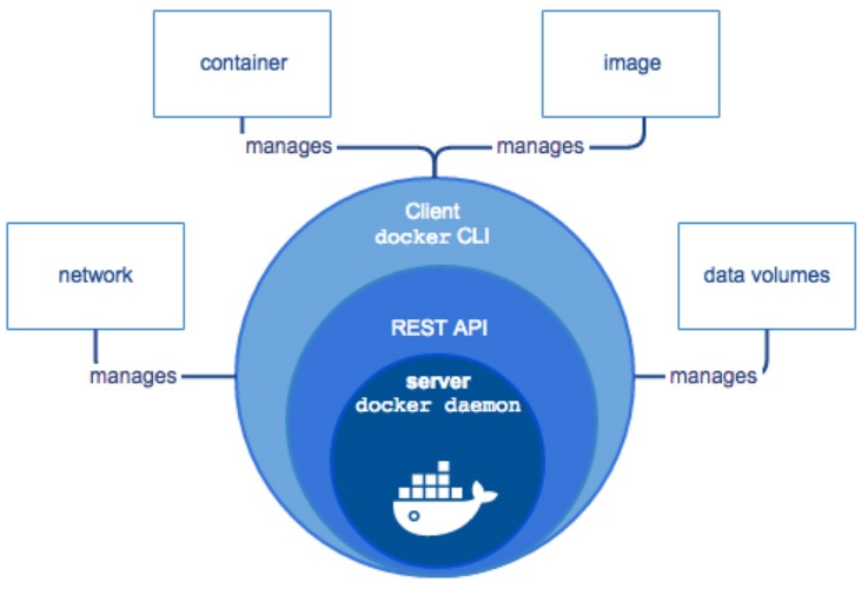
Docker Cheatsheet

This is a quick and dirty Docker, Docker-Machine and Docker Swarm cheatsheet, this is still being worked on.

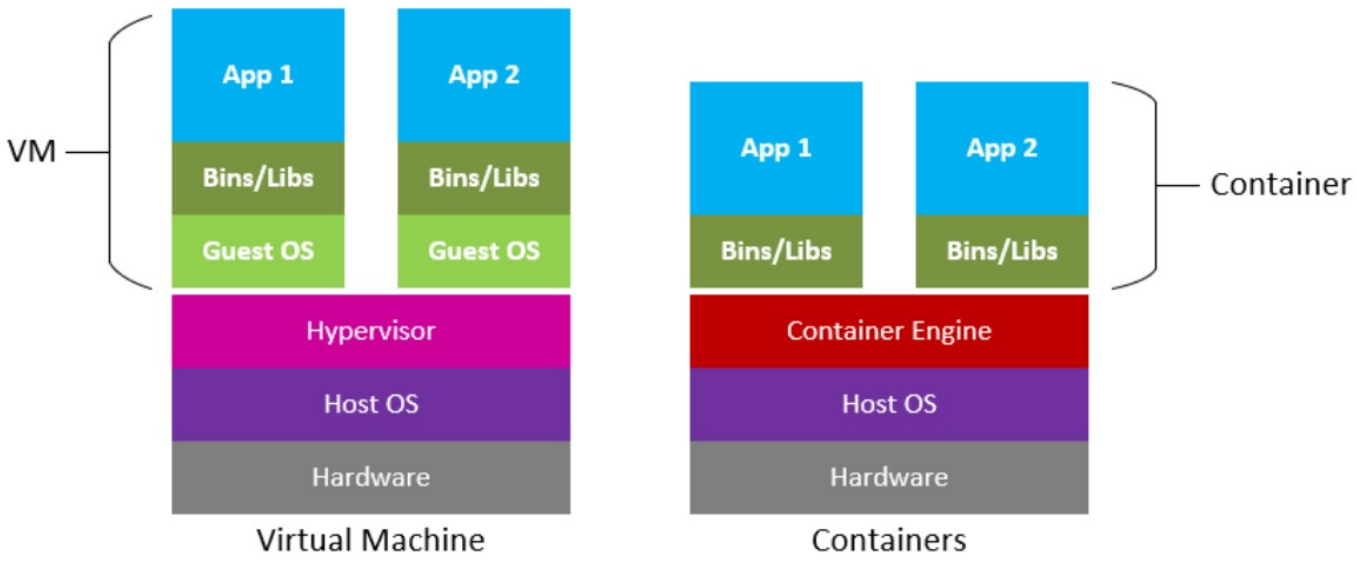
First lets start with some Docker terminology, below are the commonly used parts of the Docker Platform

