Docker

Sergio Paniego Blanco
@sergiopaniego
sergiopaniegoblanco@gmail.com
https://sergiopaniego.github.io/

Docker

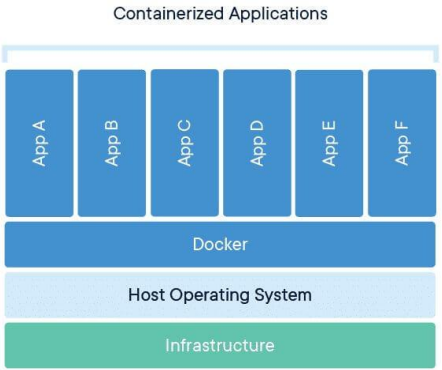
- Introduction to Docker
- Basic concepts of Docker
- Building a Docker Image for a Simple Python Application
- Docker compose for managing multi-service applications.
- Hands-On: dockerizing a simple Django application
- Best practices and image optimization.
- Conclusion and next steps

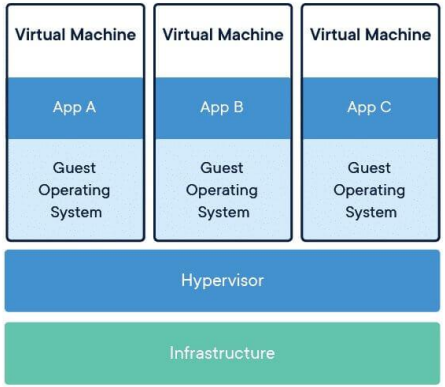
Introduction to Docker

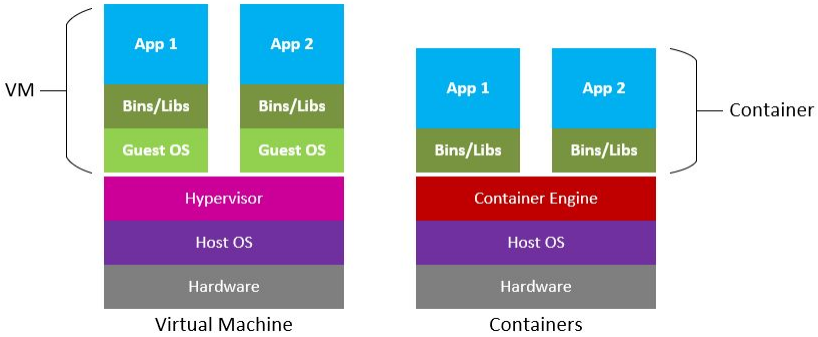
- What is Docker?
  - Docker is a platform that enables developers to create, deploy and run applications in isolated environments → containers.
  - Containers include everything needed to run the application (code, runtime, system tools, libraries) but are more lightweight than virtual machines.
  - Critical tool for modern development pipelines due to its efficiency and ease of use.

Introduction to Docker

- Containers vs. virtual machines (VMs)
  - **VMs:** include a guest OS for each instance, taking up more resources.
  - **Containers:** share the host OS kernel, which makes them more lightweight and efficient.
- Containers provide an isolated environment similar to VMs but require less memory and boot up faster.







Introduction to Docker

- Benefits of Docker in development and deployment:
  - **Consistency Across Environments:** Containers ensure that the application runs the same way in development, staging, and production.
  - **Isolation:** Applications and their dependencies are encapsulated, so one container's changes don't affect others.
  - **Portability:** Docker containers can run on any system with Docker installed, making it easy to move applications across environments.
  - **Efficiency:** Containers use fewer resources than VMs, reducing overhead costs.

