CHEAT SHEETS Talan Labs

docker

Docker Command

General Usage

Start a container in Background
```
$ docker run -d jenkins
```
Start an interactive container
```
$ docker run -it ubuntu bash
```
Start a container and remove once stopped
```
$ docker run -rm jenkins
```
Expose a port from the container on the host
```
$ docker run -p 8000:4000 -d jenkins
```
Start a named container
```
$ docker run --name myDb -d postgres
```
Stop a running container
```
$ docker stop myDb
```
Start a stoped container
```
$ docker start myDb
```

Debug

Run a shell command in a running container
```
$ docker exec -it myNamedContainer sh
```
Follow logs of a running container
```
$ docker logs -f myRunningContainer
```
Show open port of container
```
$ docker port myRunningContainer
```

Building Images

Build an image from a Dockerfile in same dir
```
$ docker build -t myImage .
```
Force rebuild an image
```
$ docker build -t myImage --no-cache .
```
Create an image from a container
```
$ docker commit sha123123 myNewImage
```
Remove an image
```
$ docker rmi myNewImage
```

Container Management

List running container
```
$ docker ps
```
List all container
```
$ docker ps -a
```
Inspect container Metadata
```
$ docker inspect sha1231234
```
List local images
```
$ docker images
```
Kill all container
```
$ docker kill $(docker ps -q)
```
Remove all stopped container
```
$ docker rm $(docker ps -q -a)
```
Removing all untagged image
```
$ docker rmi $(docker images \
  | grep "^<none>" | awk '{print $3}')
```

Volumes

Mounting a local Directory on a container
```
$ docker run -V myFolder//data myContainer
```
Create a local volume
```
$ docker volume create --name myVolume
```
Mounting a volume on a container
```
$ docker run -V myVolume:/data myContainer
```
Destroy a volume
```
$ docker volume rm myVolume
```
List volumes
```
$ docker volume ls
```

Network

Create a local Network
```
$ docker network create myNetwork
```
Attach a container to a Network on startup
```
$ docker run --net myNetwork
```
Connect a running container to a network
```
$ docker network connect myNetwork myContainer
```
Disconnect a running container to a network
```
$ docker network disconnect myNetwork myContainer
```

docker-compose.yml

```
version: '3'

services: ①
  proxy: ②
    image: nginx:1.15.2 ③
    ports:  ④
      - "8080:8080"
    networks: ⑤
      - frontend
  web: ②
    env_file: env.env ⑥
    build: ⑦
      context: ./dir
      dockerfile: Dockerfile-alternate
      args:
        - MyARG=NicoAsArg
    ports: ④
      - "5000:5000"
    volumes: ⑧
      - .:/config
    depends_on: ⑨
      - postgresql
    networks:  ⑤
      - database
      - frontend
  postgresql: ②
    image: postgresql ③
    networks:  ⑤
      - database

networks: ⑤
  database:
  frontend:
```

① ***services*** : docker compose run services,

② ***services names*** : each services is referenced in docker-compose using its service name and not the docker sha or docker name

③ ***images*** : instruct docker-compose that the service will use a *raw* image for the service execution

④ ***ports*** : maps container port to host port

⑤ ***networks*** : segragates services between network for discovery and security. In this example, proxy will never have access to the postgres database. But can refer to web as a known hostname, and web can access postgresql with postgresql hostname.

⑥ ***env_file*** : set list of environment variable available in the container from a file on the host - only available during execution, not build.