Docker Server	is a server with Docker installed and has images and containers (running), its where the Docker Engine (see below) is running.
Docker Client	is the primary way that many Docker users interact with Docker
Docker Machine	Docker Machine allows you to provision Docker machines in a variety of environments, including virtual machines that reside on your local system, on cloud providers, or on bare metal servers (physical computers)
Docker Compose	is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.
Docker Hub	is the world's easiest way to create, manage, and deliver your teams' container applications
Docker Image	single file with all the dependencies and configuration required to run a specific program (for example Redis, JBoss, etc). The image will also contain a startup command to start what ever service you need to be running.
Docker Container	is a instance of an image, its like a running program that offers a service (for example Redis, JBoss, etc), a container has its own memory, own networking, etc basically it's isolated from other containers. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, etc. Think of a container as a set processes that has access to a group of reources specify assigned to it.
Docker Engine	is a client-server application with these major components • A server which is a type of long-running program called a daemon process (the dockerd command) • A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do • A command line interface (CLI) client (the docker command)

The Docker Platform can be seen in the below image



The below diagram displays the difference between a virtual machine (VMware) and a Docker container, the big difference is that the running containers share the host OS kernel, virtual machines use a guest O/S.



Docker

Below are some of the commonly used docker and docker-compose commands

Docker and docker-compose installation

```
yum install docker-engine

# Docker-Compose installation
yum install epel-release
yum install -y python-pip

pip install docker-compose

yum upgrade python*                                                    # update python

pip install backports.ssl_match_hostname --upgrade                     # if get hostname
mismatch error you can use either container ID or container name
```

Directories

```
/var/lib/docker                                                        # Docker directory
/var/lib/docker/containers
```

General commands

```
docker version                                                         # get both client and
server versions
docker info                                                            # get the server docker
detailed information

docker login --username=datadiskpfv --email=paul.valle@datadisk.co.uk  # login into docker hub
docker logout                                                          # logout of docker hub

docker events                                                          # print out realtime
docker server events, you have filter start/end times

docker system prune [-a]                                               # delete all stopped
containers, unused networks, dangling images and build cache
```

Images

```
docker search <search_string>                                          # search for images on
docker.io

docker images                                                          # list all images
docker rmi [-f] <image>                                                # remove an image
docker history [--no-trunc] <image>                                    # display the history
(layers) of an image
docker inspect <image>                                                 # inspect an images, get
lots of detail
docker tag <image:tag> <image:tag>                                     # associates a repo and a
tag name with an image

docker [save|load] [-o|-i]                                             # save/loads an images
to/from a tar file, (-o = output) (-i = input)

docker commit -c 'CMD ["<command>"]' <container ID> test/cowsayimage:latest     # keep a running image
and specify a startup command

docker push <image>                                                    # when pushing images
make sure you are logged in (see general commands)
docker pull <image>                                                    # when pulling images
make sure you are logged in (see general commands)
```

Volumes

```
docker volume create                                                    # create a docker volume
docker volume ls                                                        # list all docker volumes
docker volume rm <volume name>                                          # remove a docker volume
docker volume inspect <volume name>                                     # inspect a docker volume
```

