**DOCKER COMMANDS**

**docker attach** Attach local standard input, output, and error streams to a running container

**docker build** Build an image from a Dockerfile

**docker builder** Manage builds

**docker checkpoint** Manage checkpoints

**docker commit** Create a new image from a container’s changes

**docker config** Manage Docker configs

**docker container** Manage containers

**docker context**  Manage contexts

**docker cp** Copy files/folders between a container and the local filesystem

**docker create** Create a new container

**docker deploy** Deploy a new stack or update an existing stack

**docker diff** Inspect changes to files or directories on a container’s filesystem

**docker engine** Manage the docker engine

**docker events** Get real time events from the server

**docker exec** Run a command in a running container

**docker export** Export a container’s filesystem as a tar archive

**docker history** Show the history of an image

**docker image** Manage images

**docker images** List images

**docker import** Import the contents from a tarball to create a filesystem image

**docker info** Display system-wide information

**docker inspect** Return low-level information on Docker objects

**docker kill**  Kill one or more running containers

**docker load** Load an image from a tar archive or STDIN

**docker login** Log in to a Docker registry

**docker logout** Log out from a Docker registry

**docker logs** Fetch the logs of a container

**docker manifest** Manage Docker image manifests and manifest lists

**docker network** Manage networks

**docker node** Manage Swarm nodes

**docker pause** Pause all processes within one or more containers

**docker plugin** Manage plugins

**docker port** List port mappings or a specific mapping for the container

**docker ps** List containers

**docker pull** Pull an image or a repository from a registry

**docker push** Push an image or a repository to a registry

**docker rename** Rename a container

**docker restart** Restart one or more containers

**docker rm** Remove one or more containers

**docker rmi** Remove one or more images

**docker run** Run a command in a new container

**docker save** Save one or more images to a tar archive (streamed to STDOUT by default)

**docker search** Search the Docker Hub for images

**docker secret** Manage Docker secrets

**docker service** Manage services

**docker stack** Manage Docker stacks

**docker start** Start one or more stopped containers

**docker stats** Display a live stream of container(s) resource usage statistics

**docker stop** Stop one or more running containers

**docker swarm** Manage Swarm

**docker system** Manage Docker

**docker tag** Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

**docker top** Display the running processes of a container

**docker trust** Manage trust on Docker images

**docker unpause**  Unpause all processes within one or more containers

**docker update** Update configuration of one or more containers

**docker version** Show the Docker version information

**docker volume** Manage volumes

**docker wait** Block until one or more containers stop, then print their exit codes

**1. Create a Container Using Images from Docker Hub**

$ docker create -it ubuntu:xenial bash

**2. List Docker Containers**

$ docker ps

$ docker ps -a

**3. Start Your Container**

$ docker start CONTAINER\_ID

**4. Connecting to Your Container**

$ docker attach CONTAINER\_ID

**5. Terminating the Container**

$ docker stop CONTAINER\_ID

**6. Deleting a Container**

$ docker rm CONTAINER\_ID

**7. Killing a Container**

$ docker kill CONTAINER\_ID

**8. Detaching from Containers**

Ctrl+P Ctrl+Q

**9. Copying Contents between Containers and the File System**

$ docker cp CONTAINER\_ID:/test\_file test\_file

$ docker cp test\_file CONTAINER\_ID:/test\_file

**10. Running Commands inside Containers**

$ docker exec CONTAINER\_ID command

$docker exec -it container\_id bash

**11. List all Docker Images**

$ docker images

**12. Removing Docker Images**

$ docker rmi <IMAGE\_ID>

$ docker rmi REPOSITORY:TAG

**13. Adding Persistent Volumes for Containers**

$ docker create -it -v $(pwd):/var/www ubuntu:xenial bash

**14. Creating and Running Containers using Docker Run**

$ docker run -it -d ubuntu:xenial bash

$ docker run --name <optional\_name> -dt <image\_name>

**15. Creating a Named Container**

$ docker create --name TEST nginx

**16. Exposing Ports for Containers**

$ docker run --name NEW nginx -p 8080:80

**17. Look for Changes in a Container**

$ docker diff CONTAINER\_ID

**18. Creating a New Image from Modified Containers**

$ docker commit CONTAINER\_ID REPOSITORY:TAG

**19. Monitoring Server for Docker Containers**

$ docker events

**20. Inspect Image Histories**

$ docker history IMAGE\_ID

**21. Loading Docker Images from Tars**

$ docker load < IMAGE.tar.gz

$ docker load --input IMAGE.tar

**22. Display System-Wide Information about Docker**

$ docker info

**23. Renaming Docker Containers**

$ docker rename CONTAINER NEW\_NAME

**24. Restarting Docker Containers**

$ docker restart CONTAINER[S]