Introduction to Docker

- Let's explore Docker Hub (https://hub.docker.com/)
  - Source for pre-built images.
  - Search for popular images like `python` or `nginx`.
  - Explore the tags for the image.
  - *How would using an image from Docker Hub benefit a developer who wants a specific version of Python for their project?*

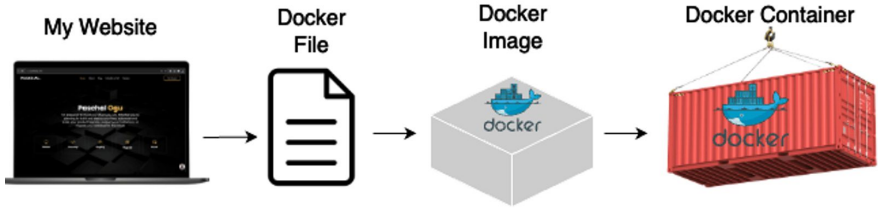
Introduction to Docker

- Let’s install docker
  - https://docs.docker.com/engine/install/
- Check that docker is correctly installed:

```
docker --version
docker run hello-world
```

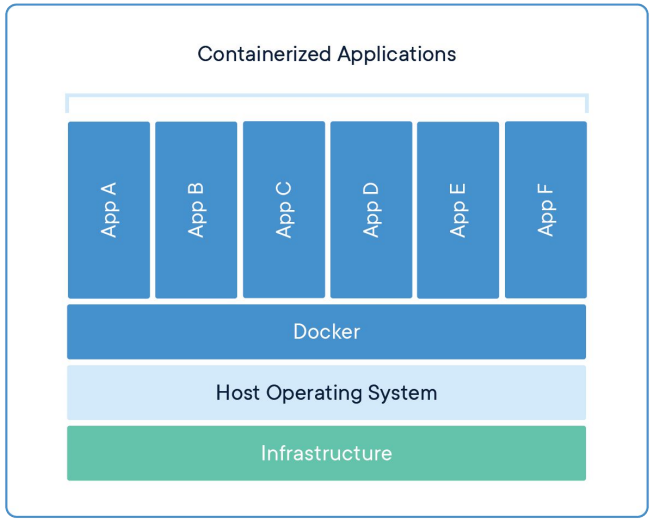
Docker basics

- Docker image
  - Lightweight, standalone, and executable software package that includes everything needed to run a piece of software: code, runtime, libraries, environment variables, and configuration files.
  - Images are static files and can be shared through registries (like Docker Hub)



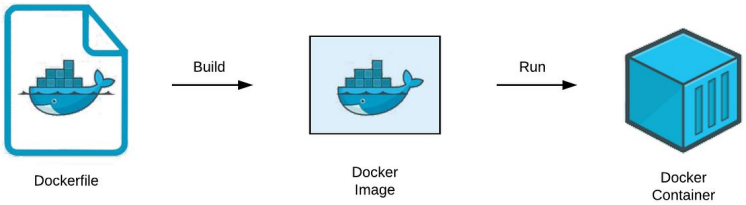
Docker basics

- Docker container
  - Runtime instance of an image. It’s a running environment isolated from the host system, ensuring that applications run the same way regardless of where they are deployed.
  - Containers are created from images and can be started, stopped, and deleted without affecting the host.



Docker basics

- Dockerfile
  - Simple text file that contains a series of instructions to build a Docker image.
  - Common instructions: FROM (base image), COPY (copy files into the image), and RUN (execute commands in the image during build
  - Docker builds images step-by-step based on the Dockerfile instructions



