

#_ the Ultimate DOCKER Quick Reference

1. Basic Docker Concepts and Terms:

1. Docker Image:

A lightweight, stand-alone, executable package that includes everything needed to run a piece of software.

2. Docker Container:

A runtime instance of a Docker image.

3. Docker Hub:

A cloud-based registry service where Docker users and partners create, test, store, and distribute container images.

4. Dockerfile:

A text document that contains all the commands a user could call on the command line to assemble an image.

5. Docker Compose:

A tool for defining and running multi-container Docker applications.

2. Basic Docker Commands:

- `docker --version`: Display Docker version.
- `docker info`: Display system-wide information.
- `docker run image`: Run a Docker container from an image.
- `docker ps`: List running Docker containers.
- `docker ps -a`: List all Docker containers.
- `docker stop container_id`: Stop a running container.
- `docker rm container_id`: Remove a Docker container.
- `docker images`: List Docker images.
- `docker rmi image_id`: Remove a Docker image.

- `docker pull image`: Pull an image from a Docker registry (Docker Hub by default).
 - `docker push image`: Push an image to a Docker registry.
 - `docker exec -it container_id command`: Execute a command in a running container.
 - `docker logs container_id`: Fetch the logs of a container.
 - `docker start`: Starts one or more stopped containers.
 - `docker stop`: Stops one or more running containers.
 - `docker build`: Builds an image from a Dockerfile.
 - `docker pull`: Pulls an image or a repository from a registry.
 - `docker push`: Pushes an image or a repository to a registry.
 - `docker export`: Exports a container's filesystem as a tar archive.
 - `docker exec`: Runs a command in a run-time container.
 - `docker search`: Searches the Docker Hub for images.
 - `docker attach`: Attaches to a running container.
 - `docker commit`: Creates a new image from a container's changes.
-

3. Intermediate Docker Commands:

- `docker run -d image`: Run a Docker container in detached mode.
 - `docker run -p host_port:container_port image`: Map a port from the host to a container.
 - `docker run -v host_volume:container_volume image`: Mount a volume from the host to a container.
 - `docker run -e VAR=VALUE image`: Set environment variables in a container.
 - `docker inspect container_id/image_id`: Return low-level information on Docker objects.
 - `docker build -t tag .`: Build a Docker image with a tag from a Dockerfile in the current directory.
 - `docker tag image new_tag`: Tag an image with a new tag.
-

4. Dockerfile Commands:

- **FROM image**: Set the base image.
 - **RUN command**: Run a command.
 - **CMD command**: Set a default command that will run when the container starts.
 - **ENV VAR=VALUE**: Set environment variables.
 - **ADD source destination**: Copy files from source to the container's filesystem at the destination.
 - **COPY source destination**: Copy new files or directories from source and add them to the filesystem of the container at the destination.
 - **ENTRYPOINT command**: Allow you to configure a container that will run as an executable.
 - **LABEL**: Adds metadata to an image.
 - **EXPOSE**: Informs Docker that the container listens on the specified network ports at runtime.
 - **ENTRYPOINT**: Allows you to configure a container that will run as an executable.
 - **VOLUME**: Creates a mount point with the specified name and marks it as holding externally mounted volumes from native host or other containers.
 - **USER**: Sets the user name (or UID) and optionally the user group (or GID) to use when running the image and for any **RUN**, **CMD** and **ENTRYPOINT** instructions that follow it in the Dockerfile.
 - **WORKDIR**: Sets the working directory for any **RUN**, **CMD**, **ENTRYPOINT**, **COPY**, and **ADD** instructions that follow it in the Dockerfile.
 - **ARG**: Defines a variable that users can pass at build-time to the builder with the **docker build** command.
 - **ONBUILD**: Adds a trigger instruction when the image is used as the base for another build.
-

5. Docker Compose Commands:

- `docker-compose up`: Create and start containers.
 - `docker-compose down`: Stop and remove containers, networks, images, and volumes.
 - `docker-compose build`: Build or rebuild services.
 - `docker-compose logs`: View output from containers.
 - `docker-compose restart`: Restart services.
 - `docker-compose pause`: Pause services.
 - `docker-compose unpause`: Unpause services.
 - `docker-compose start`: Starts existing containers for a service.
 - `docker-compose stop`: Stops running containers without removing them.
 - `docker-compose config`: Validates and views the compose file.
-

6. Docker Networking:

- `docker network ls`: List networks.
- `docker network create network`: Create a network.
- `docker network rm network`: Remove a network.
- `docker network inspect network`: Display detailed information on one or more networks.