**Docker Commands for Networking**

**25. List all Networks**

$ docker network ls

**26. Connect to Networks**

$ docker network connect NET CONTAINER\_ID

**27. Disconnect from Networks**

$ docker network disconnect NET CONTAINER\_ID

**28. Create Networks**

$ docker network create NET

**29. View Detailed Information on Networks**

$ docker network inspect NET

**30. Delete Networks**

$ docker network rm NET

**31. Delete All Unused Networks**

$ docker network prune

**32. Build Images from Dockerfile**

$ docker build . -t APP:v1

**33. Removing all Unused Resources**

$ docker system prune

**34. Removing Specific Volumes**

$ docker volume rm VOLUME\_NAME

**35. Searching Images in Docker Hub**

$ docker search nginx

**36. Delete Containers upon Exit**

$ docker run --rm --it xenial /bin/bash

**37. Get Networking Logs at Host**

$ journalctl -u docker.service

**38. Export Container File Systems as Tars**

$ docker export giant\_panda > new.tar

$ docker export --output="new.tar" giant\_panda

**39. View Version Information**

$ docker version

$ docker -v

**40. View Help Information**

$docker --help

$ docker command[S] --help

$ docker network --help

**41. You can also pull an image:**

$docker pull centos

**42.Container logs**

$docker logs centosprod -t

**43. Pause and upause container**

$docker pause centosprod

$docker unpause centosprod

**44. Docker login**

$docker login

**45. Image push**

$docker image push thetunnelix/centostunnelix

**Network**

**46. Connect a container to a network**

$docker network connect

$ docker run -itd --network=mynet busybox

**47.Create a network**

$docker network create

$ docker network create -d bridge my-bridge-network

$ docker network create -d overlay my-multihost-network

$ docker network create --driver=bridge --subnet=192.168.0.0/16 br0

$ docker network create --driver=bridge --subnet=172.28.0.0/16 --ip-range=172.28.5.0/24 --gateway=172.28.5.254 br0

$ docker network create -d overlay \

--subnet=192.168.1.0/25 \

--subnet=192.170.2.0/25 \

--gateway=192.168.1.100 \

--gateway=192.170.2.100 \

--aux-address="my-router=192.168.1.5" --aux-address="my-switch=192.168.1.6" \

--aux-address="my-printer=192.170.1.5" --aux-address="my-nas=192.170.1.6" \

my-multihost-network

docker network create –driver=bridge –subnet=172.31.16.0/20 –gateway=172.31.20.206 –gateway=52.66.208.192 br-rancher0

**48. Disconnect a container from a network**

$docker network disconnect

**49.Display detailed information on one or more networks**

$docker network inspect

**50. List networks**

$docker network ls

**51. Remove all unused networks**

$docker network prune

**52.Remove one or more networks**

$docker network rm

**Docker Run**

$ docker run -d -p 80:80 my\_image service nginx start

$ docker run -d -p 80:80 my\_image nginx -g 'daemon off;'

$ docker run -it --rm --pid=host myhtop

$ docker run --name my-redis -d redis

$ docker run -d --name redis example/redis --bind 127.0.0.1

$ # use the redis container's network stack to access localhost

$ docker run --rm -it --network container:redis example/redis-cli -h 127.0.0.1

$ docker network create -d bridge my-net

$ docker run --network=my-net -itd --name=container3 busybox

$ docker run -it --add-host db-static:86.75.30.9 ubuntu cat /etc/hosts

**To check restart count**

$ docker inspect -f "{{ .RestartCount }}" my-container

$ docker run --restart=always redis

This will run the redis container with a restart policy of always so that if the container exits, Docker will restart it.

$ docker run --restart=on-failure:10 redis

We set memory limit and disabled swap memory limit, this means the processes in the container can use 300M memory

$ docker run -it -m 300M --memory-swap -1 ubuntu:14.04 /bin/bash

We set both memory and swap memory, so the processes in the container can use 300M memory and 700M swap memory.