⑦ ***build*** : instruct docker-compose to build the container from a Dockerfile. Dockerfile filename and path can be overiden as described

⑧ ***volumes*** : volumes from host can also be mounted in the container very usefull in developpement to have your apps changes available in the service without rebuilding the container

⑨ ***depends_on*** : wait for depended services to be started - doesn't mean it's ready, just that compose has started the depended service. watch the other side of the poster for more info on service dependencies

Dockerfile

```
LABEL maintainer="nicolas.savois@talan.com" ①

FROM debian:jessie ②

ENV nginxVer="XX.Y-Z" ③

RUN apt-get install open-ssl ④

RUN curl http://xx.org/.../nginx_${nginxVer}.deb -o nginx.deb -s && \ ⑤
    dpkg -i nginx.deb && \
    rm nginx.deb && \ ⑥
    ln -s /etc/nginx/sites-available/site /etc/nginx/sites-enabled/site

COPY nginx.conf /etc/nginx/nginx.conf ⑦

ADD myapp.conf /etc/nginx/sites-available/ ⑧

USER 1000:1000 ⑨

WORKDIR /path/to/workdir ⑩

ENTRYPOINT nginx start ⑪
```

① **LABEL** : Add a label to the metadata of the docker image

② **FROM** : The base image used to build the new image

③ **ENV** : Create and environment variable reusable later, check (5) for usage

④ **RUN** : Run a command to build the image like adding a package, touching file, etc...

⑤ **&& \** : Each line in the dockerfile create a new layer in the docker image. To avoid the layer multiplication we group commands with this shell feature

⑥ ***trick*** : Remove the downloaded file from the layer - no need to keep it once installed

⑦ **COPY** : Copy inside the image a file from the host (replace if it exists)

⑧ **ADD** : Copy inside the specified folder, - just use **COPY**, **ADD** comes with Magic around, and we all hate magic! (right?)

⑨ **USER** : Change user, goes back to the kernel and run the next commands as the user with UID:GID from the docker host (1000:1000 is the first user created on nearly all linux distribution)

⑩ **WORKDIR** : Change directory, (most expensive cd in the world)

⑪ **ENTRYPOINT** : Command run when the container start (PID=1)

Docker Compose Command

General Usage

build the container from docker-compose.yml
```
docker-compose build
```
specify non default compose file
```
docker-compose -f myConfig.yml run backup
```
specify a project name
```
docker-compose -p myproject run backup
```
used by compose to define container name with `docker ps`, defaults to the folder name

create an alias for docker-compose
```
alias dc='docker-compose'
```
will save you a lot of typing :)

Managing Composed Services

run the services in foreground
```
docker-compose up
```
run the services in background
```
docker-compose up -d
```
run only one service
```
docker-compose up web
```
stop & remove all services, volmes & network
```
docker-compose down
```
stop one service
```
docker-compose stop web
```
restart a stoped service
```
docker-compose start web
```
remove a container associated with service
```
docker-compose rm web
```
stop and remove everything
```
docker-compose rm -vfs web
```

Debuging Composed Services

Running Commands in started container
```
docker-compose exec web sh
```
Running commands in container
```
docker-compose exec web sh
```
follow logs of the containers
```
docker-compose logs -f --tail=10
```
tail only display 10 lines of history, useful when compose runs for a long time...

display running services
```
docker-compose ps
```
validate compose config and show compose file
```
docker-compose config
```

Best Practices

The Twelve Factors

α. ***Codebase***
One codebase tracked in revision control, many deploys

β. ***Dependencies***
Explicitly declare and isolate dependencies

γ. ***Config***
Store config in the environment

δ. ***Backing services***
Treat backing services as attached resources

ε. ***Build, release, run***
Strictly separate build and run stages

ζ. ***Processes***
Execute the app as one or more stateless processes

η. ***Port binding***
Export services via port binding

θ. ***Concurrency***
Scale out via the process model

ι. ***Disposability***
Maximize robustness with fast startup and graceful shutdown

κ. ***Dev/prod parity***
Keep development, staging, and production as similar as possible

λ. ***Logs***
Treat logs as event streams

μ. ***Admin processes***
Run admin/management tasks as one-off processes

Package a single application per container

Properly handle PID 1, signal handling, and zombie processes

Optimize for the Docker build cache

Remove unnecessary tools

Build the smallest image possible

Use vulnerability scanning in Container Registry

Properly tag your images

Carefully consider whether to use a public image

Docker-Compose Best Practices

Talanlabs publicity

Dockerfile Things

don't be too much stupid

Multistage Build