- **Bridge:**

Docker's default networking driver. If you don't specify a driver, this is the type of network you are creating.

- **Host:**

For standalone containers, removes network isolation between the container and the Docker host.

- **Overlay:**

Networks connect multiple Docker daemons together and enable swarm services to communicate with each other.

- **Macvlan:**

Assigns a MAC address to a container, making it appear as a physical device on your network.

7. Docker Volumes:

- `docker volume ls`: List volumes.
 - `docker volume create volume`: Create a volume.
 - `docker volume rm volume`: Remove a volume.
 - `docker volume inspect volume`: Display detailed information on one or more volumes.
-

8. Docker Object Commands:

- `docker image`: Manages images.
 - `docker container`: Manages containers.
 - `docker network`: Manages networks.
 - `docker volume`: Manages volumes.
 - `docker secret`: Manages Docker secrets.
 - `docker plugin`: Manages plugins.
-

9. Docker Advanced Commands:

- `docker history image`: Show the history of an image.
 - `docker save image > file`: Save an image to a tar archive.
 - `docker load < file`: Load an image from a tar archive.
 - `docker commit container image`: Create a new image from a container's changes.
-

10. Docker System Commands:

- 44. `docker info`: Displays system-wide information.
 - 45. `docker version`: Shows the Docker version information.
 - 46. `docker system df`: Shows Docker disk usage.
 - 47. `docker system events`: Gets real-time events from the server.
 - 48. `docker system prune`: Removes unused data.
-

11. Docker Swarm Commands:

- `docker swarm init`: Initialize a swarm.
- `docker swarm join`: Join a node to a swarm.
- `docker node ls`: List nodes in a swarm.
- `docker service create image`: Create a service.
- `docker service ls`: List services in a swarm.
- `docker service rm service`: Remove a service.
- `docker swarm`: Manages Swarm.
- `docker node`: Manages Swarm nodes.
- `docker stack`: Manages Docker stacks.
- `docker service`: Manages services.

11.1. Container Orchestration with Docker Swarm:

- **Services:**

The definition of the tasks to `execute` on the manager `or` worker nodes.

- **Tasks:**

A single runnable `instance` of a service.

- **Worker nodes:**

Nodes that receive and execute tasks dispatched from manager nodes.

- **Manager nodes:**

The only nodes that can `execute` Docker commands, `or` authorize other nodes to join the swarm.

- **Raft Consensus Algorithm:**

Manager nodes use the Raft Consensus Algorithm to agree on task scheduling and status updates.

- **Services scaling:**

In Docker Swarm mode you can scale your services up or down for optimal resource utilization.

12. Docker Security:

- `docker secret create secret file`: Create a secret from a file.
- `docker secret ls`: List secrets.
- `docker secret rm secret`: Remove a secret.

- **Docker Security Scanning:**

A security feature that you can use in Docker repositories.

- **Docker Content Trust:**

Provides the ability to use digital signatures for data sent to and received from remote Docker registries.

- **Docker Secrets:**

Allows you to manage sensitive data, such as passwords, SSH private keys, SSL certificates, and other data.

13. Docker Troubleshooting and Monitoring:

- `docker stats`: Display a live stream of container(s) resource usage statistics.
- `docker system df`: Display the space usage of Docker daemon entities.
- `docker inspect`: Return low-level information on Docker objects.
- `docker events`: Get real time events from the server.
- `docker logs`: Fetch the logs of a container.