Containers

```
docker ps [-a] [--no-trunc]                                             # list the containers (-a
will list all stopped containers)
docker stop <container ID>                                              # stop a running
container gracefully if possible, otherwise a kill will be sent after 10 seconds
docker kill <container ID>                                              # stop a running
container in its tracks
docker start [-i|-a] <container ID>                                     # start an existing
container (use docker ps -a to list all)
docker rm <container ID>                                                # remove a container
(make sure its stopped first)
docker rm -v $(docker ps -aq -f status=exited)                          # remove all exited
containers
docker rm $(docker ps -aq)                                              # another version of
above
docker top <container ID>                                               # TOP process command for
a container

docker export/import                                                    # export/import an images
to a tar file (no layer history, port CMD, endpoints, etc)

docker create <image>                                                   # create a container
using a image

docker rename <container ID> <new name>                                 # rename a container

docker inspect <container ID>                                           # get container
information (get IP address for example)
docker inspect --format '{{.NetworkSettings.IPAddress}}' <container ID>         # get specific container
information
docker inspect -f '{{.Mounts}}' <container ID>                          # another example this
time looking at Mounts

docker diff <container ID>                                              # see if any changes to
the container has happened

docker logs <container ID> [-f]                                         # list what has happened
inside container, -f like tail

docker exec -it <container ID> bash                                     # connect to a running
container, must have image id and bash at end
docker exec -it --user <user> <container ID> sh                         # connect to a running
container as a specific user, also using shell this time
docker attach <container ID>                                            # connect to a running
container, when disconnect stops container (use ctrl-P and then Ctrl-Q to ext with stopping)

docker port <container ID>                                              # display any ports that
are used by the container
```

Run commands

```
docker run -i -t debian /bin/bash                                       # run a container (image
debian) but give us a shell inside container (-t = tag/image, -i interactive shell)

docker run test/cowsay-dockerfile                                       # start the container
(will create a separate container for each run)
docker run --rm -it --link myredis:redis redis /bin/bash                # start a container and
link it to the myredis container

docker run -it --name container-test -h CONTAINER -v /docker/cowsay/data:/data debian /bin/bash
# the v means mount /data in the container to the external /home/pvalle/data
docker run -it -h NEWCONTAINER --volumes-from container-test debian /bin/bash
# mount the above /data in a new container, now they share

docker run --rm --volumes-from dbdata -v $(pwd):/backup debian tar cvf /backup/backup.tar
/var/lib/postgresql/data  # backup a containers  postgresql /data area

docker run -d -p 8000:80 nginx                                          # start a container bind
a local port 8000 to the container port 80
docker run -d -P nginx                                                  # same as above but
docker will select a free server port to connect to container port 80

docker run -p 8080:8080 -p 9990:9990 -p 9999:9999 -it jboss             # run a jboss contain and
assign ports
```

Dockerfile and Images

A Dockerfile is a text document that contains all the commands a user could call on the command line to assemble an image. below are some (but not all) of the commonly used instructions, a dockerfile must start with the FROM instruction

- **FROM** - This will set the base image using the parent image you have specified, examples are hello-word, ubuntu,
- **COPY** - add files or directories to the image, this is more simple than ADD
- **ADD** - add files or directories to the image, has more features than COPY
- **ENV** - used to define environment variables
- **RUN** - will execute commands, useful if you want to update the OS image for example, or create a user, etc
- **VOLUME** - tell Docker to store specific files in a specific directory that should be stored on the host file system not in the containers file system
- **USER** - from this point forward run as a specific user
- **WORKDIR** - define a working directory, handy if you need to copy files to a specific place
- **EXPOSE** - inform your users about the ports your application is listening on for example port 80 for a HTTP connection.
- **CMD** - is the instruction to specify what component is to be run by your image with arguments
- **ENTRYPOINT** - helps you to configure a container that you can run as an executable

Docker build commands

```
docker build .                                          # Will use Dockfile in current directory

docker build -f Dockerfile.dev .                        # use the -f to specify the name of the docker
file

# tag name convention is <your Docker ID>/<repo or project name>:<version>
docker build -t test/cowsay-dockerfile:latest .         # build a image (the . means use a Dockerfile, -t
tag name)

Note: the default name of a docker file is Dockerfile
```

