**Content**

[1. Docker community edition configuration 2](#_Toc79855363)

[1. Storage driver 2](#_Toc79855364)

[2. Docker run 2](#_Toc79855365)

[3. Docker other commands 3](#_Toc79855366)

[4. Logging drivers 3](#_Toc79855367)

[5. Docker Swarm 4](#_Toc79855368)

[1.5.1. Swarm commands 4](#_Toc79855369)

[1.5.2. Swarm Backup-Restore 5](#_Toc79855370)

[2. Docker images 7](#_Toc79855371)

[1. Building Efficient images 10](#_Toc79855372)

[3. Whizlabs learnings 11](#_Toc79855373)

[1. Replicated and global services 11](#_Toc79855374)

1. Docker community edition configuration
   1. Storage driver

Provide a pluggable framework for managing the temporary, internal storage of a container’s writable layer. The best storage driver depends on the environemnt and on the needs.

overlay2: File-based storage, default for Ubuntu and CentOS 8+

devicemapper: Blcik storage, more efficient for doing lots of writes.

Docker info shows the current dewice mapper in use.

Docker daemon

sudo vi /usr/lib/systemd/system/docker.service

# docker daemon configuration The preferred way. By default does not exists

sudo vi /etc/docker/daemon.json

# restarting docker after editing the configuration

sudo systemctl restart docker

sudo systemctl status docker

* 1. Docker run

docker run [OPTIONS] IMAGE[:TAG] [COMMAND] [ARG...]

-d , --detach

Run the container in background and print the container ID. The run command will exit.

-i , --interactive

Keep STIN open even if not attached

-a , --attach

Attach to STDIN, STDOUT or STDERR

--rm

Automatically remove container when it exits (when it stops running).

-t , --tty

Allocate a pseudo-TTY

-p , --publish

Publish a container port to the host

--name

A container is assigned to a random name by default, but u can assign new

--restart

When the container should be automatically restarted

[“no”, “on-failure”, “always”, “unless-stopped”]

* 1. Docker other commands

# list running containers

docker ps

# list all containers

docker ps -a

docker container stop nginx

docker container start nginx

# delete a container (it must be stopped first)

docker container rm nginx

* 1. Logging drivers

Logging drivers are pluggable framework for accessing log data from services and containers in Docker.

# log driver can be set in the docker deamon config

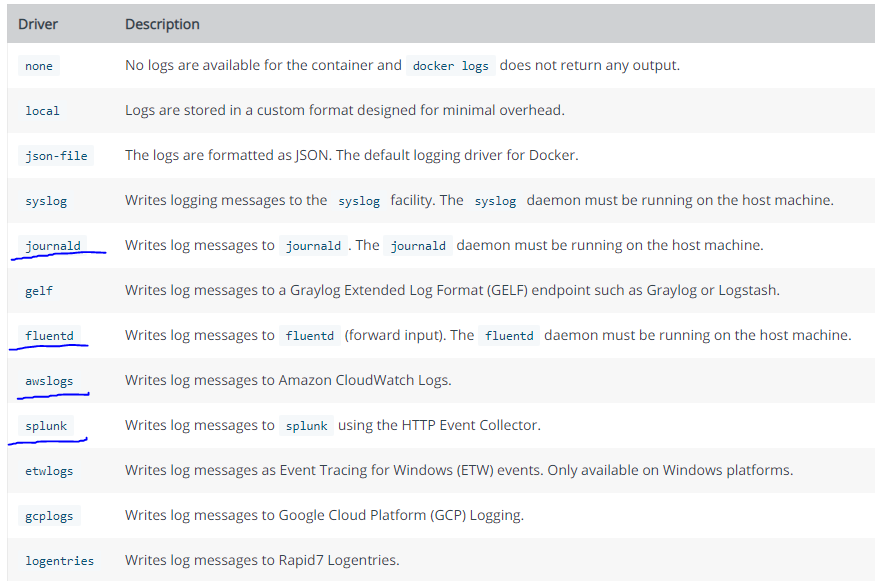
sudo vi /etc/docker/daemon.json

# check the current loggin driver

docker info | grep Logging

# to find where the json-file logging driver is logging

docker inspect --format='{{.LogPath}}' NAME|ID



**Journald** is a system service for collecting and storing log data, introduced with systemd. It tries to make it easier for system administrators to find interesting and relevant information among an ever-increasing amount of log messages.