$ docker run -it -m 300M --memory-swap 1G ubuntu:14.04 /bin/bash

when the container consumes memory more than 200M and less than 500M

$ docker run -it -m 500M --memory-reservation 200M ubuntu:14.04 /bin/bash

$ docker run -it --memory-reservation 1G ubuntu:14.04 /bin/bash

**We set memory and kernel memory, so the processes in the container can use 500M memory in total, in this 500M memory, it can be 50M kernel memory tops.**

$ docker run -it -m 500M --kernel-memory 50M ubuntu:14.04 /bin/bash

$ docker run -it --memory-swappiness=0 ubuntu:14.04 /bin/bash

If there is 1 CPU, this means the container can get 50% CPU worth of run-time every 50ms.

$ docker run -it --cpu-period=50000 --cpu-quota=25000 ubuntu:14.04 /bin/bash

**Cpuset constraint**

**This means processes in container can be executed on cpu 1 and cpu 3.**

$ docker run -it --cpuset-cpus="1,3" ubuntu:14.04 /bin/bash

$ docker run -it --cpuset-cpus="0-2" ubuntu:14.04 /bin/bash

**Docker build**

$ docker build https://github.com/docker/rootfs.git#container:docker

$docker build .

**This exact the tar and store in to Dockerfile and then it run**

$ docker build -f ctx/Dockerfile http://server/ctx.tar.gz

**Docker commit**

Create a new image from a container’s changes

$docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

Options

**Name, shorthand** **Default Description**

--**author** , -a Author (e.g., “John Hannibal Smith hannibal@a-team.com”)

--**change** , -c Apply Dockerfile instruction to the created image

--**message** , -m Commit message

--**pause** , -p true Pause container during commit

**Docker service**

$ docker service ls

**Compose** **file**

The deploy command supports compose file version 3.0 and above.

$ docker stack deploy --compose-file docker-compose.yml vossibility

**Port test**

$ docker port test

$ docker port test 7890/udp

2014/06/24 11:53:36 Error: No public port '7890/udp' published for test

**Update**

**Update a container with cpu-shares and memory**

**To update multiple resource configurations for multiple containers:**

$ docker update --cpu-shares 512 -m 300M abebf7571666 hopeful\_morse

$ docker update --kernel-memory 80M test

**Docker Volume**

$ docker volume create hello

$ docker run -d -v hello:/world busybox ls /world

$ docker volume create --driver local --opt type=nfs --opt o=addr=192.168.1.1,rw --opt device=:/path/to/dir foo

**Related commands**

**Command** **Description**

docker volume create Create a volume

docker volume inspect Display detailed information on one or more volumes

docker volume ls List volumes

docker volume prune Remove all unused local volumes

docker volume rm Remove one or more volumes

#### Save and Load Docker Container Image to/from a tar file

[root@docker ~]# docker save debian -o mydebian.tar

# docker load -i mydebian.tar

#### Export and Import Container to / from tar archive

[root@docker ~]# docker export web\_container -o  web\_container.tar

[root@docker ~]# docker import web\_container.tar

**Resource usage statistics without live streaming**

[root@docker ~]# docker stats --no-stream

**Docker Compose**

**========================**

## Step 1 — Creating the Docker Compose File

**Mkdir docker-compose**

**Vi docker-compose.yaml**

|  |
| --- |
| **web:** |
|  | **image: nginx:latest** |
|  | **ports:** |
|  | **- "8080:80"** |
|  | **volumes:** |
|  | **- ./code:/code** |
|  | **- ./nginx.conf:/etc/nginx/conf.d/default.conf** |
|  | **links:** |
|  | **- php** |
|  |  |
|  | **php:** |
|  | **build: .** |
|  | **volumes:** |
|  | **- ./code:/code** |
|  | **environment:** |
|  | **XDEBUG\_CONFIG: remote\_host=host.docker.internal** |
|  | **links:** |
|  | **- mysql** |
|  | **- maildev** |
|  |  |
|  | **mysql:** |
|  | **image: mysql:latest** |
|  | **ports:** |
|  | **- "3306:3306"** |
|  | **environment:** |
|  | **- MYSQL\_ROOT\_PASSWORD=password** |
|  |  |
|  | **phpmyadmin:** |
|  | **image: phpmyadmin/phpmyadmin** |
|  | **links:** |
|  | **- mysql:db** |
|  | **ports:** |
|  | **- "8082:80"** |
|  | **environment:** |
|  | **- PMA\_USER=root** |
|  | **- PMA\_PASSWORD=password** |
|  | **- PHP\_UPLOAD\_MAX\_FILESIZE=100MB** |
|  |  |
|  | **maildev:** |
|  | **image: djfarrelly/maildev** |
|  | **ports:** |
|  | **- "8081:80"** |

