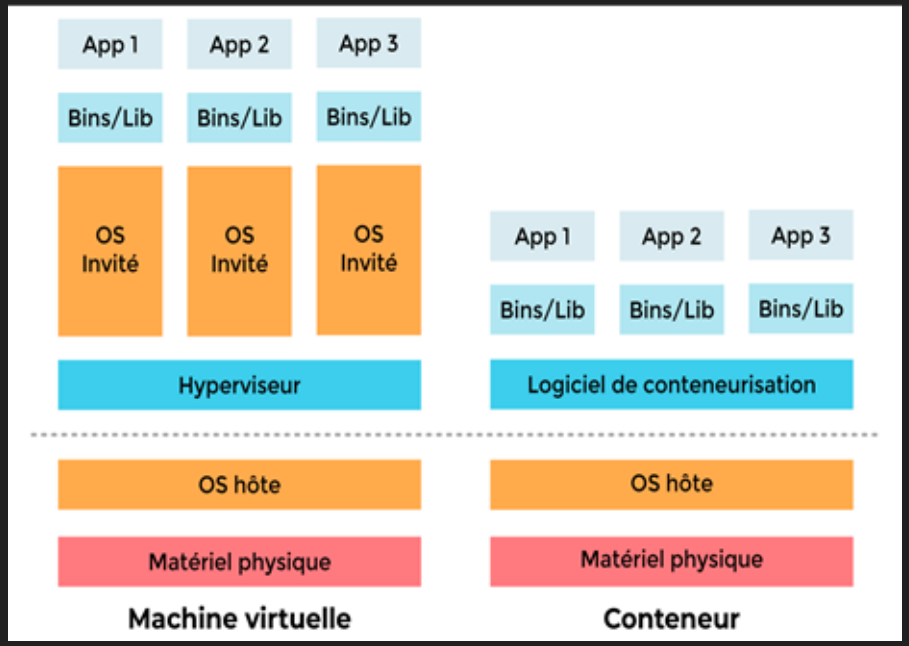
Docker and Container Creation

- **What is Docker?**
  - An open-source platform for developing, shipping, and running applications in containers.
- **Benefits of Docker?**
  - Isolation, portability, version control, and efficient resource usage.

Docker vs. Virtual Machines

- **Docker**
  - Containers share the host system's kernel.
  - Lightweight as they don't need a full OS for each instance.
- **Virtual Machines**
  - Each VM runs its own full copy of an operating system.
  - Heavier resource usage due to the OS overhead.

