Antony Kervazo-Canut

Docker for Teenagers

SUITABLE FOR ADULTS

Installation and verification

```
# Debian Installation
sudo apt install docker.io

# Red Hat Installation (Fedora, CentOS, etc.)
sudo yum install docker

# MacOS Installation (via Homebrew)
brew install docker

# Windows Installation (via Chocolatey)
choco install docker-desktop
```

```
# Version
docker --version
```

```
# Starting on Linux
sudo systemctl start docker

# On MacOS or Windows, launch the Docker Desktop application
```

DockerFile

Exemple : nginx

A Dockerfile is a scripted configuration used to create Docker container images. It uses Docker-specific commands like FROM, RUN, COPY, and EXPOSE to define the steps for building an image. In the context of Nginx, a typical Dockerfile would include installing Nginx on a base Linux image, exposing the necessary ports, and configuring commands to start the web server.

Tips for Nginx:

- Lightweight Base Image: Prefer lightweight images as a base to reduce the Docker image size.
- Cleanup After Installation: Clean up temporary files after installation to avoid an oversized image.
- Minimization of Layers: Group RUN commands to reduce the number of layers in the image.

DockerFile

Exemple : nginx

```dockerfile
# Use a base image to create the new image. Example: Ubuntu, Alpine, etc.
FROM ubuntu:latest

# Sets an environment variable used in the Dockerfile
ENV APP_HOME /app

# Sets the working directory in the container
WORKDIR $APP_HOME

# Copies files or directories from the local build context to the image
COPY . $APP_HOME

# Executes commands to install software or configure the image
RUN apt-get update && apt-get install -y nginx

# Informs Docker that the container listens on the specified ports at runtime
EXPOSE 80

# Sets the default command to run when the container starts
# If you need your container to run as an executable,
# use ENTRYPOINT instead of CMD.
CMD ["nginx", "-g", "daemon off;"]

# Creates a mount point to attach a volume
VOLUME /var/www/html
```

DockerFile

Exemple : build .Net App

The Dockerfile for a .NET application establishes a uniform environment by installing the .NET dependencies, copying the source files, and specifying the command to run the application. Additionally, it is important to note that with Docker, it is possible to compile virtually any type of application, not just .NET applications, thanks to the flexibility and variety of images available.

Tips for .NET Applications:

- Multi-Stage Builds: Opt for multi-stage builds to reduce the size of the final image.
- Specific Versions: Use precise versions for base images and dependencies for security and compatibility.
- Optimization of Layers: Organize instructions to maximize the efficiency of Docker layer caching and speed up the build. These tips aim to optimize the creation of Docker images, whether for a web server like Nginx or for a complex application like a .NET application, ensuring efficiency, security, and performance.

DockerFile

Exemple : build .Net App


```dockerfile
# Use the .NET SDK image to build the application
FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build-env

WORKDIR /app

# Copies the csproj file and restores dependencies
COPY *.csproj ./
RUN dotnet restore

# Copies the other project files and builds the application
COPY . ./
RUN dotnet publish -c Release -o out

# Use the .NET runtime image to run the application
FROM mcr.microsoft.com/dotnet/aspnet:5.0
WORKDIR /app
COPY --from=build-env /app/out .

# Exposes the port on which the application will run
EXPOSE 80

# Sets the command to start the application
CMD ["dotnet", "YourApplication.dll"]
```

CLI

Command-line interfaces (CLI) for Docker are essential for interacting with Docker, providing detailed and precise control over the management of containers, images, networks, and volumes. Although less visual than graphical user interfaces (GUI), CLIs offer superior flexibility and power for experienced users. Advantages of Docker CLI:

- Precise Control: Allows for detailed and specific operations on containers and images.
- Automation: Facilitates the automation of Docker tasks through scripts.
- Use in Headless Environment: Ideal for servers without a graphical interface.
- Integration with DevOps Tools: Integrates well with CI/CD tools, versioning, and other development tools.

