root@docker:~# docker exec <conteneurid> /bin/bash -c 'ls /var/www/html'

Stop / start / pause / unpause

root@docker:~# docker stop <conteneurid>

root@docker:~# docker start <conteneurid>

root@docker:~# docker pause <conteneurid>

root@docker:~# docker unpause <conteneurid>

docker build -f DockerFile

Sauvegarde une image dans un fichier tar

root@docker:~# docker save <imageid> -o /tmp/debian.tar

Charger une image depuis un tar

root@docker:~# docker load -i /tmp/debian.tar

Commiter un conteneur vers une image

root@docker:~# docker commit <conteneurid> <imagename>

# Updating and committing an image

docker commit -m "Added json gem" -a "Kate Smith" 0b2616b0e5a8 ouruser/sinatra:v2

Debug

Obtenir des informations sur une image / un conteneur

root@docker:~# docker inspect <conteneurid|imageid>

Afficher les ressources du conteneur

root@docker:~# docker stats <conteneurid>

Voir le STDOUT d'un conteneur

root@docker:~# docker logs <conteneurid>

-f : suivre en permanence les logs de conteneurs (correspond à tail -f)

-t : affiche date et l'heure de reception

# Show container logs

docker logs -f name\_container

Commande de gestion reseau

docker network create

docker network connect

docker network ls

docker network rm

docker network disconnect

docker network inspect

# Show networks

docker network ls

# Create Bridge network

docker network create -d bridge my-bridge-network

# Show network detail

docker network inspect my-bridge-network

docker inspect --format='{{json .NetworkSettings.Networks}}' name\_container

# Show IP address Container

docker inspect --format='{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' web

# Connect networks

docker network connect my-bridge-network web

# Attach Container to network

docker network connect my-bridge-network web

The following command disconnects the redis container from the frontend-network.

docker network disconnect frontend-network redis

# Adding a data volume

docker run -d -P --name web -v /webapp training/webapp python app.py

docker run -d --name=redis --net=backend-network redis

# Start Container as a daemon

docker run -d ubuntu /bin/sh -c "while true; do echo hello world; sleep 1; done"

# Set name container

docker run -d -P --name web training/webapp python app.py

# Mount a host directory as a data volume

docker run -d -P --name web -v /src/webapp:/webapp training/webapp python app.py

Build NGINX PROXY

docker run -d -p 80:80 -e DEFAULT\_HOST=proxy.example -v /var/run/docker.sock:/tmp/docker.sock:ro --name nginx jwilder/nginx-proxy

For Nginx-proxy to start sending requests to a container you need to specify the VIRTUAL\_HOST environment variable. This variable defines the domain where requests will come from and should be handled by the container.

In this scenario we'll set our HOST to match our DEFAULT\_HOST so it will accept all requests.

docker run -d -p 80 -e VIRTUAL\_HOST=proxy.example katacoda/docker-http-server

Launch a second container using the same command as we did before.

docker run -d -p 80 -e VIRTUAL\_HOST=proxy.example katacoda/docker-http-server

While nginx-proxy automatically creates and configures NGINX for us, if you're interested in what the final configuration looks like then you can output the complete config file with docker exec as shown below.

docker exec nginx cat /etc/nginx/conf.d/default.conf

Set name and value ofenvironnement variable

docker run -d --name my-production-running-app -e NODE\_ENV=production -p 3000:3000 my-nodejs-app

DOCKERFILE ONBUILD EXAMPLE

FROM node:7

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

ONBUILD COPY package.json /usr/src/app/

ONBUILD RUN npm install

ONBUILD COPY . /usr/src/app

CMD [ "npm", "start" ]

Building Dockerfile

docker build -t nopassword .

Ignoring file while Copy from host

echo passwords.txt >> .dockerignore

Create a Data Container for storing configuration files using

docker create -v /config --name dataContainer busybox

Using the --volumes-from <container> option we can use the mount volumes from other containers inside the container being launched. In this case, we'll launch an Ubuntu container which has reference to our Data Container. When we list the config directory, it will show the files from the attached container.

docker run --volumes-from dataContainer ubuntu ls /config

If a /config directory already existed then, the volumes-from would override and be the directory used. You can map multiple volumes to a container.

If we wanted to move the Data Container to another machine then we can export it to a .tar file.

docker export dataContainer > dataContainer.tar

docker import dataContainer.tar

will import the Data Container back into Docker.