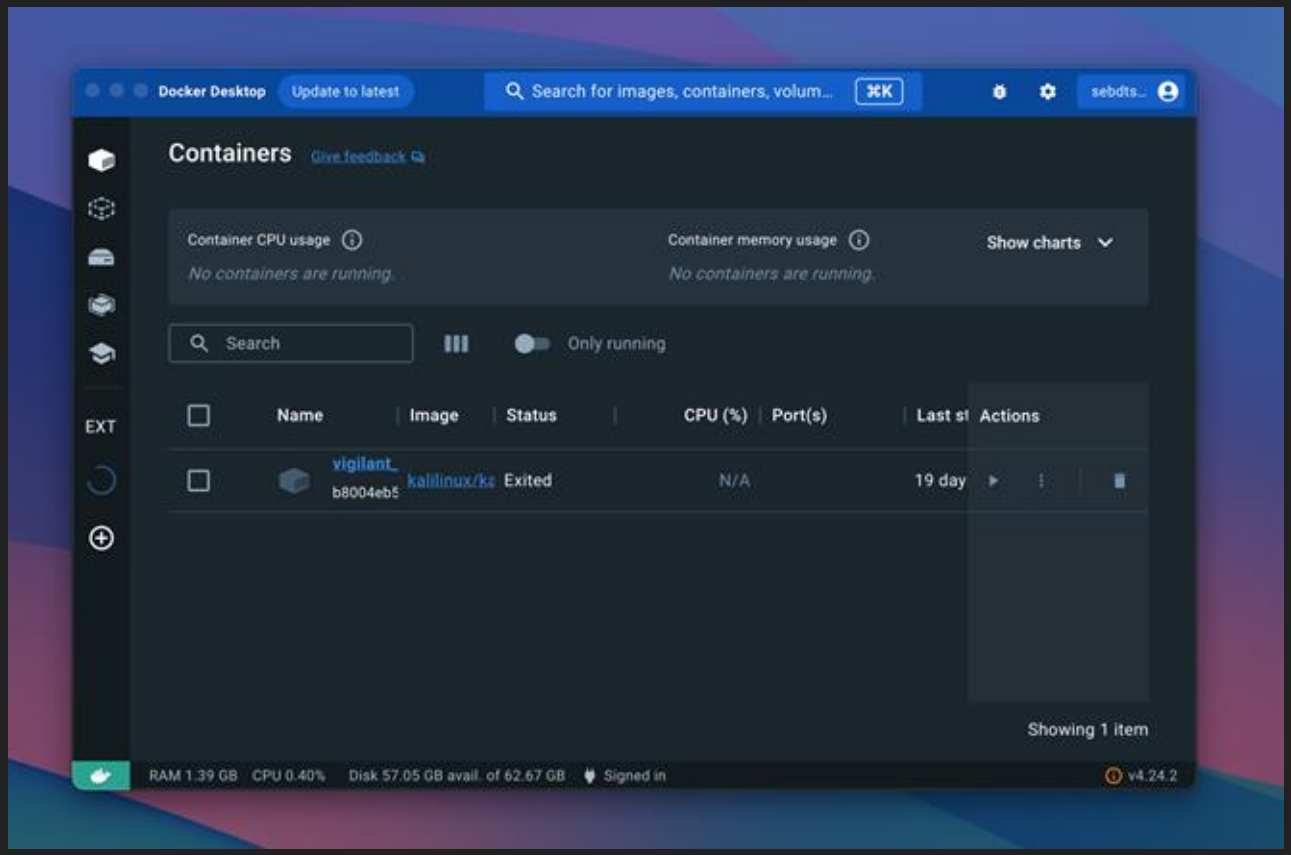


Installing Docker

- **Install Docker**
  - Download:
  - https://www.docker.com/products/docker-desktop/
  - Docs :
  - https://docs.docker.com/get-docker/

Installing Docker

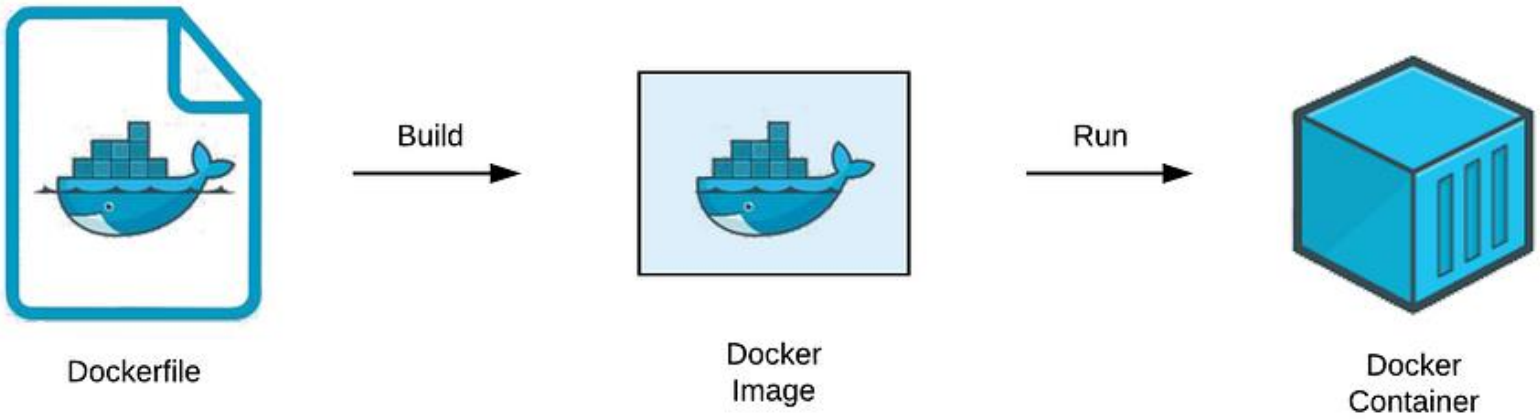
- **Open and register:**



Basic Concepts of Docker

- **Dockerfile :** This is a text script containing a sequence of instructions and commands used to create a Docker image.
- **Docker Images** : Read-only templates for creating containers.
- **Docker Containers** : Executable instances of images.
- **Docker Hub** : Public registry to share Docker images.

