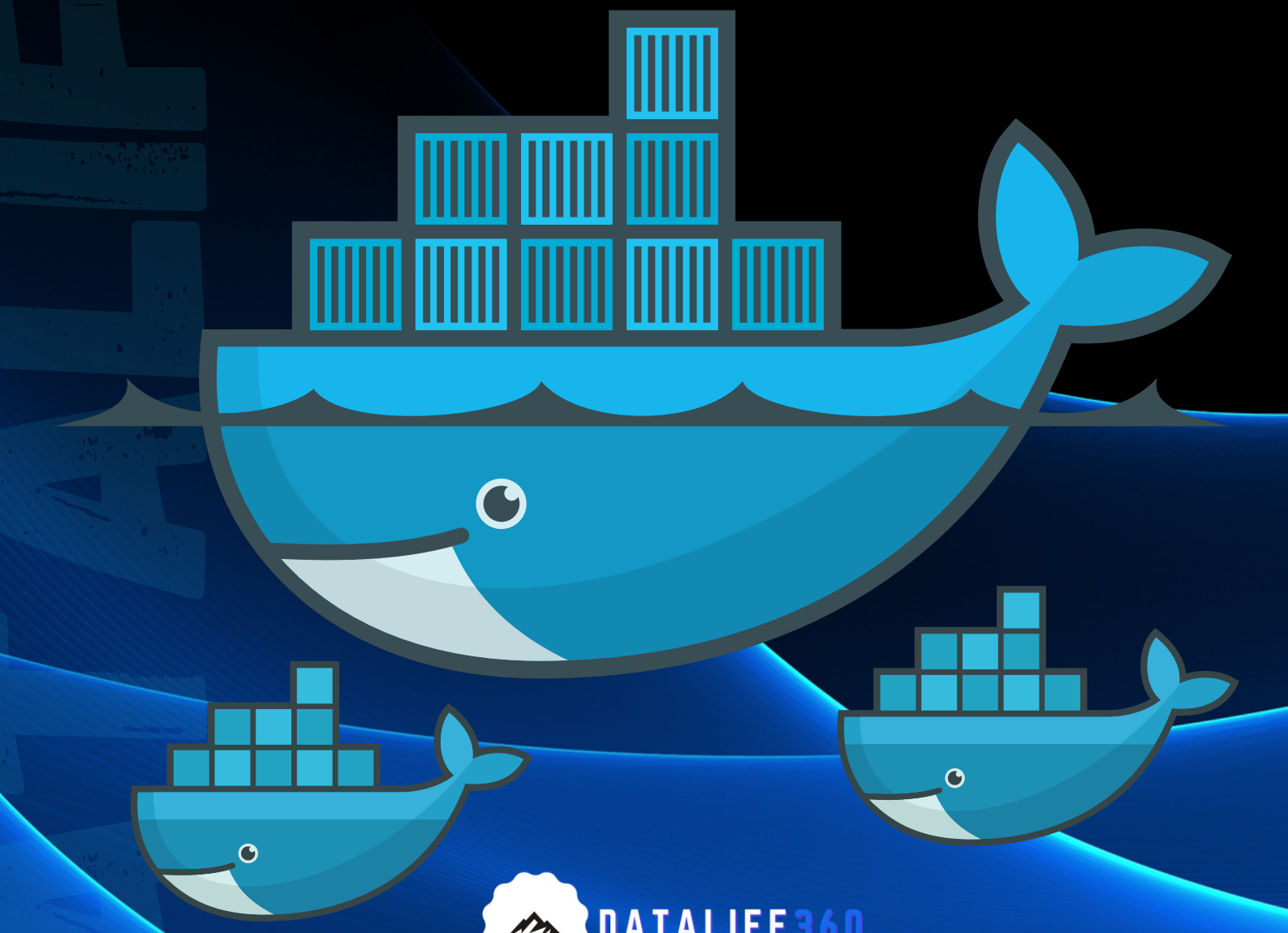
DOCKER

Beginner Cheat Sheet



DATALIFE360

INTRODUCTION

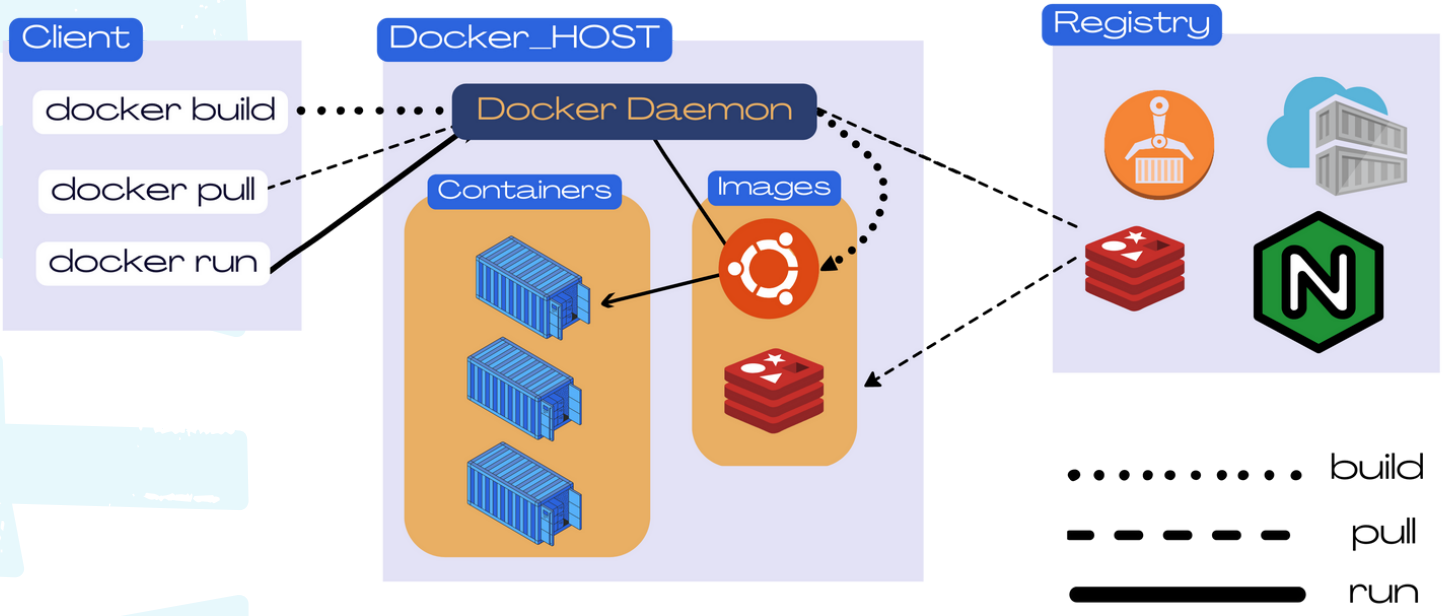
Why Docker?

Containers package your application and what you need to run it in a container image.
You can put a base operating system, libraries, files and folders, environment variables, etc.

A container image is a template for container execution. Multiple containers can run from the same image - all sharing the same behavior. This improves scaling and distribution of the application. Images can be stored in a remote registry to ease the distribution.

Docker is also useful for machine learning (ML) models because it allows for reproducibility, versioning, isolation, and portability. You can create a consistent and reproducible environment for your ML models, version them, and deploy them across different environments.

Container Architecture



DOCKERFILE

Docker File Instruction Arguments

Dockerfiles are written in a specific syntax. The order of the commands is also important. Some commands are optional, and others may be used multiple times within a single Dockerfile, depending on your requirements.

