers-in-my-dockerfile

Multiple commands on start in a Dockerfile

This is an example of how to run multiple commands on start for a container specified in the Dockerfile:

```
CMD service nginx start; service php7.2-fpm start; /usr/sbin/sshd -D
```

Multiple commands on start via ENTRYPOINT

1. Create `docker-entrypoint.sh` with the commands you want to run, for example:

```
set -e
service nginx start
service php7.2-fpm start
/bin/bash /opt/setup.sh &
/usr/sbin/sshd -D
```

2. Add the following to your `Dockerfile`:

```
COPY files/start.sh /start.sh
RUN chmod +x /start.sh
ENTRYPOINT ["/bin/bash", "/docker-entrypoint.sh"]
```

Resource: https://www.edureka.co/community/10736/how-to-run-multiple-commands-in-docker-at-once

Commit container

This command will get the container ID of a running container and create an image called `yourname/saved_container_image`:

```
docker commit $(docker ps -aqf "name=container_name") yourname/saved-container-image
```

Resource: https://stackoverflow.com/questions/34496882/get-docker-container-id-from-container-name

---

Run SSH as a specific user

Use the `Dockerfile` found here as the base for doing this.

Add the following to the following at the end of your `Dockerfile`:

```
USER youruser
WORKDIR /home/youruser

EXPOSE 22

CMD ["/usr/bin/sudo", "/usr/sbin/sshd", "-D", "-o", "ListenAddress=0.0.0.0"]
```

Please note that this user will need to be in the sudoers file for it to work properly.

Resources: https://stackoverflow.com/questions/22886470/start-sshd-automatically-with-docker-container https://dev.to/s1ntaxe770r/how-to-setup-ssh-within-a-docker-container-i5i https://github.com/jmal98/ansiblecm/blob/master/demo/machine/Dockerfile

---

Dangers

Do not under any circumstances expose `/var/run/docker.sock` to other containers. For `docker-compose`, you can looking for an entry like this:

```
volumes:
  - "/var/run/docker.sock:/var/run/docker.sock"
```

Resource: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html

Run command as user

```
docker exec -u <username> -it <container name> <command to run>
```

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

Kill all containers

```
docker kill $(docker ps -q)
```

Resource: https://www.codenotary.com/blog/extremely-useful-docker-commands/

Remove all containers

```
docker system prune
```

Docker logs

See live logs with the log file

1. Find the docker container log files location:

```
docker inspect --format='{{.LogPath}}' container_id
```

2. Run `tail -f` against that file

See live logs with docker cli

```
docker logs container_id --follow
```

Find logs since a particular date and time

```
docker logs container_id --since YYYY-MM-DDTHH:MM
```

Note that you can also just do the date by omitting the `:MM`.

Resources:

- Command to find container log
- https://www.decodingdevops.com/docker-logs-tail-docker-logs-to-file/

Execute binary on start

Add this to the end of your `Dockerfile`:

```
ENTRYPOINT ["./vuln"]
```

Resources:

- Provided answer
- Breakdown of CMD vs ENTRYPOINT
- More about CMD vs ENTRYPOINT

Dockerfile run multiple processes

```
CMD nohup ./service -port=8080 2>&1 & nohup ./servicedos 2>&1 & sleep infinity
```

Resource: https://stackoverflow.com/questions/52856059/how-to-launch-and-keep-a-background-process-inside-a-docker-container

---

GHCR

1. Create a classic Personal Access Token with the following permissions:

   delete:packages, repo, workflow, write:packages

and assign it to the `$GITHUB_TOKEN` env var.

2. Build the container from `Dockerfile` and tag it:

   IMG_NAME=cowdogmoo/delete-pods
   docker build . -t $IMG_NAME
   docker tag IMGNAME"ghcr.io/{IMG_NAME}:latest"
3. Login to the repository and push the container image:

   docker login ghcr.io -u CowDogMoo -p $GITHUB_TOKEN

   docker push ghcr.io/cowdogmoo/delete-pods:latest

Container Registry in Gitlab

Login to the repository:

```
docker login repository.gitlab.com:4567
```

Create container image:

```
docker build -t repository.gitlab.com:4567/project-name/container-name:version .
# To also have a latest tag:
docker build -t repository.gitlab.com:4567/project-name/container-name:latest .
```

Upload container image:

```
docker push repository.gitlab.com:4567/project-name/container-name:version
# To also have a latest tag:
docker push repository.gitlab.com:4567/project-name/container-name:latest
```

Logout of the repository:

```
docker logout repository.gitlab.com:4567
```

Detect if process running in container

```
if grep -sq 'docker\|lxc' /proc/1/cgroup; then
    echo "I am running on Docker."
fi
```

Resource: https://stackoverflow.com/questions/20010199/how-to-determine-if-a-process-runs-inside-lxc-docker

---

Get the name of a running container

```
docker run -dit -p 5901:5901 ubuntu && CONTAINER=$(docker ps | awk -F '  ' '{print $7}' |
xargs) && echo $CONTAINER
```

Original Article