The most common scenario for connecting to containers is an application connecting to a data-store. The key aspect when creating a link is the name of the container. All containers have names, but to make it easier when working with links, it's important to define a friendly name of the source container which you're connecting to.

Start Data Store

Run a redis server with a friendly name of redis-server which we'll connect to in the next step. This will be our source container.

docker run -d --name redis-server redis

To connect to a source container you use the --link <container-name|id>:<alias> option when launching a new container. The container name refers to the source container we defined in the previous step while the alias defines the friendly name of the host.

By setting an alias we separate how our application is configured to how the infrastructure is called. This means the application configuration doesn't need to change as it's connected to other environments.

How Links Work

In this example, we bring up a Alpine container which is linked to our redis-server. We've defined the alias as redis. When a link is created, Docker will do two things.

First, Docker will set some environment variables based on the linked to the container. These environment variables give you a way to reference information such as Ports and IP addresses via known names.

You can output all the environment variables with the env command. For example:

docker run --link redis-server:redis alpine env

Secondly, Docker will update the HOSTS file of the container with an entry for our source container with three names, the original, the alias and the hash-id. You can output the containers host entry using cat /etc/hosts

docker run --link redis-server:redis alpine cat /etc/hosts

This example will use Redis as a way to persist data. Start a Redis container below, and create a data volume using the -v parameter. This specifies that any data saved inside the container to the /data directory should be persisted on the host in the directory /docker/redis-data.

docker run -v /docker/redis-data:/data \

--name r1 -d redis \

redis-server --appendonly yes

We can pipe data into the Redis instance using the following command.

cat data | docker exec -i r1 redis-cli --pipe

Redis will save this data to disk. On the host we can investigate the mapped direct which should contain the Redis data file.

ls /docker/redis-data

This same directory can be mounted to a second container. One usage is to have a Docker Container performing backup operations on your data.

docker run -v /docker/redis-data:/backup ubuntu ls /backup

Mounting Volumes gives the container full read and write access to the directory. You can specify read-only permissions on the directory by adding the permissions :ro to the mount.

If the container attempts to modify data within the directory it will error.docker run -v /docker/redis-data:/data:ro -it ubuntu rm -rf /data

The command below will redirect the redis logs to syslog.

docker run -d --name redis-syslog --log-driver=syslog redis

set the log-driver to none. No output will be logged.

docker run -d --name redis-none --log-driver=none redis

command below will output the LogConfig section for each of the containers.

docker inspect --format '{{ .HostConfig.LogConfig }}' redis-server

You can launch an instance using

docker run -d --name restart-default scrapbook/docker-restart-example

The option --restart=on-failure:# allows you to say how many times Docker should try again. In the example below, Docker will restart the container three times before stopping.

docker run -d --name restart-3 --restart=on-failure:3 scrapbook/docker-restart-example

Use the always flag to automatically restart the container when is crashes for example docker run -d --name restart-always --restart=always scrapbook/docker-restart-example

docker logs restart-default

If you're adding multiple labels, then these can come from an external file. The file needs to have a label on each line, and then these will be attached to the running container.

This line creates two labels in the file, one for the user and the second assigning a role.

echo 'user=123461' >> labels && echo 'role=cache' >> labels

The --label-file=<filename> option will create a label for each line in the file.

docker run --label-file=labels -d redis

Using the -f option you can filter the JSON response to just the Labels section we're interested in.

docker inspect rd

docker inspect -f "{{json .Config.Labels }}" rd

docker ps --filter "label=user=scrapbook"

Inspecting images works in the same way however the JSON format is slightly different, naming it ContainerConfig instead of Config.

docker inspect -f "{{json .ContainerConfig.Labels }}" katacoda-label-example

docker images --filter "label=vendor=Katacoda"

STATISTIQUE

docker stats nginx

docker ps -q | xargs docker stats

MULTIUSAGE DOCKERFILE

Create the desired Docker Image using the build command below.

docker build -f Dockerfile.multi -t golang-app .

# First Stage

FROM golang:1.6-alpine

RUN mkdir /app

ADD . /app/

WORKDIR /app

RUN CGO\_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main .

