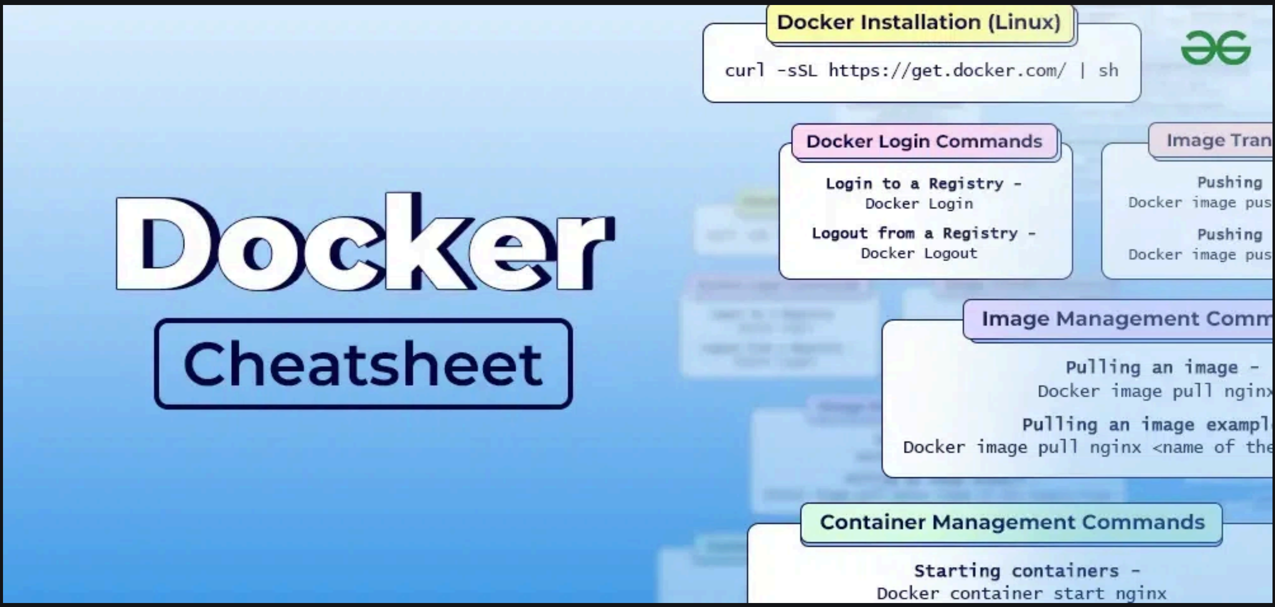
Docker Cheat Sheet : Complete Guide (2024)

Last Updated : 08 Oct, 2024

Docker is a very popular tool introduced to make it easier for developers to create, deploy, and run applications using containers. A container is a utility provided by Docker to package and run an application in a loosely isolated environment. Containers are lightweight and contain everything needed to run an application, such as libraries and other dependencies packed by the developer during the application's packaging process. This assures developers that their application can be run on any other machine. Here, we're going to provide you with an ultimate **Docker Cheat Sheet** that will help you to **learn Docker Commands** easily.

This **Docker command cheatsheet** is a summary of **commonly used Docker commands** and their options, as well as other useful information related to Docker. It covers all the important commands required for Docker operations, including Docker installation, building, running, shipping, and cleaning up, as well as interaction with containers. This cheat sheet will be a handy reference for you to perform various tasks with Docker, such as installing, building, running, shipping, and cleaning up containers. This **Docker cheat sheet** is useful for both – DevOps freshers who're learning Docker and experienced Docker users who need to recall a specific command or option but may not remember all the details.



```
curl -sSL https://get.docker.com/ | sh
```

Login to a Registry -
Docker Login
Logout from a Registry -
Docker Logout

Pushing
Docker image pus
Pushing
Docker image pus

Pulling an image -
Docker image pull nginx
Pulling an image exampl
Docker image pull nginx <name of the

Starting containers -
Docker container start nginx

What is Docker?

