Cheatography

Docker, Docker-Compose and Docker-Swarm Cheat Sheet

by Christian Knell (christian.knell) via cheatography.com/89637/cs/20403/

docker

CONCEPTS

Docker	A platform to develop, deploy and run applications with containers.
Dockerfile	A text document that contains all the commands a user could call on the command line to assemble an image.
Layer	Each instruction in a Dockerfile creates a layer in the image, where each layer is a set of differences from the previous layer.
Image	An executable package that includes everything needed to run an application--the code, a runtime, libraries, environment variables, and configuration files.
Container	A runtime instance of an image — what the image becomes in memory when executed (that is, an image with state, or a user process).
Service	Runs one image, but it codifies the way that image runs — what ports it should use, how many replicas of the container should run so the service has the capacity it needs, and so on.
Stack	A group of interrelated services that share dependencies, and can be orchestrated and scaled together. A single stack is capable of defining and coordinating the functionality of an entire application.

NETWORK TYPES

Bridge (default)	Allows containers connected to the same bridge network to communicate, while providing isolation from containers which are not connected to that bridge network.
Overlay (distributed, docker swarm)	Creates a distributed network among multiple Docker daemon hosts.
Host (useful for performance optimization)	The container's network is not isolated from the Docker host. The container shares the host's networking namespace and does not get its own IP-address allocated.
Macvlan	Connects the container directly to the physical network and assigns a MAC address to each container's virtual network interface.
Disabled	Disabled the networking stack on a container.

By **Christian Knell** (christian.knell)
cheatography.com/christian-knell/

Not published yet.
Last updated 2nd September, 2019.

docker

STORAGE TYPES

Volumes (preferred way to persist data)	A volume is stored within a directory on the Docker host and is mounted into the container. Volumes are managed by Docker and are isolated from the core functionality of the host. A volume can be mounted into multiple containers simultaneously. When you mount a volume, it may be **named** or **anonymous** - with no difference in their behaviour. Anonymous volumes get a random name by Docker that is guaranteed to be unique within the Docker host. Volumes support the use of volume drivers, which allow you to store your data on remote hosts or cloud providers.
Bind mounts (preferred way for sharing config-uration files)	A file or directory on the host machine is mounted into a container. The file or directory is referenced by its full path on the host machine. The file or directory does not need to exist on the Docker host already. It is created on demand if it does not yet exist.
tmpfs mounts (preferred way, when no need to persist data)	A tmpfs mount is not persisted on disk, either on the Docker host or within a container. It can be used by a container during the lifetime of the container, to store non-persistent state or sensitive information.
named pipes	An npipe mount can be used for communication between the Docker host and a container. Common use case is to run a third-party tool inside of a container and connect to the Docker Engine API using a named pipe.

docker

BUILD

`docker build -t IMAGE:TAG`	Build an image from the Dockerfile in the current directory and tag it
`-f /path/to/dockerfile`	Define the Dockerfile, which should be used
`--no-cache`	Force a complete new build from scrath
`docker image ls, docker images`	List all images that are locally stored within the Docker engine
`docker rmi IMAGE:TAG`	Delete an image from the local image store
`docker history IMAGE`	Show the layers of a Docker image

SHIP

`docker login my.registry.com:8000`	Log in to a registry (the Docker Hub by default)
`docker tag IMAGE:TAG REPOSITORY/IMAGE:TAG`	Retag a local image with a new image name and tag
`docker push REPOSITORY/IMAGE:TAG`	Push an image to a registry
`docker pull REPOSITORY/IMAGE:TAG`	Pull an image from a registry

RUN

`docker run [OPTIONS] IMAGE[:TAG]`	
`-d`	Run container in the background
`-it`	Connect the container to the current terminal
`-p PUBLISHED:TARGET`	Expost port PUBLISHED externally and map to port TARGET inside the container
`--name CONTAINERNAME`	Name the container with CONTAINERNAME
`--rm`	Remove the container automatically after it exists
`-v /PATH/TO/VOLUME`	Create a host mapped volume inside the container
`/bin/bash`	The command to run inside the container
`docker stop CONTAINERNAME`	Stop the running container CONTAINERNAME through SIGTERM
`docker kill CONTAINERNAME`	Stop the running container CONTAINERNAME through SIGKILL
`docker logs [OPTIONS] CONTAINERNAME`	Fetch the logs of a container named CONTAINERNAME
`--details`	Show extra details provided to logs
`--follow, -f`	Follow log output

RUN (cont)

`--tail LINES`	Number of LINES to show from the end of the logs
`--timestamps, -t`	Show timestamps

NETWORK

`docker network ls`	List networks
`docker network create [OPTIONS] NETWORKNAME`	Create a network named NETWORKNAME
`--driver, -d (bridge | overlay | macvlan)`	Driver to manage the Network
`--attachable`	Enable manual container attachment
`--gateway IP_ADDRESS`	IPv4 or IPv6 Gateway for the master subnet
`--subnet IP_ADDRESS/NETWORK`	Subnet in CIDR format that represents a network segment
`docker network inspect [OPTIONS] NETWORK [NETWORK...]`	Display detailed information on one or more networks
`--verbose, -v`	Verbose output for diagnostics
`docker network rm NETWORK [NETWORK...]`	Remove one or more networks
`docker network connect [OPTIONS] NETWORK CONTAINER`	Connect a container to a network
`--ip IP_ADDRESS`	IPv4 address (e.g., 172.30.100.104)
`--ip6 IP_ADDRESS`	IPv6 address (e.g., 2001:db8::33)
`docker network disconnect [OPTIONS] NETWORK CONTAINER`	Disconnect a container from a network
`--force, -f`	Force the container to disconnect from a network