# Second Stage

FROM alpine

EXPOSE 80

CMD ["/app"]

# Copy from first stage

COPY --from=0 /app/main /app

To limit which columns are displayed, use the \_--format\_\_ parameter. The parameter allows pretty-printing containers using a Go template syntax.

docker ps --format '{{.Names}} container is using {{.Image}} image'

As it's using Go templates, it includes helper functions such as table.

docker ps --format 'table {{.Names}}\t{{.Image}}'

The format parameter can then access all of the container information. Below is an example of listing all the IP addresses for the running containers.

docker ps -q | xargs docker inspect --format '{{ .Id }} - {{ .Name }} - {{ .NetworkSettings.IPAddress }}'

Docker have made available a script which will deploy the required components for the new Rootless version.

Run the following command as lowprivuser to execute the script and install the components.

curl -sSL https://get.docker.com/rootless | sh

Rootless Docker has now been installed. The daemon can be started using the following script:

export XDG\_RUNTIME\_DIR=/tmp/docker-1001

export PATH=/home/lowprivuser/bin:$PATH

export DOCKER\_HOST=unix:///tmp/docker-1001/docker.sock

/home/lowprivuser/bin/dockerd-rootless.sh --experimental --storage-driver vfs

This will run in the foreground and allow you to see the debug output from the Rootless Docker Daemon.

Click the following command to launch a second terminal window and change the context to run as roootlessuser.

sudo su lowprivuser

: "Second Terminal running as lowprivuser"

id

To access Docker, set the following environment variables. This specifies connecting to the Docker instance running for the user with id 1001, which should match the id of lowprivuser.

export XDG\_RUNTIME\_DIR=/tmp/docker-1001

export PATH=/home/lowprivuser/bin:$PATH

export DOCKER\_HOST=unix:///tmp/docker-1001/docker.sock

It's now possible to access the Docker Daemon running for user 1001.

The standard Docker CLI commands work in the same way. The following command lists all the containers running for the user, currently it should return an empty list.

docker ps

It's possible to inspect details of the Daemon running:

docker info

Containers can be in the same way.

docker run -it ubuntu bash

Users within the container will still be reported as root. They will be able to install packages and modify parts of the system running inside of the Docker. However, if they managed to break out they wouldn't be able to interfer with the host.

id

In a separate terminal window, as root it's possible to explore which processes are running and which user started them. Using ps aux you can verify that our new container instance is managed and owned by our low privileged user.

id; ps aux | grep lowprivuser

The system is now running Docker Containers without requiring any additionals permissions, allowing our systems to operatoe with increased security.

$ docker container update --cpu-shares 512 --memory 128M --memory-swap 256M nginx\_limited

Comment retarder le stop d’un container ?

Bien que bizarre, cette commande est utilise lorsque l’application dockerisée à un processus d’arret assez long (plus que 10s).

Elle évite un kill “tranchant” sur notre application et lui laisse le temps de finir sa procédure de fermeture.

Les bases de données sont souvent visées par ce comportement

$ docker container stop -t 60 cassandra

Comment lister la redirection des ports d’un container ?

# commande simple

$ docker container port cassandra

# commande plus complexe

$ docker inspect --format='{{range $p, $conf := .Config.ExposedPorts}} {{$p}} {{end}}'containerid

80/tcp-> 8000

EXAMPLE MAKEFILE

NAME = katacoda/docker-http-server

TAG=v1

INSTANCE = scrapbook-http-server

.PHONY: default build copy debug clean push buildrelease

default: buildrelease

build:

docker build -t $(NAME)-dev .

copy:

docker create --name $(INSTANCE) $(NAME)-dev

docker cp $(INSTANCE):/app/main $(shell pwd)/app

docker rm $(INSTANCE)

release:

docker build -t $(NAME):$(TAG) -f Dockerfile-release .

docker tag $(NAME):$(TAG) $(NAME):latest

buildrelease: build copy release

clean:

docker rm $(INSTANCE)

debug:

docker run --rm -it --name $(INSTANCE) $(NAME)-dev /bin/bash

run:

docker run --rm -p 80:80 --name $(INSTANCE) $(NAME)

dev:

docker run -it --rm -p 80:80 -w /go/src/github.com/$(NAME) -v $(shell pwd)/vendor/github.com/:/go/src/github.com/ -v $(shell pwd):/go/src/github.com/$(NAME) golang:1.6

push:

docker push $(NAME):$(TAG)

docker push $(NAME):latest