COMMAND	MEANING	EXAMPLE
FROM	specifies the base image for the container	FROM ubuntu:20.04
RUN	runs a command in the container to install software or make other changes	RUN pip install --upgrade -r requirements.txt
COPY	copies files or directories from the host machine to the container	COPY . /app
ENV	Sets environment variables in the container	ENV NGINX_PORT=80
EXPOSE	specifies the ports that the container listens on	EXPOSE 80
CMD	specifies the command that should be run when the container starts	CMD ["python", "train.py"]

DOCKERFILE

Docker File Instruction Arguments

COMMAND	MEANING	EXAMPLE
ENTRYPOINT	command that should be run when the container starts, with the CMD being passed as arguments.	ENTRYPOINT ["nginx"]
ADD	used to copy files from the host file system into a container	ADD my_model.pkl /app/models/ ADD requirements.txt /app/
LABEL	adds metadata to the image. Can be used to note versions	LABEL maintainer="Tony Stark <tstark@example.com>"
USER	sets the user or UID that runs the commands in the container	USER 1001
WORKDIR	sets the working directory for the container	WORKDIR /app

DOCKERFILE

Docker Command Order

1. FROM: Specify the base image for the container

2. RUN: Execute commands to build and configure the image

3. COPY or ADD: Copy files from the host file system into the container

4. ENV: Set environment variables

5. EXPOSE: Declare the ports that the container will listen on

6. CMD: Specify the command to run when the container starts

7. LABEL: add metadata to the image

8. ENTRYPOINT: configure a default command or script that will run whenever a container is created from the image

Commands are can be important since they are executed in the order they appear in the file. Some commands are executed during the build time, while others are executed at runtime.

REGISTRY & REPO COMMANDS

To execute docker commands there are several ways:

Command Line Interface
You can open a terminal on your system and type the command directly in the command prompt.

Docker Compose
Use the docker-compose command-line tool to define and run multi-container applications using a YAML file

Docker API
Interact with the Docker daemon using the Docker API, which allows you to control Docker using a programmatic interface.

Docker GUI Tools
hPortainer, Kitematic, and Docker Desktop are examples. They provide a user interface for managing and interacting with Docker.

Script or Shell File
Create a script or shell file that contains a series of Docker commands, and then execute the script by running it in the terminal.

REGISTRY & REPO COMMANDS

Login to a Registry

```
docker login
```

```
docker login --username=yourusername --password=yourpassword
```

Logout from a Registry

```
docker logout
```

```
docker logout registry.hub.docker.com
```

Searching for a Image

```
docker search mysql
```

Pulling a Image

```
docker pull mysql
```

Push a Image

```
docker push yourusername/myimage
```

CONTAINERS

Build Image

Build an Image

```
docker build -t my-image .
```

Build Image from Github Repo

```
docker build --t helloworld github.com/folder/hello-world .
```

Running a Container

Run Container

```
docker run my-image
```

Run Container, Open a Port at 6969

```
docker run -p 6969:6969 my-image
```

Renaming a Container

```
docker rename old_container_name new_container_name
```

Removing a Container

Remove One Container

```
docker rm container_name
```

Remove One Container

```
docker rm container1 container2
```

Note:
You can use either name or container id

CONTAINERS

Start a Container

```
docker start my-container
```

Stop a Container

```
docker stop my-container
```

Restart a Container

```
docker restart my-container
```

Restart a Container

```
docker pause my-container
```

Restart a Container

```
docker unpause my-container
```

Block a Container

```
docker wait my-container
```

GETTING CONTAINER INFO

List Containers

List Running Containers

```
docker ps
```

List All Containers (Stopped and Running)

```
docker ps -a
```

Container Logs

Show Docker Logs

```
docker logs my-container
```

Show Detailed Docker Logs

```
docker logs --details my-container
```

Inspect Containers

```
docker inspect my-container
```

Find Current Ports for a Container

```
docker port my-container
```

DATALIFE360

Check Us Out on YouTube, Substack, and LinkedIn!

#datalife360