Docker Steps



Microsoft Doc :
https://learn.microsoft.com/fr-fr/dotnet/architecture/microservices/docker-application-development-process/docker-app-development-workflow

Dockerfile

- **Writing a Dockerfile** :
  - FROM:
    - Specifies the base image from which you are building
    - `FROM ubuntu:latest or FROM python:3.8-slim`
  - RUN:
    - Executes commands in a new layer on top of the current image and commits the results
    - `RUN apt-get update && apt-get install -y git`
  - CMD:
    - Provides default commands and arguments for an executing container.
    - Only the last CMD instruction will take effect
    - `CMD ["python", "./app.py"]`

Dockerfile

- ENV:
  - Sets environment variables within the image
  - `ENV PATH /app/bin:$PATH`
- ADD and COPY:
  - ADD copies files, directories, or remote file URLs from your host to your image's filesystem
  - COPY is more straightforward and is used to copy local files and directories into the image
  - `COPY . /app`
- WORKDIR:
  - Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD instructions that follow in the Dockerfile
  - `WORKDIR /app`

Dockerfile: Exemple

projet — micro Dockerfile — 122×27

```
# Use the latest Ubuntu image as the base
FROM ubuntu:latest

# Update Ubuntu Software repository
RUN apt-get update

# Install Python 3
RUN apt-get install -y python3

# Optional: Install pip for Python 3
RUN apt-get install -y python3-pip

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

# Optional: Copy the current directory contents into the container at /usr/src/app
COPY . .

# Optional: If you have a requirements.txt file, uncomment the following line to install Python dependencies
RUN pip3 install --no-cache-dir -r requirements.txt

# The command the container will execute by default when it starts
CMD ["python3"]
```

Dockerfile + (24,1) | ft:dockerfile | unix | utf-8 Alt-g: bindings, Ctrl-g: helpp

Docker Image

- **Building an Image :**

```
docker build -t myimage .
```

- ‘`docker build`’ is used to build an image from a Dockerfile

- The ‘`-t`’ tags your image, making it easier to find and use. ‘myimage’ is the tag you assign to your image. You can also include a version number (e.g., myimage:v1)

- ‘`.`’ indicates the location of the Dockerfile and the context of the build. Here, the current directory

Exemple: Add Git Repository

- Clone the Repository :
  - Add a command to clone the repository in your Dockerfile.

```
RUN git clone https://github.com/user/repo.git
/local/path.
```

- Rebuild the Image :
  - Rebuild your Docker image to include these changes.

```
docker build -t myimagegit .
```

Running Docker Containers

- Runs a container from the image myimage

```
docker run -d --name -p 8000:80 mycontainer myimage
```

  - ‘`-d`’: Stands for "detached mode", which means the container runs in the background and does not block the terminal or shell.
  - ‘`--name mycontainer`’: Assigns the name mycontainer to the new container. This is useful for identifying and managing the container later.
  - ‘`-p 8000:80`’: Maps port 80 inside the container to port 8000 on the host machine.
    - Port docs : https://docs.docker.com/network/

Connect to a Running Docker Container

```
docker exec -it [container_name_or_id] /bin/bash
```

- ‘`docker exec`’: Execute a command inside a running Docker container.
- ‘`-it`’: -i "interactive" allowing interaction with the container. -t providing you with a text-based terminal inside the container.
- `[container_name_or_id]`: ID of your Docker container. You can find the container's ID by using the docker ps command.
- ‘`/bin/bash`’: This launches the bash shell inside the container.

Docker Container Management

- Lists all running containers:

```
docker ps
```

- Stops a running container:

```
docker stop <container_id>
```

- Removes a container:

```
docker rm <container_id>
```

- Docker command line:

https://docs.docker.com/engine/reference/commandline/docker/

Working with Docker Hub

- Docker Hub is a cloud-based service provided by Docker, Inc. It acts as a centralized resource for container image discovery, distribution, and change management.

- docker pull:
  - Purpose: Pulls an image or a repository from Docker Hub or another registry.

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
```

- docker push:
  - Purpose: Pushes an image or a repository to Docker Hub or another registry.

```
docker push [OPTIONS] NAME[:TAG]
```

Docker Best Practices and Vigilance Points

- **Regularly Update and Rebuild Images**
  - Description: Rebuild your Docker images frequently to incorporate updates and security patches. Always rebuild after modifying source code or dependencies.
  - Why It's Important: Prevents vulnerabilities, ensures up-to-date dependencies, and maintains consistency.

- **Minimize Image Size**
  - Description: Optimize Dockerfiles to create smaller images. Use multi-stage builds, avoid including unnecessary files, and choose lightweight base images.
  - Why It's Important: Smaller images are faster to build, transfer, and deploy. They also reduce the attack surface for security threats.

- **Manage Secrets Securely**
  - Description: Avoid hardcoding secrets like passwords or API keys in Dockerfiles or image layers. Use environment variables, Docker Secrets, or external secrets management tools.
  - Why It's Important: Protects sensitive information and maintains the security of your applications and infrastructure.

- **Use Specific Base Image Tags**
  - Description: Instead of using the 'latest' tag for base images in Dockerfiles, specify a particular version.
  - Why It's Important: Ensures consistent and predictable builds. Using 'latest' can lead to unexpected changes if the base image is updated.