VOLUMES

`docker volume ls`	List volumes
`docker volume create [OPTIONS] [VOLUME]`	Create a volume
`--driver, -d`	Specify volume driver name
`--name`	Specify volume name
`docker volume inspect VOLUME [VOLUME...]`	Display detailed information on one or more volumes
`docker volume rm [OPTIONS] VOLUME [VOLUME...]`	Remove one or more volumes
`--force, -f`	Force the removal of one or more volumes

MANAGE

`docker container ls, docker ps`	List all running containers
`docker system df`	Show docker disk usage
`--verbose, -v`	Show detailed information on space usage
`docker system prune [OPTIONS]`	Remove unused data
`--all, -a`	Remove all unused images not just dangling ones
`--force, -f`	Do not prompt for confirmation
`--volumes`	Prune volumes
`docker image prune [OPTIONS]`	Remove unused images
`--all, -a`	Remove all unused images not just dangling ones
`--force, -f`	Do not prompt for confirmation
`docker container prune [OPTIONS]`	Remove all stopped containers
`--force, -f`	Do not prompt for confirmation
`docker volume prune [OPTIONS]`	Remove all unused local volumes
`--force, -f`	Do not prompt for confirmation

docker-compose

docker-compose.yml

```
version: '3'
services:
  service1:
    image: registry/repository/image:tag
    depends_on:
      - service2
    env_file: path/to/file
    environment:
      - ENV_VAR=value
    networks:
      - network1
    ports:
      - "3000"
      - "3000-3005"
      - "8000:8000"
      - "9090-9091:8080-8081"
      - "127.0.0.1:8001:8001"
      - "127.0.0.1:5000-5010:5000-5010"
      - "6060:6060/udp"
    restart: (no | always | on-failure | unless-stopped)
    volumes:
      - /path/in/container                    # Just specify a path and let the Engine create a volume
      - /path/on/host:/path/in/container      # Specify an absolute path mapping
      - ./path/on/host:/path/in/container     # Path on the host, relative to the Compose file
      - ~/path/on/host:/path/in/container/:ro # User-relative path
      - namedvolume:/path/in/container        # Named volume
  service2:
    image: registry/repository/another_image:tag
networks:
  network1:
volumes:
  namedvolume:
```

docker-compose.yml (cont)

```
    driver: local # See https://docs.docker.com/engine/extend/legacy_plugins/#/volume-plugins  for other drivers
    external: (false | true) # If true, docker-compose does not attempt to create it
```

Komplette File-Reference: https://docs.docker.com/compose/compose-file

Docker-Compose Parameters

`docker-compose [options] [COMMAND]`

`--version, -v`	Print version
`--file, -f`	Specify an compose file (default: docker-compose.yml)
`--verbose`	Show more output
`--log-level LEVEL`	Set log level (DEBUG, INFO, WARNING, ERROR, CRITICAL)

Command Overview

`docker-compose up [OPTIONS]`	Starts all containers
`--detached, -d`	detached mode: Run containers in the background
`--force-recreate`	Recreate containers even if their configuration and image haven't changed
`--remove-orphans`	Remove containers for services not defined in the Compose file
`docker-compose down [OPTIONS]`	Stops containers and removes containers, networks, volumes, and images created by up
`--volumes, -v`	Remove named and anonymous volumes
`--remove-orphans`	Remove containers for services not defined in the Compose file
`docker-compose stop [SERVICE]`	Stops running containers without removing them
`docker-compose kill [SERVICE]`	Forces running containers to stop by sending a SIGKILL signal
`docker-compose rm [OPTIONS] [SERVICE...]`	Removes stopped service containers
`--force, -f`	Don't ask to confirm removal
`--stop, -s`	Stop the containers before removing
`-v`	Remove any anonymous volumes attached to containers
`docker-compose pull SERVICE`	Pulls an image associated with the SERVCE
`docker-compose logs SERVICE`	Displays log output from the SERVICE

Cheatography

Docker, Docker-Compose and Docker-Swarm Cheat Sheet

by Christian Knell (christian.knell) via cheatography.com/89637/cs/20403/

Docker Swarm

SWARM AWAY

`docker swarm init`	Initialize swarm mode
`--advertise-addr IP`	listen on a specific interface
`docker swarm join-token (worker|manager)`	Create a join token for a worker|manager node
`docker swarm join --token <token> IP:2377`	Join an existing swarm (under IP) as a manager node
`docker node ls`	List the nodes participating in a swarm

ORCHESTRATE

`docker service ls`	List the services running in a swarm
`docker service ps SERVICENAME`	List the tasks of the SERVICENAME
`docker service create [OPTIONS] IMAGE`	Create a new service
`--replicas NUMBER`	NUMBER of tasks
`--publish, -p EXPOSED:TARGET`	Publish a port (TARGET) as a node port (EXPOSED)
`--name SERVICENAME`	Give the service a name called SERVICENAME
`docker service scale SERVICENAME=NUMBER`	Scale the SERVICENAME to NUMBER

By **Christian Knell**
(christian.knell)
cheatography.com/christian-knell/

Not published yet.
Last updated 2nd September, 2019.