**Example : 2**

|  |
| --- |
| **version: '3'** |
|  | **services:** |
|  | **web:** |
|  | **image: nginx:alpine** |
|  | **volumes:** |
|  | **- "./etc/nginx/default.conf:/etc/nginx/conf.d/default.conf"** |
|  | **- "./etc/ssl:/etc/ssl"** |
|  | **- "./web:/var/www/html"** |
|  | **- "./etc/nginx/default.template.conf:/etc/nginx/conf.d/default.template"** |
|  | **ports:** |
|  | **- "8000:80"** |
|  | **- "3000:443"** |
|  | **environment:** |
|  | **- NGINX\_HOST=${NGINX\_HOST}** |
|  | **command: /bin/sh -c "envsubst '$$NGINX\_HOST' < /etc/nginx/conf.d/default.template > /etc/nginx/conf.d/default.conf && nginx -g 'daemon off;'"** |
|  | **restart: always** |
|  | **depends\_on:** |
|  | **- php** |
|  | **- mysqldb** |
|  | **php:** |
|  | **image: nanoninja/php-fpm:${PHP\_VERSION}** |
|  | **restart: always** |
|  | **volumes:** |
|  | **- "./etc/php/php.ini:/usr/local/etc/php/conf.d/php.ini"** |
|  | **- "./web:/var/www/html"** |
|  | **composer:** |
|  | **image: "composer"** |
|  | **volumes:** |
|  | **- "./web/app:/app"** |
|  | **command: install** |
|  | **myadmin:** |
|  | **image: phpmyadmin/phpmyadmin** |
|  | **container\_name: phpmyadmin** |
|  | **ports:** |
|  | **- "8080:80"** |
|  | **environment:** |
|  | **- PMA\_ARBITRARY=1** |
|  | **- PMA\_HOST=${MYSQL\_HOST}** |
|  | **restart: always** |
|  | **depends\_on:** |
|  | **- mysqldb** |
|  | **mysqldb:** |
|  | **image: mysql:${MYSQL\_VERSION}** |
|  | **container\_name: ${MYSQL\_HOST}** |
|  | **restart: always** |
|  | **env\_file:** |
|  | **- ".env"** |
|  | **environment:** |
|  | **- MYSQL\_DATABASE=${MYSQL\_DATABASE}** |
|  | **- MYSQL\_ROOT\_PASSWORD=${MYSQL\_ROOT\_PASSWORD}** |
|  | **- MYSQL\_USER=${MYSQL\_USER}** |
|  | **- MYSQL\_PASSWORD=${MYSQL\_PASSWORD}** |
|  | **ports:** |
|  | **- "8989:3306"** |
|  | **volumes:** |
|  | **- "./data/db/mysql:/var/lib/mysql"** |

**Example 3 :**

**version: '3'**

**services:**

**#PHP Service**

**app:**

**build:**

**context: .**

**dockerfile: Dockerfile**

**image: digitalocean.com/php**

**container\_name: app**

**restart: unless-stopped**

**tty: true**

**environment:**

**SERVICE\_NAME: app**

**SERVICE\_TAGS: dev**

**working\_dir: /var/www**

**networks:**

**- app-network**

**#Nginx Service**

**webserver:**

**image: nginx:alpine**

**container\_name: webserver**

**restart: unless-stopped**

**tty: true**

**ports:**

**- "80:80"**

**- "443:443"**

**networks:**

**- app-network**

**#MySQL Service**

**db:**

**image: mysql:5.7.22**

**container\_name: db**

**restart: unless-stopped**

**tty: true**

**ports:**

**- "3306:3306"**

**environment:**

**MYSQL\_DATABASE: laravel**

**MYSQL\_ROOT\_PASSWORD: your\_mysql\_root\_password**

**SERVICE\_TAGS: dev**

**SERVICE\_NAME: mysql**

**networks:**

**- app-network**

**#Docker Networks**

**networks:**

**app-network:**

**driver: bridge**