Docker file example

```
FROM node:alpine

# change to /app directory in container
WORKDIR '/app'

# copy from filesytem to container
COPY ./package.json ./

# run the command npm install inside the container
RUN npm install

# copy from filesytem to container
COPY . .

# run the command "npm run start" inside the container
CMD ["npm", "run", "start"]
```

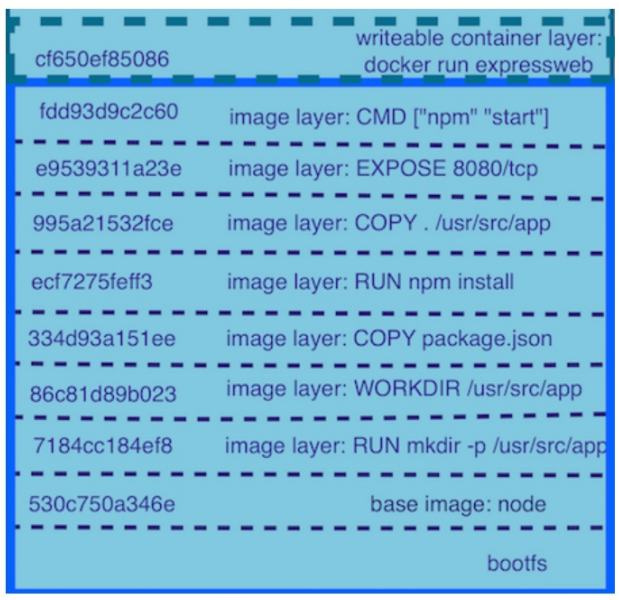
Docker file example

```
# build image for JBoss EAP 7.1

# Use parent image eap71-openshift
FROM registry.access.redhat.com/jboss-eap-7/eap71-openshift

# file author / maintainer
MAINTAINER "FirstName LastName" "emailaddress@gmail.com"

# Deploy your application by copying war file to deployments folder
COPY app.war $JBOSS_HOME/standalone/deployments/

# User root to modify war owners and create volume
USER root

# Modify owners war
RUN chown jboss:jboss $JBOSS_HOME/standalone/deployments/app.war

# Specify a external volume to keep the log files
VOLUME /opt/jboss/wildfly/standalone/log

# Important, use jboss user to run image
USER jboss
```

Using phases

```
# Phase One
FROM node:alpine as builder                    # will build in /app/build and be used in phase two
WORKDIR '/app'
COPY package.json .
RUN npm install
COPY . .
RUN npm run build

# Phase Two (using phase one build)
FROM nginx
COPY --from=builder /app/build /usr/share/nginx/html
```

The docker image created is built in layers, the first layer being the parent image (FROM instruction), then any additional instructions are a specific layer within the image as per the diagram below. Docker when rebuilding is clever enough to know what layers have been previous built and will use those existing layers instead of rebuilding them if they have not changed, if no changes are made then it will use the already cached version.



Docker Compose

Docker Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

There are many commands you can use with the docker-compose command

```
Commands:
  build              Build or rebuild services
  bundle             Generate a Docker bundle from the Compose file
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show the Docker-Compose version information
```

Below are some docker Compose commands and some example docker-compose files.