Docker allows you to collect and run an application in a container, which is a loosely isolated environment. Because of the isolation and security, you can run multiple containers on a single host at the same time. Containers are lightweight and include everything required to run the application, eliminating the need to rely on what is already installed on the host. You may easily share containers while working, and you can ensure that everyone with whom you share gets the same container that functions in the same way.

> ***Pre-requisite****: Docker, DockerHub*
>
> *The below Docker cheat sheet contains commands to manage the docker containers, images, network, volumes, building running, and deploying containers and also covered commands related to Docker Compose.*

Docker Cheat Sheet – Table of Content

Table of Content

A **Docker cheat sheet** is a handy tool for quick reference, but mastering Docker requires hands-on practice. For an in-depth guide to using Docker in DevOps, the **DevOps Engineering – Planning to Production** course provides detailed instructions on Docker's full range of capabilities.

Docker Commands Cheat Sheet

The Docker cheat sheet will help you as a reference guide from where you can quickly read of mostly used common commands of Docker. The cheat sheet will help as a handy guide for developers and other system administrations who are working with Docker. Let's get started:

Installation Commands

Name	Command
Installation on Linux	curl -sSL https://gcurl -fsSL https://get.docker.com -o get-docker.sh && sudo sh get-docker.sh

Docker Login Commands

Name	Command
Log in to a Registry	docker login
Logout from a Registry	docker logout

Image Management Commands

Docker images are self-contained software packages that contain all the necessary components to run an application. These components include the code, runtime, system tools, system libraries, and settings. Docker images are lightweight and easy to use.

Name	Command
Build an image	docker build -t <image_name>
Pulling an Image	docker image pull nginx
Pulling an Image Example	docker image pull <Name of The Image>:<Tag>

Image Transfer Commands

Name	Command
Pushing an Image	docker image push <usernameofregistry:Imagename:tag>
Pushing an Image Example	docker image push eon01/nginx localhost:5000/myadmin/nginx

Docker Hub Commands

Docker Hub is a service provided by Docker for finding and sharing container images with your team. Learn more and find images at "***https://hub.docker.com***".

Name	Command
Login into Docker	-docker login -u <username>
Publish an image to Docker Hub	-docker push <username>/<image_name>
Search Hub for an image	-docker search <image_name>
Pull an image from a Docker Hub	-docker pull <image_name>

General Docker Commands

Name	Command
Start the docker daemon	docker -d
Get help with Docker. Can also use –help on all subcommands	docker –help
Display system-wide information	docker info

Containers Management Commands

CONTAINERS

A docker image's runtime instance is referred to as a container. The container remains consistent regardless of the infrastructure in use. This isolation of software from its environment guarantees uniformity in function, even in cases where there are discrepancies between development and staging.

Name	Command
Starting Containers	docker container start nginx
Stopping Containers	docker container stop nginx
Restarting Containers	docker container restart nginx
Pausing Containers	docker container pause nginx
Unpausing Containers	docker container unpause nginx
Blocking a Container	docker container wait nginx
Sending SIGKILL Containers	docker container kill nginx
Sending another signal	docker container kill -s HUP nginx
Connecting to an Existing Container	docker container attach nginx
Check the Containers	docker ps
To see all running containers	docker container ls
Container Logs	docker logs infinite
‘tail -f’ Containers’ Logs	docker container logs infinite -f
Inspecting Containers	docker container inspect infinite

Name	Command
Inspecting Containers for certain	docker container inspect –format '{{ .NetworkSettings.IPAddress }}' $(docker ps -q)
Containers Events	docker system events infinite
docker system events infinite	docker container port infinite
Running Processes	docker container top infinite
Container Resource Usage	docker container stats infinite
Inspecting changes to files or directories on a container's filesystem	docker container diff infinite

Docker Image Management Commands

Name	Command
Listing Images	docker image ls
Building Images	docker build.
From a Remote GIT Repository	docker build github.com/creack/docker-firefox
Instead of Specifying a Context, You Can Pass a Single Dockerfile in the URL or Pipe the File in via STDIN	docker build – < Dockerfile