**Fluentd** is a cloud native logging solution to unify data collection and consumption.

**json-file** -by default, Docker captures the standard output (and standard error) of all your containers, and writes them in files using the JSON format. The JSON format annotates each line with its origin (stdout or stderr) and its timestamp. Each log file contains information about only one container.

* 1. Docker Swarm

It allows to build a distributed cluster of docker machines to run the containers. Orchestration, high-availability, scaling.

The swarm manager is going to delegate and assign containers to the worker nodes and the worker nodes will run them.

* + 1. Swarm commands

docker swarm init --advertise-addr <swarm manager private IP>

docker info # basic info about the current status of the swarm

docker node ls # list the current nodes in the swarm and their statuses

# run this command on the manager, to get a join token

docker swarm join-token worker

# to actually join

docker swarm join --token <token> <swarm manager private IP>:2377

# backup on the manager

sudo systemctl stop docker

sudo tar -zvcf backup.tar.gz -C /var/lib/docker/swarm .

sudo systemctl start docker

# restore on the manager

sudo systemctl stop docker

sudo rm -rf /var/lib/docker/swarm/\*

sudo tar -zxvf backup.tar.gz -C /var/lib/docker/swarm/

sudo systemctl start docker

docker node ls

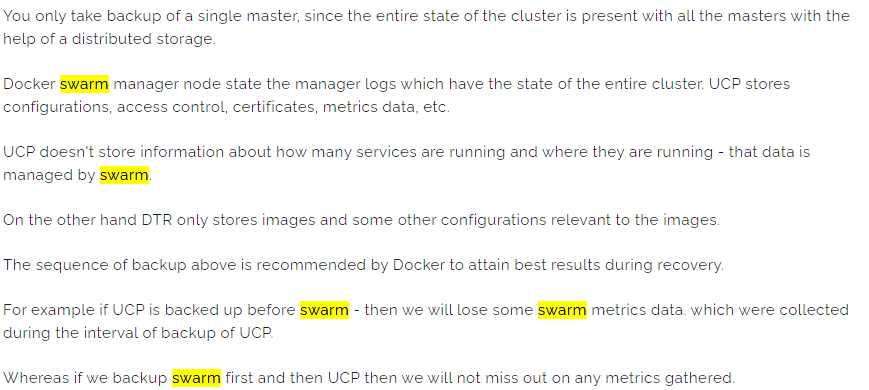
# to change the desired state of the service running in the swarm

docker service scale <SERVICE-ID>=<NUMBER-OF-TASKS>

# to see the updated task list

docker service ps <SERVICE-ID>

* + 1. Swarm Backup-Restore
* backing up data from 1 manager node is enough, since all managers store the same data
* swarm, uco, dtr



To backup:

* stop the docker service
* Back up all data in the directory /var/lib/docker/swarm
* Start the docker service

To restore

* stop the docker service
* delete any existing files or dictories uner /bar/lib/docker/swarm
* copy the backed-up files to /var/lib/docker/swarm
* Start the docker service

1. Docker images

A docker image is a file containing the code and components needed to run software in a container.

Containers and images use a layered file system. Each layer contains only the differences from the previous layer. The image consists of one or more read-only layers, while the conatiner adds one addtion writable layer. This layered system allows multiple images and containers to share the same layers.

docker image pull nginx

docker image history nginx

docker build -t custom-nginx .

docker run --name custom-nginx -d -p 8080:80 custom-nginx

curl localhost:8080

# locate running container

docker ps

docker container rm -f <container id>

docker image pull IMAGE[:TAG]

docker image ls

# add the –a flag to include intermediate images

# we did not download them directly, they are here because another images depend on it. like when you create image FROM. some random base images

docker image ls –a

docker image inspect IMAGE

docker image rm IMAGE

docker rmi IMAGE

# use the –f flag to automatically remove all tags and delete the image

# the –f flag does not delete the image if it is in use. (there is a container running. there will be an image running with no tag.)

# Dangling images are layers that have no relationship to any tagged images.

dockeri mage rm –f IMAGE\_ID

docker rmi –f IMAGE\_ID

# delete dangling images

docker image prune

FROM

specify the docker image what we want to use as base. Usually must be the first directive in the Dockerfile, ARG can be placed before

FROM alpine

FROM <image> as stage1

RUN

execute some commands while we are preparing our custom image

Creates a new layer on top of the previous layer by running a command inside that new layer and comiting the changes.