Docker compose commands

```
docker-compose [up|down]                                                    # start/stop the
containers using the docker-compose.yml file
docker-compose up [-d] [--build]                                            # fork to the background

docker-compose [stop|start]                                                 # stop/start the
containers using the docker-compose.yml file

docker-compose ps                                                           # show the docker group
of containers

docker-compose rm                                                           # remove existing docker-
compose containers, again it eill use the yml file

docker-compose logs                                                         # displays the logs for
the composed managed containers

docker-compose build [--no-cache]                                           # rebuilds the docker
images using the Dockerfile, no-cache forces complete rebuild

docker-compose run                                                          # spins up a container to
run a one-off command

Note: most of the docker-compose commands the docker-compose file needs to be in the current directory
```

docker-compose.yml file

```
version: "3"
services:
  redis-server:
    image: "redis"
  node-app:
    restart: on-failure
    build: .
    ports:
      - "4001:8081"
```

docker-compose.yml file

```
version: "3"
services:
  mysql:
    image: "mysql:5.7"
    container_name: mysql
    restart: always
    volumes:
      - ./mysql:/var/lib/mysql                          # <local filesystem>:<container filesystem>
    environment:                                        # key/value pair array
      - MYSQL_ROOT_PASSWORD=your_password
      - MYSQL_USER=root
      - MYSQL_PASSWORD=your_password
      - MYSQL_DATABASE=wordpress
    ports:
      - "3306:3306"                                     # <external port>:<container port>

  my_super_app:
    build:
        context: ./my_super_app                          # dockerfile will be located in this directory
        dockerfile: Dockerfile_super_app.dev             # specific Dockerfile
    container_name: my_supper_app
    depends_on:
      - mysql
    command: ["run", "my", "super_app"]
```

Docker-Machine

Docker Machine is a tool that lets you install Docker Engine on virtual hosts, and manage the hosts with docker-machine commands, its not commonly used but below are some of the commonly used docker-machine commands

Docker-Machine commands	

```
docker-machine create --driver digitalocean --digitalocean-access-token <access token> dockertest1

docker-machine create --driver generic --generic-ip-address <IP Address> --generic-ssh-
key=/root/.ssh/id_rsa --generic-ssh-user root dockertest1

docker-machine [commands - see below] dockertest1

Commands:
  active               Print which machine is active
  config               Print the connection config for machine
  create               Create a machine
  env                  Display the commands to set up the environment for the Docker client
  inspect              Inspect information about a machine
  ip                   Get the IP address of a machine
  kill                 Kill a machine
  ls                   List machines
  provision            Re-provision existing machines
  regenerate-certs     Regenerate TLS Certificates for a machine
  restart              Restart a machine
  rm                   Remove a machine
  ssh                  Log into or run a command on a machine with SSH.
  scp                  Copy files between machines
  start                Start a machine
  status               Get the status of a machine
  stop                 Stop a machine
  upgrade              Upgrade a machine to the latest version of Docker
  url                  Get the URL of a machine
  version              Show the Docker Machine version or a machine docker version
  help                 Shows a list of commands or help for one command
```

Docker Swarm

Below are some of the commonly used docker swarm commands

Setup	

```
docker-machine ip dockerm01                                                    # 192.168.1.71
docker-machine ip dockers01                                                    # 192.168.1.72
docker-machine ip dockers02                                                    # 192.168.1.73

docker swarm init --advertise-addr 192.168.1.71                                # manager, set up
manager node

docker swarm join-token worker                                                 # manager, get
join command for worker (see below)
docker swarm join-token manager                                                # manager, get
join command for manager

docker swarm join --token <token> 192.168.1.71:2377
```

Docker Swarm commands	

```
docker node ls                                                                 # manager

docker service create --replicas 2 -p 80:80 --name web nginx                   # manager

docker service rm web                                                          # manager

docker service ls                                                              # manager

docker service ps [web]                                                        # manager
docker node ps dockers01                                                       # manager

docker service scale web=2                                                     # manager

docker node inspect [self|dockers01|dockers02]                                 # manager

docker node inspect --pretty [self|dockers01|dockers02]                        # manager

docker node update --availability drain dockers02                              # manager
docker node update --availability active dockers02                             # manager

docker service update --image <imagename>:<version> web                        # manager

docker swarm leave                                                             # client or
manager
docker node demote <ID>                                                        # manager
docker node rm <ID>                                                            # manager

docker network ls                                                              # manager
```

Return to Main Page