- **docker healthcheck**: Checks the health of a running container.
-

14. Docker Registries and Repositories:

- **Docker Hub:**

Docker's public registry instance.

- **Docker Trusted Registry (DTR):**

Docker's commercially supported storage for Docker images.

- **Docker Content Trust (DCT):**

Provides the ability to use digital signatures for data sent to and received from remote Docker registries.

15. Docker and CI/CD:

- **Docker in Jenkins:**

Jenkins provides built-in Docker integration for CI/CD workflows.

- **Docker in Travis CI:**

Travis CI also provides Docker integration for CI/CD workflows.

- **Docker in GitLab CI:**

GitLab CI has native Docker support for CI/CD workflows.

- **Docker in CircleCI:**

CircleCI offers Docker support to build and push Docker images.

- **Docker in Azure DevOps:**

Azure DevOps can build, push, or run Docker images, or run a Docker command.

16. Docker and the Cloud:

- **Docker on AWS:**

AWS provides services like Amazon Elastic Container [Service](#) (ECS) and AWS Fargate for running Docker containers.

- **Docker on Azure:**

Azure provides Azure Kubernetes [Service](#) (AKS) for running Docker containers.

- **Docker on Google Cloud:**

Google Cloud provides Google Kubernetes Engine (GKE) for [running](#) Docker containers.

17. Docker Best Practices:

- **Container immutability:**

The idea that you never update a running container, instead, you should always [create a new one](#).

- **Single process per container:**

Each container should [address](#) a single concern and do it well.

- **Minimize layer counts in Dockerfiles:**

The fewer commands that create layers, the smaller your image is likely to be.

- **Leverage build cache:**

Docker will cache the results of the first build of a Dockerfile allowing subsequent builds to be super fast.

- **Use .dockerignore:**

Prevents sending unnecessary [files to the daemon](#) when building images.

- **Use specific tags for production images:**

Using specific versions of an image ensures that your [application](#) consistently works as expected.

- **Always use the latest version of Docker:**

Each new version of Docker includes security improvements, bug fixes, and new features.

18. Docker and Microservices:

- **Service discovery:**

Docker Swarm Mode has a built-in DNS server that other containers can use to resolve the service name to an IP address.

- **Service scaling:**

In Docker Swarm Mode you can scale your services up or down.

- **Load balancing:**

Docker has a built-in load balancer that can distribute network connections to all instances of a replicated service.

- **Secure communication between services:**

Docker Swarm Mode has a built-in routing mesh that provides secure communication between services.

19. Docker Plugins:

- **Storage Plugins:**

These plugins provide storage capabilities to Docker containers.

- **Network Plugins:**

These plugins provide networking capabilities to Docker containers.

- **Authorization Plugins:**

These plugins restrict the Docker APIs that can be accessed.

20. Docker API:

- **Docker REST API:**

An API used by applications to interact with the Docker daemon.

- **Docker SDK:**

SDKs for Go and Python, built on top of the Docker REST API.

- **Docker Engine API:**

The API Docker clients use to communicate with the Docker daemon.

21. Docker Editions:

- **Docker Community Edition (CE):**

Ideal for individual developers and small teams looking to get started with Docker and experimenting with container-based apps.

- **Docker Enterprise Edition (EE):**

Designed for enterprise development and IT teams who build, ship, and run business-critical applications in production at scale.

22. Docker Architecture:

- **Docker Engine:**

A client-server application with three major components: a server, a REST API, and a command-line interface (CLI).

- **Docker Daemon:**

Listens for Docker API requests and manages Docker objects such as images, containers, networks, and volumes.

- **Docker Client:**

The primary way that many Docker users interact with Docker. When you use commands such as `docker run`, the client sends these commands to `dockerd`, which carries them out.

- **Docker Images:**

The basis of containers. An Image is an ordered collection of root filesystem changes and the corresponding execution parameters for use within a container runtime.

- **Docker Containers:**

A runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI.

- **Docker Services:**

Allows you to scale containers across multiple Docker daemons, which all work together as a swarm with multiple managers and workers.