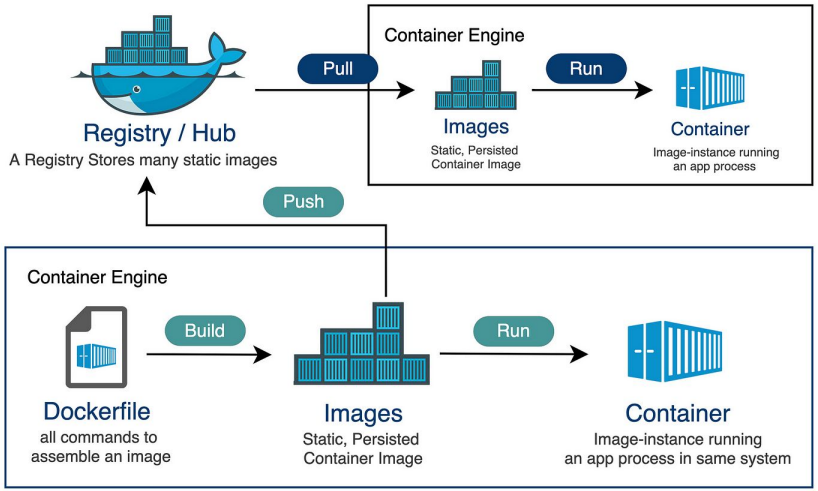
Dockerfile U

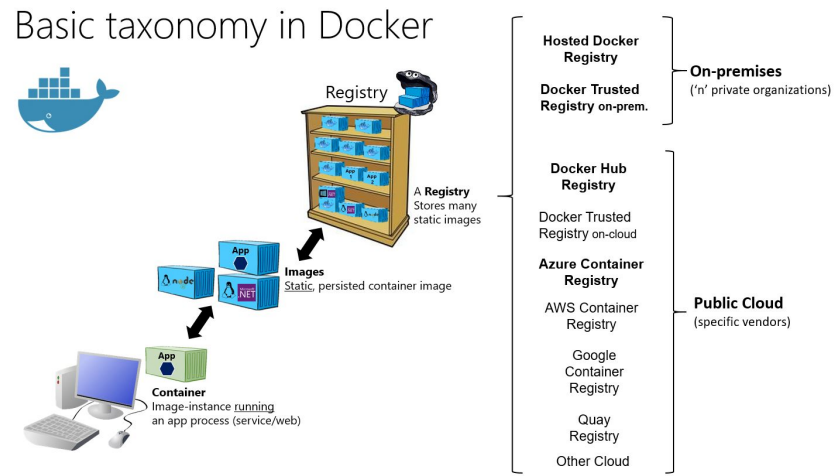
Dockerfile > ...

```
1  FROM node:14-alpine3.16
2
3  WORKDIR /app
4
5  COPY . .
6
7  RUN npm install
8
9  CMD [ "npm", "start" ]
```

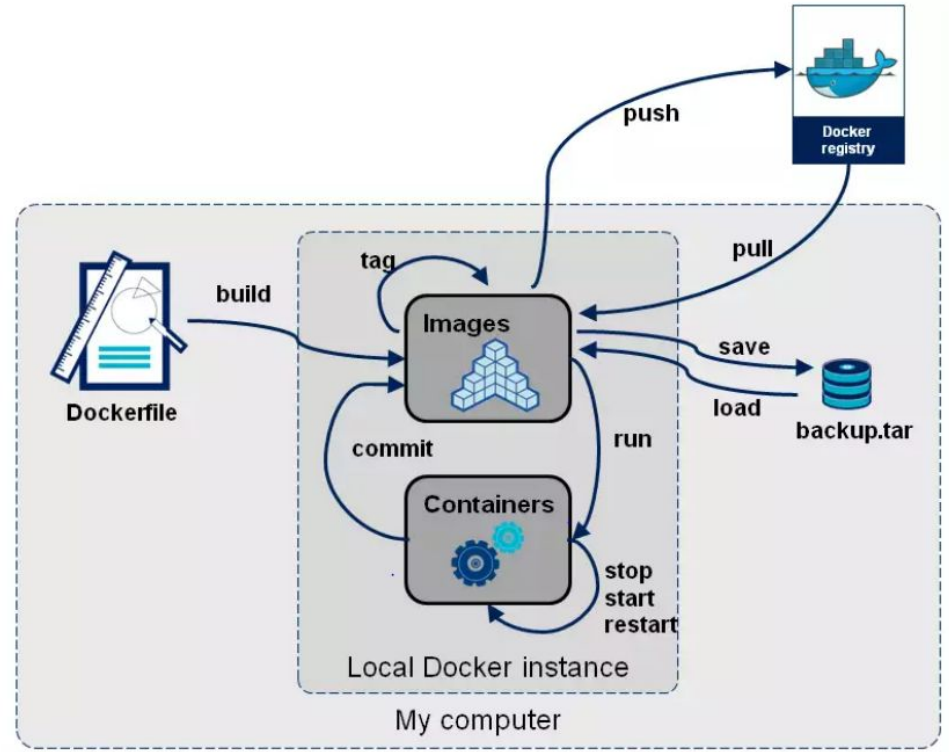
Docker basics

- Image registry (Docker Hub)
  - A registry is a storage and distribution system for Docker images. Docker Hub is the default public registry where users can find official and user-contributed images.
  - Images are versioned with tags, so users can pull a specific version of an image (e.g., `python:3.9`).