Name	Command
Building and Tagging	docker build -t eon/infinite.
Building a Dockerfile while Specifying the Build Context	docker build -f myOtherDockerfile.
Building from a Remote Dockerfile URI	curl example.com/remote/Dockerfile | docker build -f – .
Removing an Image	docker image rm nginx
Loading a Tarred Repository from a File or the Standard Input Stream	docker image load < ubuntu.tar.gz
Saving an Image to a Tar Archive	docker image save busybox > ubuntu.tar
Showing the History of an Image	docker image history
Creating an Image From a Container	docker container commit nginx
Tagging an Image	docker image tag nginx eon01/nginx
Pushing an Image	docker image push eon01/nginx

Docker Network Commands

Name	Command
Creating an Overlay Network	docker network create -d overlay MyOverlayNetwork

Name	Command
Creating a Bridge Network	docker network create -d bridge MyBridgeNetwork
Creating a Customized Overlay Network	docker network create -d overlay \ –subnet=192.168.0.0/16 \ –subnet=192.170.0.0/16 \ –gateway=192.168.0.100 \ –gateway=192.170.0.100 \ –ip-range=192.168.1.0/24 \ –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" \ MyOverlayNetwork
Removing a Network	docker network rm MyOverlayNetwork
Listing Networks	docker network ls
Getting Information About a Network	docker network inspect MyOverlayNetwork
Connecting a Running Container to a Network	docker network connect MyOverlayNetwork nginx
Connecting a Container to a Network When it Starts	docker container run -it -d –network=MyOverlayNetwork nginx
Disconnecting a Container from a Network	docker network disconnect MyOverlayNetwork nginx

Docker Exposing Ports Commands

Name	Command
Exposing Ports	EXPOSE <port_number>
Mapping Ports	docker run -p $HOST_PORT:$CONTAINER_PORT –name <container_name> -t <image>

Docker Commands Removing Containers, Images, Volumes, And Networks

Name	Command
Removing a Running Container	docker container rm nginx
Removing a Container and its Volume	docker container rm -v nginx
Removing all Exited Containers	docker container rm $(docker container ls -a -f status=exited -q)
Removing All Stopped Containers	docker container rm \`docker container ls -a -q\`
Removing a Docker Image	docker image rm nginx
Removing Dangling Images	docker image rm $(docker image ls -f dangling=true -q)
Removing all Images	docker image rm $(docker image ls -a -q)

Name	Command
Removing all Untagged Images	docker image rm -f $(docker image ls | grep “^” | awk “{print $3}”)
Stopping & Removing all Containers	docker container stop $(docker container ls -a -q) && docker container rm $(docker container ls -a -q)
Removing Dangling Volumes	docker volume rm $(docker volume ls -f dangling=true -q)
Removing all unused (containers, images, networks and volumes)	docker system prune -f
Clean all	docker system prune -a

Docker Swarm Commands

Name	Command
Installing Docker Swarm	curl -ssl https://get.docker.com | bash
Initializing the Swarm	docker swarm init –advertise-addr 192.168.10.1
Getting a Worker to Join the Swarm	docker swarm join-token worker
Getting a Manager to Join the Swarm	docker swarm join-token manager
Listing Services	docker service ls

Name	Command
Listing nodes	docker node ls
Creating a Service	docker service create –name vote -p 8080:80 instavote/vote
Listing Swarm Tasks	docker service ps
Scaling a Service	docker service scale vote=3
Updating a Service	docker service update –image instavote/vote:movies vote
Updating a Service	docker service update –force –update-parallelism 1 –update-delay 30s nginx

Docker file Commands

Command	Description	Example
FROM	Specifies the base image for the build	FROM ubuntu:latest
RUN	Executes a command inside the container during build time	RUN apt-get update && apt-get install -y curl
CMD	Specifies the default command to run when the container starts	CMD [“npm”, “start”]
EXPOSE	Informs Docker that the container listens on	EXPOSE 80/tcp