RUN apk add –update redis

CMD

what shuld be executed when our image is used to start up a brand new container

CMD [“redis-server”]

CMD ["executable","param1","param2"]

LABEL

MAINTAINER (deprecated)

EXPOSE

EXPOSE 3000

to expose a given port

ENV

Set environment variables. These can be referenced in the Dockerfile itself and are visible to the container at runtime

ADD

Allows src to be a URL. If is a local tar archive in a recognized compression format (identity, gzip, bzip2 or xz) then it is unpacked as a directory

COPY

COPY <src> <dest>

COPY ./ ./

COPY index.html /var/www/index.html

COPY hom\* /mydir/

COPY hom?.txt /mydir/

COPY test.txt relativeDir/

COPY test.txt /absoluteDir/

# multi-stage build

# selectively copy only the files you need from previous layers

FROM <image> as stage1

COPY --from=stage1 ...

ENTRYPOINT

VOLUME

USER

WORKDIR

WORKDIR /path/to/workdir

The WORKDIR instruction sets the working directory for any RUN, CMD, ENTRYPOINT, COPY and ADD instructions that follow it in the Dockerfile. If the WORKDIR doesn’t exist, it will be created even if it’s not used in any subsequent Dockerfile instruction. Effects even the docker exec command.

ARG

ONBUILD

STOPSIGNAL

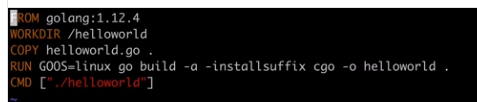
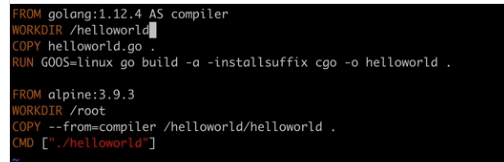
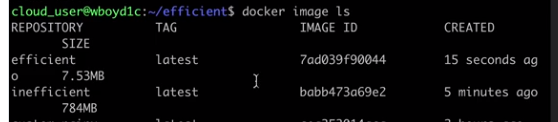
Specifies the signal that will be used to stop the container. Most of the times the default stop signal is fine, but if you have some custom software you can set here what stop signal to send when running the docker stop command.

STOPSIGNAL SIGTERM

HEALTCHECK

Spacify a command to run in order to perform a custom health check to verify that the container is working properly

HEALTHCHECK CMD curl localhost:80

* 1. Building Efficient images
* as small as possible
* ephemeral images, so if the image is destroyed not a problem (decoupled)
* put on lower level frequently changing images (utilize caching)
* avoid creating unnecessary layers (boundle)
* avoid including unnecessary files, packages
* multi-stage build (with multiple from directives)
* each stage begins a completely new set of file system laters, allowing ou to selectively copy only the files you need from pervious layers.
* Use the - -from flag with COPY to copy files from a previous stage.
* example: from alpine, run xx (create some layers, generate files) then a new from with selective copy
* 
* 
* 

1. Swarm
   1. Docker Compose

Allows to run multicontainer applications

When you are using Swarm you should use Stack instead of Docker Compose.

The DCA is interested only in their relation to docker swarm.

* 1. Docker Stacks
* docker run is used to create a standalone container
* docker service create is used to create instances (called tasks) of that service running in a cluster (called swarm) of computers (called nodes). Those tasks are containers of cource, but not standalone containers. In a sense a service acts as a template when instantiating tasks

Docker stacks are similar to the multi-container applications created using Docker Compose. However, they can be scaled and executed across the swarm just like normal swarm services.

Docker Stacks also uses a yaml config file (same sintax as docker compose)

1. Whizlabs learnings
   1. Replicated and global services

There are two types of service deployments, replicated and global.

For a **replicated service**, you specify the number of identical tasks you want to run. For example, you decide to deploy an HTTP service with three replicas, each serving the same content.

A **global service** is a service that runs one task on every node. There is no pre-specified number of tasks. Each time you add a node to the swarm, the orchestrator creates a task and the scheduler assigns the task to the new node. Good candidates for global services are monitoring agents, an anti-virus scanners or other types of containers that you want to run on every node in the swarm.