Docker basics



Docker

Docker basics

- Essential Docker Commands
  - `docker pull`: Downloads an image from a Docker registry (e.g., Docker Hub).

```
docker pull hello-world
```

  - `docker run`: Creates and starts a container from an image. This command can also execute specific commands within the container. It will pull the image if it isn't available locally, create a new container from it, and run the default command for that image.

```
docker run hello-world
docker run python:3.9 python -c "print('Hello from Docker!')"
```

Docker basics

- Essential Docker Commands
  - `docker ps`: lists all running containers.

```
docker ps
```

    - Let's check it running: docker run -d python:3.9 sleep 60 and then docker ps.

```
docker run -d python:3.9 sleep 60
docker ps
```

    - `-d` is for running in detached mode so the container continues running in the background.

Docker basics

- Essential Docker Commands
  - `docker stop`: stops a running container.

```
docker stop <container_id>
```

    - Let's check it by first identifying the container ID using `docker ps` and then running `docker stop <container_id>`

```
docker ps
docker stop <container_id>
```

Docker basics

- Essential Docker Commands
  - `docker rm`: delete a stopped container

```
docker rm <container_id>
```

    - Let's try it out.
  - `docker rmi`: removes an image from the local machine

```
docker rmi python:3.9
```

    - Let's remove the hello-world image. Images can only be removed if no containers depend on them.

```
docker rmi hello-world
```

ACTIVITY

- Working with and nginx container:
  - Pull the nginx image.
  - Run the nginx container in detached mode exposing port 8080. `-p`
  - Verify that it runs.
  - Check nginx in the browser
  - Stop the nginx container
  - Remove the container

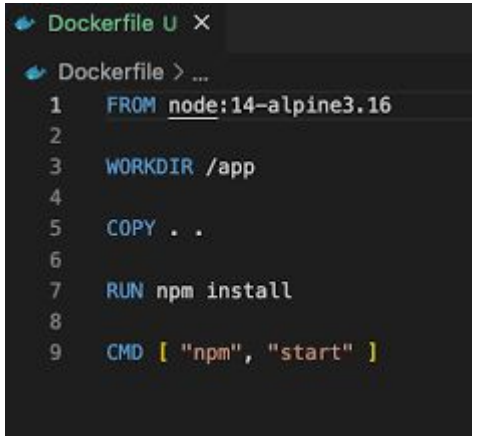
ACTIVITY

- Working with an nginx container

```
docker pull nginx
docker run -d -p 8080:80 nginx
docker ps
# Open a browser and navigate to http://localhost:8080
docker stop <container_id>
docker rm <container_id>
docker rmi nginx
```

Building a Docker image

- To build a Docker image, we need to write a Dockerfile.
  - Dockerfile: text file containing instructions to build a Docker image.
  - Each line in the Dockerfile represents a step in the image-building process, and these instructions allow Docker to replicate the application environment consistently.



Building a Docker image

- Key Dockerfile instructions:
  - `FROM`: Specifies the base image from which the image will be built. Common images for Python applications include `python:3.9` or `alpine` for a lighter version

```
FROM python:3.9
```

  - `COPY`: copies files from the host machine into the container's file system. Commonly used to add application code or configuration files to the container.

```
COPY app.py /app/app.py
```

Building a Docker image

- Key Dockerfile instructions:
  - RUN: executes commands during the image build process. Often used to install dependencies or prepare the environment.

```
RUN pip install requests
```

  - CMD: specifies the command that should be run when a container is started. Unlike RUN, which is executed during the build, CMD defines the container's default behavior.

```
CMD ["python", "/app/app.py"]
```

  - EXPOSE: indicates the network port that the container will use. It doesn't actually publish the port but serves as documentation for what ports should be exposed.

```
EXPOSE 8080
```

Building a Docker image

- Let's see how to build it with a practical example, a Python calculator app.
- We need to write:
  - `calculator.py`
  - `Dockerfile`
- Once that is completed, we can build the Docker image doing:

```
docker build -t cli-calculator .
```

- `-t` for adding a tag name and `.` specifies the current directory as build context.