Command	Description	Example
	specific network ports at runtime	
ENV	Sets environment variables inside the container	ENV NODE_ENV=production
COPY	Copies files or directories from the build context into the container	COPY app.js /usr/src/app/
ADD	Similar to COPY but supports additional features like URL retrieval and decompression	ADD https://example.com/file.tar.gz /usr/src/
WORKDIR	Sets the working directory for subsequent instructions	WORKDIR /usr/src/app
ARG	Defines variables that users can pass at build-time to the builder with the docker build command	ARG VERSION=1.0
ENTRYPOINT	Configures a container to run as an executable	ENTRYPOINT [“python”, “app.py”]
VOLUME	Creates a mount point and assigns it to a specified volume	VOLUME /data

Command	Description	Example
USER	Sets the user or UID to use when running the image	USER appuser
LABEL	Adds metadata to an image in the form of key-value pairs	LABEL version=”1.0" maintainer=”John Doe
ONBUILD	Configures commands to run when the image is used as the base for another build	ONBUILD ADD . /app/src

Docker Volume Commands

Command	Description	Example
volume create	Creates a named volume	docker volume create mydata
volume ls	Lists the available volumes	docker volume ls
volume inspect	Displays detailed information about a volume	docker volume inspect mydata
volume rm	Removes one or more volumes	docker volume rm mydata
volume prune	Removes all unused volumes	docker volume prune

Docker CP commands

Command	Description	Example
docker cp [OPTIONS] SRC_PATH CONTAINER:DEST_PATH	Copies files or directories from the local filesystem to the specified container	docker cp myfile.txt mycontainer:/usr/src/app/
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH	Copies files or directories from the specified container to the local filesystem	docker cp mycontainer:/usr/src/app/result.txt /tmp/result/

Docker Security Commands (Docker Scout)

Command	Description	Example
docker scout compare	[experimental] Compare two images and display differences	docker scout compare image1:tag image2:tag
docker scout cves	Display CVEs identified in a software artifact	docker scout cves image: tag
docker scout Quickview	Quick overview of an image	docker scout quickview image: tag

Command	Description	Example
docker scout recommendations	Display available base image updates and remediation recommendations	docker scout recommendations image:tag
docker scout version	Show Docker Scout version information	docker scout version

Conclusion

In conclusion, this **Docker cheat sheet** helps you with a **quick revision of all the Docker commands** that are required for Docker operations, including Docker installation, building, running, shipping, and cleaning up, as well as interaction with containers.

Docker CheatSheet – FAQs

1. What is the architecture of Docker?

Answer:

Docker follows a client-server architecture. The Docker client communicates with the Docker daemon, which is responsible for building, running, and managing Docker containers. The client and daemon can run on the same host, or the client can connect to a remote daemon.

2. Which language is Docker built on?

Answer:

Docker is built using Go programming language because of its advantage of several features of the Linux kernel to deliver its functionality.

3. Does Docker require coding?

Answer:

No, Docker does not require any prior coding knowledge. It is a containerization platform that enables developers to package, deploy, and run applications using containers.

4. Are Docker secrets safe?

Answer:

You can use Docker secrets to centrally manage this data and securely transmit it to only those containers that need access to it. Secrets are encrypted during transit and at rest in a Docker swarm.

5. How many types of volumes are there in Docker?

Answer:

Docker supports three types of volumes:

*a) **Named Volumes:** These are volumes with a user-defined name that can be used across multiple containers.*

*b) **Bind Mounts:** These are directories on the host machine that are mounted into a container, allowing direct access to the host's file system.*

*c) **tmpfs Mounts:** These are volumes stored in the host's memory, allowing fast read and write operations but with limited size and durability.*

6. What is the flag in Docker?

Answer:

In Docker, a flag is a command-line option that modifies the behavior of a Docker command. Flags are used to provide additional instructions or parameters to Docker commands, allowing you to customize the execution according to your needs.

7. Why is Docker used in DevOps?

Answer:

Docker is widely used in DevOps practices due to its ability to create reproducible and portable environments. With Docker, developers can package their applications and dependencies into containers, ensuring consistent behavior across different stages of the software development lifecycle. Docker also facilitates the automation of deployment, testing, and scaling processes, enabling faster and more reliable software delivery in DevOps pipelines.