Key Points:

- Power and Flexibility: Offer a wide range of commands for all Docker functions.
- Learning: Require learning to master but offer a deep understanding of Docker.
- Preferred by Developers and DevOps: Often the choice of professionals due to their efficiency and integrability into complex workflows. In summary, CLIs for Docker are a powerful tool for advanced users, allowing for precise and efficient management of Docker environments, while providing the necessary capabilities for automation and integration into broader development and operations systems.

Container management


```
# Create and start a container from an image.
# docker run [options] [image_name]
# -d: detached mode
# -p [port]: map a container port to a host port
docker run -d -p 80:80 nginx

# Displays the list of active containers with details like
ID, image, status, ports, etc.
docker ps

# Stop a running container.
# docker stop [container_ID_or_name]
docker stop mycontainer

# Restart a stopped container.
# docker restart [container_ID_or_name]
docker restart mycontainer

# Remove a specific container.
# docker rm [container_ID_or_name]
docker rm mycontainer
```

Image management

```
# Build an image from a Dockerfile.
# docker build -t [image_name] [directory_path]
# -t: names the built image.
docker build -t myimage .

# Displays the list of images with details like ID, size,
tag, etc.
docker images

# Remove a specific Docker image.
# docker rmi [image_ID_or_name]
docker rmi myimage

# Detailed list of the image layers and their changes.
# docker history [image_name]
docker history nginx
```

Network management

```
# Displays the list of networks with details like ID, name,
driver, etc.
docker network ls

# Create a new Docker network.
# docker network create [options] [network_name]
# --driver: specifies the network type, such as bridge,
overlay, none, etc.
docker network create --driver bridge mynetwork

# Connect a container to a Docker network.
# docker network connect [network_name] [container_name]
docker network connect mynetwork mycontainer

# Displays detailed information about the network, like
connected containers, options, IP, etc.
# docker network inspect [network_name]
docker network inspect mynetwork
```

Data management


```
# Create a new Docker volume for persistent storage.
# docker volume create [volume_name]
docker volume create myvolume

# Displays the list of volumes with details like name,
driver, etc.
docker volume ls

# Attach a volume to a container for persistent storage.
# docker run -v [volume_name]:[path_in_container]
[image_name]
# -v: attaches the named volume to a specific path in the
container.
docker run -v myvolume:/data nginx

# Remove a specific Docker volume.
# docker volume rm [volume_name]
docker volume rm myvolume
```

Monitoring and logs


```
# Affiche les logs de sortie du conteneur spécifié.
# docker logs [ID_ou_nom_du_conteneur]
docker logs monconteneur

# Fournit un aperçu en temps réel de l'utilisation des
ressources pour tous les conteneurs en cours d'exécution.
docker stats

# Ouvre un shell interactif dans le conteneur, si bash est
utilisé comme commande.
# docker exec [options] [ID_ou_nom_du_conteneur] [commande]
# -it : attache une session interactive TTY.
docker exec -it monconteneur bash

# Affiche un JSON détaillé avec des informations complètes
sur le conteneur, y compris l'état du réseau, les volumes
montés,
# les paramètres de configuration, etc.
# docker inspect [ID_ou_nom_du_conteneur]
docker inspect monconteneur
```

Debugging


```
# Displays a detailed JSON with comprehensive information
about the container, including network state, mounted
volumes,
# configuration settings, etc.
# docker inspect [container_ID_or_name]
docker inspect mycontainer

# Saves a container to a new image. Useful for creating an
image from a modified container.
# docker commit [container_ID_or_name] [image_name]
docker commit mycontainer myimage

# Copy files or folders to and from a container. Very useful
for data transfer.
# To copy from container to host:
# docker cp [container_ID_or_name]:[path_in_container]
[local_path]
docker cp mycontainer:/file.txt /my/local/folder

# To copy from host to container:
# docker cp [local_path] [container_ID_or_name]:
[path_in_container]
docker cp /my/local/folder mycontainer:/file.txt

# Lists the files and folders that have been added, modified,
or deleted in the container.
# docker diff [container_ID_or_name]
docker diff mycontainer
```