Building a Docker image

- Let’s run the dockerized calculator: let0s run a container from the newly build image to start the calculator app.
  - `-it` makes the container interactive so we can input numbers and operations

```
docker run -it cli-calculator
```

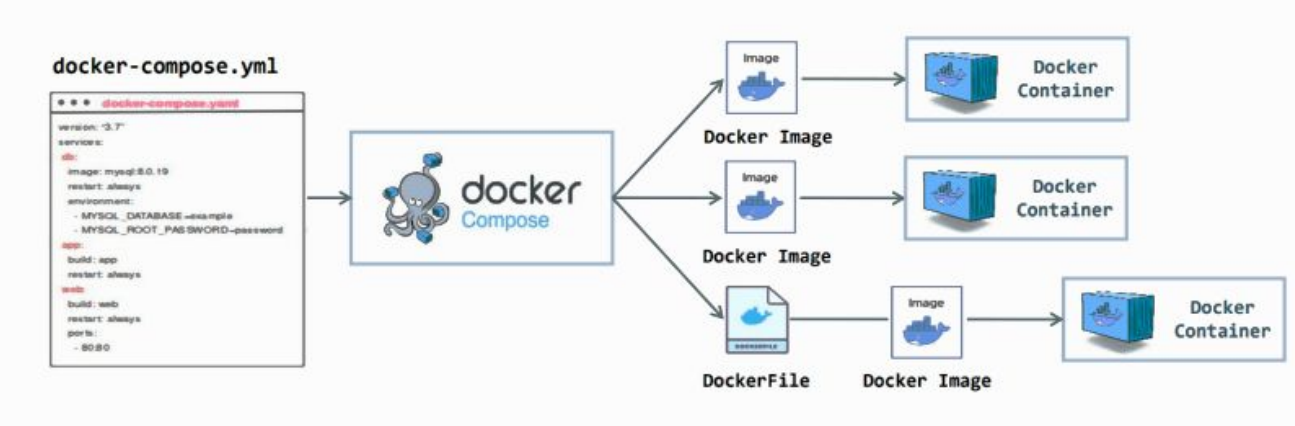
ACTIVITY

- Write a Python script that performs temperature conversions, then build a Docker image to containerize it.

```
docker build -t temp-converter .
docker run -it temp-converter
```

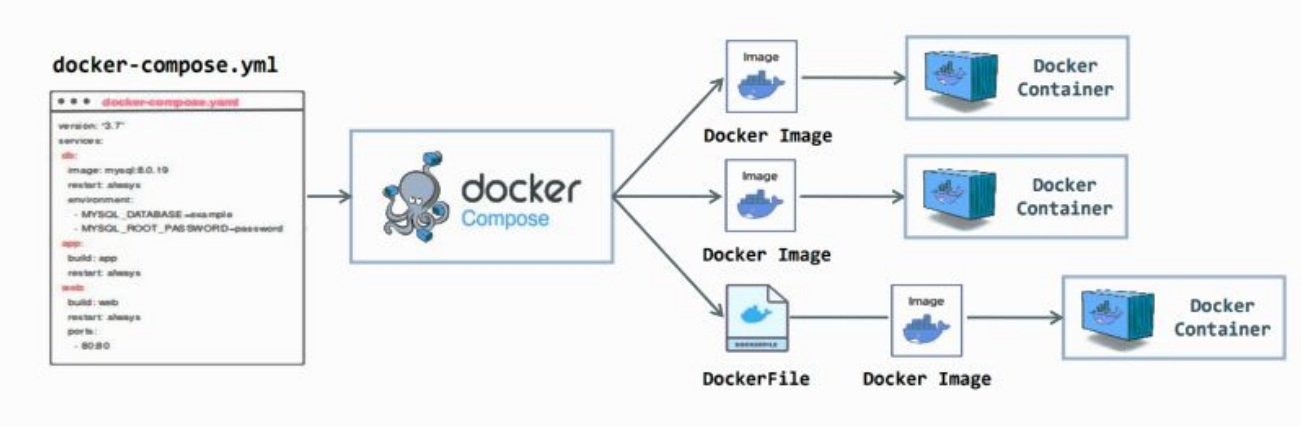
Docker compose for managing

- Docker compose: tool for defining and running multi-container Docker applications.
- Allows users to define an application stack with all its services in a single `docker-compose.yml` file



