Docker COMMANDS Cheat Sheet

Created By: Naveen Khunteta - Naveen Automation Labs
Linkedin: https://www.linkedin.com/in/naveenkhunteta/

##
Docker is a containerization platform that allows developers to package and deploy applications in a portable and efficient way.
Docker containers are lightweight, standalone environments that include everything needed to run an application, including its code, runtime, system tools, libraries, and settings.

Here are some basic concepts to understand about Docker:

1. **Docker Image**: An image is a snapshot of a container at a specific point in time. It includes the application code, dependencies, and other files needed to run the application.
2. **Docker Container**: A container is a running instance of an image. It is a lightweight, portable environment that contains all the necessary dependencies to run the application.
3. **Docker Registry**: A registry is a repository for Docker images. Docker Hub is the most popular public registry, but companies can also set up their own private registries.
4. **Dockerfile**: A Dockerfile is a text file that contains instructions for building a Docker image. It includes information about the application code, dependencies, and other settings needed to create an image.
5. **Docker Compose**: Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define the different services that make up an application and specify how they interact with each other.

Docker is widely used in software development and deployment because it makes it easier to manage and deploy applications in different environments, such as development, testing, and production.
##

##
 # DOCKER COMMANDS
##

```
docker build -t friendlyname .             # Create image using this directory's Dockerfile
docker run -p 4000:80 friendlyname         # Run "friendlyname" mapping port 4000 to 80
docker run -d -p 4000:80 friendlyname      # Same thing, but in detached mode
docker exec -it [container-id] bash        # Enter a running container
docker ps                                  # See a list of all running containers
docker stop <hash>                         # Gracefully stop the specified container
docker ps -a                               # See a list of all containers, even the ones not running
docker kill <hash>                         # Force shutdown of the specified container
docker rm <hash>                           # Remove the specified container from this machine
docker rm -f <hash>                        # Remove force specified container from this machine
docker rm $(docker ps -a -q)               # Remove all containers from this machine
docker images -a                           # Show all images on this machine
docker rmi <imagename>                     # Remove the specified image from this machine
docker rmi $(docker images -q)             # Remove all images from this machine
docker logs <container-id> -f              # Live tail a container's logs
docker login                               # Log in this CLI session using your Docker credentials
docker tag <image> username/repository:tag  # Tag <image> for upload to registry
docker push username/repository:tag        # Upload tagged image to registry
docker run username/repository:tag         # Run image from a registry
docker system prune                        # Remove all unused containers, networks, images (both dangling and unreferenced), and optionally,
volumes. (Docker 17.06.1-ce and superior)
docker system prune -a                     # Remove all unused containers, networks, images not just dangling ones (Docker 17.06.1-ce and superior)
docker volume prune                        # Remove all unused local volumes
docker network prune                       # Remove all unused networks
```

```
################################################################################
# DOCKER COMPOSE COMMANDS
################################################################################

docker-compose up                       # Create and start containers
docker-compose up -d                    # Create and start containers in detached mode
docker-compose down                     # Stop and remove containers, networks, images, and volumes
docker-compose logs                     # View output from containers
docker-compose restart                  # Restart all service
docker-compose pull                     # Pull all image service
docker-compose build                    # Build all image service
docker-compose config                   # Validate and view the Compose file
docker-compose scale <service_name>=<replica>   # Scale special service(s)
docker-compose top                      # Display the running processes
docker-compose run -rm -p 2022:22 web bash      # Start web service and runs bash as its command, remove old container.



################################################################################
# DOCKER SERVICES
################################################################################

docker service create <options> <image> <command>   # Create new service
docker service inspect --pretty <service_name>      # Display detailed information Service(s)
docker service ls                                   # List Services
docker service ps                                   # List the tasks of Services
docker service scale <service_name>=<replica>       # Scale special service(s)
docker service update <options> <service_name>      # Update Service options



################################################################################
# DOCKER STACK
################################################################################

docker stack ls                                     # List all running applications on this Docker host
docker stack deploy -c <composefile> <appname>  # Run the specified Compose file
docker stack services <appname>                     # List the services associated with an app
docker stack ps <appname>                           # List the running containers associated with an app
docker stack rm <appname>                           # Tear down an application



################################################################################
# DOCKER MACHINE
################################################################################

docker-machine create --driver virtualbox myvm1                 # Create a VM (Mac, Win7, Linux)
docker-machine create -d hyperv --hyperv-virtual-switch "myswitch" myvm1  # Win10
docker-machine env myvm1                                        # View basic information about your node
docker-machine ssh myvm1 "docker node ls"                       # List the nodes in your swarm
docker-machine ssh myvm1 "docker node inspect <node ID>"        # Inspect a node
docker-machine ssh myvm1 "docker swarm join-token -q worker"    # View join token
docker-machine ssh myvm1                                        # Open an SSH session with the VM; type "exit" to end
docker-machine ssh myvm2 "docker swarm leave"                   # Make the worker leave the swarm
```

```
docker-machine ssh myvm1 "docker swarm leave -f"          # Make master leave, kill swarm
docker-machine start myvm1                                # Start a VM that is currently not running
docker-machine stop $(docker-machine ls -q)               # Stop all running VMs
docker-machine rm $(docker-machine ls -q)                 # Delete all VMs and their disk images
docker-machine scp docker-compose.yml myvm1:~             # Copy file to node's home dir
docker-machine ssh myvm1 "docker stack deploy -c <file> <app>"   # Deploy an app
```

```
################################################################################
# DOCKER FILE
################################################################################
```

Example of a basic **Dockerfile** that sets up a **Node.js** environment:

Dockerfile

```dockerfile
# Specify the base image to use
FROM node:14-alpine

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the container
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application files to the container
COPY . .

# Expose port 3000
EXPOSE 3000

# Start the application
CMD [ "npm", "start" ]
```

Example of a **Dockerfile** that builds a Java application using Maven:

Dockerfile

```dockerfile
# Specify the base image to use
FROM maven:3.8.4-jdk-11

# Copy the project files to the container
COPY . /usr/src/app

# Set the working directory
WORKDIR /usr/src/app

# Build the application
RUN mvn package

# Expose port 8080
EXPOSE 8080

# Start the application
CMD ["java", "-jar", "target/my-app.jar"]
```

References:

- Docker Docs : https://docs.docker.com/engine/reference/commandline/docker/
- Docker Awesome Cheat Sheet