Cleaning

```
# Clean up unused resources such as stopped containers,
unused networks, and cached images.
docker system prune

# Remove stopped containers, unused volumes, unused networks,
# and unused images (including cached images).
docker system prune -a

# Remove all unused images, not just dangling images.
docker image prune -a

# Remove all stopped containers.
docker container prune

# Remove all unused volumes.
docker volume prune
```

Registry management

Exemple : DockerHub


```
# Log in to DockerHub to enable pushing and pulling images.
# docker login -u [username] -p [password]
docker login -u myusername -p mypassword
# docker login -u [username] -p [access_token]
docker login -u myusername -p
9f86d081884c7d659a2feaa0c55ad015a

# Download an image from DockerHub to the local system.
# docker pull [image_name]
docker pull ubuntu

# Push a local image to DockerHub.
# docker push [username/image_name]
docker push myuser/myimage

# Search for images on DockerHub.
# docker search [search_term]
docker search mysql
```

Docker Compose

Docker Compose is a Docker tool used to define and manage multi-container applications. It relies on a YAML file to configure the application's services, which simplifies the deployment and management of interdependent containers.
Advantages of Docker Compose:

- Simple Configuration: Uses a YAML file for clear and easy configuration.
- Integrated Management: Launches and manages multiple containers as a single grouped service.
- Environment Isolation: Ensures consistency across development, testing, and production environments.

Usage: Docker Compose is ideal for local development, automated tests, and the production deployment of applications composed of multiple interconnected services. In short, Docker Compose is essential for easily managing complex applications on Docker, offering a structured and efficient approach to orchestrating multiple services.
In the following example, the docker-compose.yaml file allows launching two containers simultaneously on ports 8080 (which maps to port 80 of the container) and port 1433 while passing some environment variables.

Docker Compose

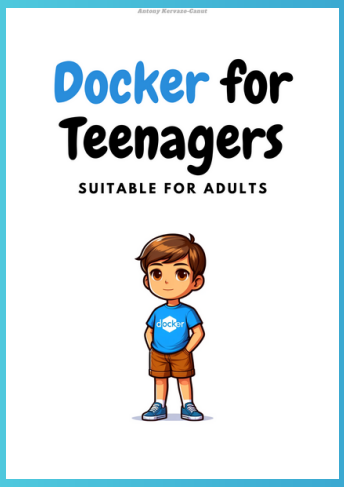
```yaml
version: '3.8'

services:
  webapp:
    image: mynetapp:latest
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8000:80"
    environment:
      -
"ConnectionStrings:DefaultConnection=Server=db;Database=MyDb;
User=sa;Password=Your_password123;"
    depends_on:
      - db

  db:
    image: mcr.microsoft.com/mssql/server:2019-latest
    environment:
      SA_PASSWORD: "Your_password123"
      ACCEPT_EULA: "Y"
    ports:
      - "1433:1433"
```

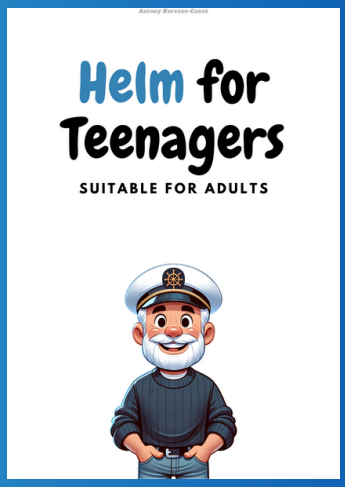
Antony Kervazo-Canut

Management of Docker Compose

```
# Start the services specified in the docker-compose.yml file
in the background.
docker compose up -d

# Stop and remove resources (containers, networks, volumes)
created.
docker compose down

# Display running Compose projects with their state.
docker compose ls

# Display logs for all services.
docker compose logs

# Execute a one-time command in a service.
# docker compose run [service_name] [command]
docker compose run webapp echo "Hello World"

# Update and rebuild a specific service.
# docker compose up -d --no-deps [service_name]
docker compose up -d --no-deps webapp

# Stop a specific service.
# docker compose stop [service_name]
docker compose stop webapp
```

In the same collection





ANTONYCANUT

ANTONY KERVAZO-CANUT