Docker compose for managing

- Benefits of using Docker compose
  - Simplifies the management of multiple services (e.g., web applications, databases).
  - Facilitates services orchestration, allowing for easy scaling, updates, and deployment.
  - Streamlines the development workflow by ensuring all services run in isolated environments.



Docker compose for managing

- Key commands:
  - `docker-compose up`: builds and starts the containers as defined in the `docker-compose.yml`.
    - `-d` to run in detached mode (background).
    - `--build` to force a rebuild of the images before starting the containers. Useful if you have made changes to the Dockerfile or dependencies.
  - `docker-compose down`: stops and removes the containers defined in the `docker-compose.yml`.
    - `-- volumes` removes also associated volumes.
  - `docker-compose logs`: displays logs from the services defined in the `docker-compose.yml`. Useful for debugging and monitoring.

Docker compose for managing

- Let's user docker compose to set up a simple REST API using Flask and PostgreSQL database.

```
docker-compose up --build
```

- Check that the application is up accessing http://localhost:5001

ACTIVITY

- Add new routes to the application and apply the changes. For example:
  - Add a `/greet` route

```
docker-compose up --build
```

Dockerizing a simple Django application

- Let’s create a Dockerized Django application
  - Setup a new Django project

```
mkdir django-docker-app
cd django-docker-app
django-admin startproject myproject .
```

Dockerizing a simple Django application

- Let's create a Dockerized Django application
  - Create a Dockerfile
  - Create a requirements file
  - Create a Docker compose file
  - Build the application, start it and run database migrations
  - We can access the app at http://localhost:8000/

```
cd myproject
docker-compose up --build
docker-compose exec web python manage.py
migrate
```

Dockerizing a simple Django application

- Let’s create a Dockerized Django application with a view
  - Setup a new Django project

```
mkdir django-docker-app
cd django-docker-app
mkdir myapp
touch myapp/Dockerfile
touch docker-compose.yml
touch myapp/requirements.txt
```

Dockerizing a simple Django application

- Let's create a Dockerized Django application with a view
  - Create a new Django project

```
docker-compose run web django-admin startproject config .
```

  - Create a new Django app

```
docker-compose run web python manage.py startapp main
```

  - Update Django config/setting.py adding new 'main' app and `ALLOWED_HOSTS = [*]`

Dockerizing a simple Django application

- Let's create a Dockerized Django application with a view
  - Create a simple HTML view
    - Define a view in `main/views.py`
    - Create a URL route for the view in `main/urls.py`
    - INclude the app's URLs in the project's `urls.py` in `config/urls.py`
    - Create a templates directory inside main and add `home.html` template.
    - Start django server and test http://localhost:8000/

```
docker-compose up
```

ACTIVITY

- Add a new view to the application and check that it works correctly.

Best practices and image optimization

- Optimizing Docker images is essential to ensure efficient, secure and quick to deploy apps.
- Possible optimizations:
  - Use lightweight base images: reduce the final image size, speed up build time and reduce the amount of storage and bandwidth needed.
  - Example: instead of python:3.9, use python 3.9-slim

```
# Use a lightweight base image
FROM python:3.9-slim
```

Best practices and image optimization

- Best practices for layer management in Dockerfile:
  - Combine commands

```
RUN apt-get update
RUN apt-get install -y libpq-dev
# instead use:
RUN apt-get update && apt-get install -y libpq-dev
```

  - Order instructions carefully: place frequently changing instructions at the end.
- Use `.dockerignore` to exclude unnecessary files.

Conclusion and next steps

- Further learning suggestions:
  - **Kubernetes:** Docker is just one part of the container ecosystem. Kubernetes, a container orchestration platform, is essential for managing multiple Docker containers across distributed environments.