Introduction to Docker Compose

- **What is Docker Compose?**
  - A tool for defining and running multi-container applications.
  - Uses YAML files for configuration.

- **Why Use Docker Compose?**
  - Simplifies and automates the deployment process.
  - Ensures consistency across different environments. (dev & prod)
  - Managing dependencies between services

Docker Docs :
https://docs.docker.com/desktop/

Usages

- Application development and testing
- Deployment of small applications in production (Microservices)
- Continuous Integration / Continuous Deployment (CI/CD)
- Local Development Environment

YML

- **Version** :

  Specifies the Docker Compose version used

- **Services** :

  The main section, where you define the various containers (services) that make up your application.

- image : The name of the Docker image to be used.
- container_name : The name of the Dockerfile to be used.
- build : Path to a directory containing a Dockerfile, if you want to build your own image.
- ports : The ports to be exposed, usually in the form "port_host:port_container".

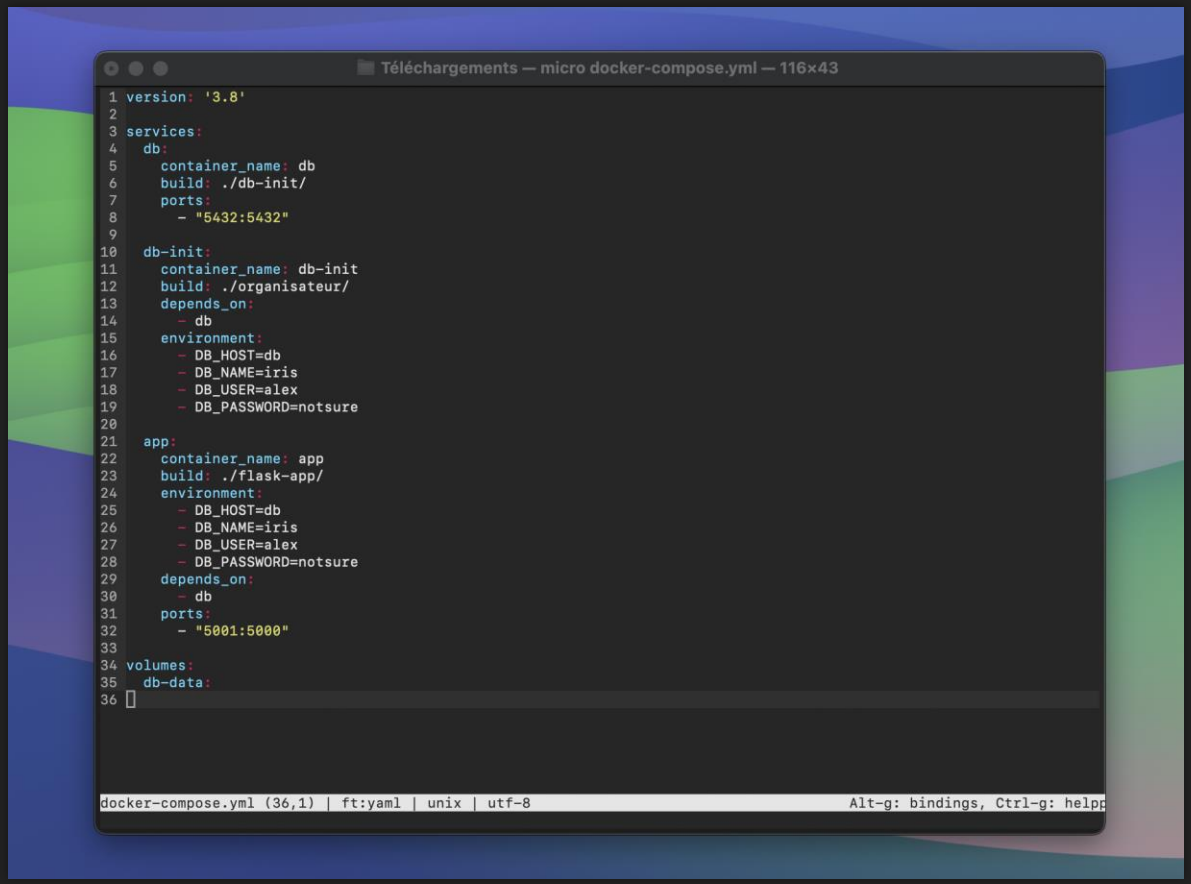
YML

- volumes : Volume mounts for data persistence or file sharing.
- environment : Environment variables to be defined in the container.
- depends_on : Dependencies between services, indicating the order in which services are started.
- command : Command to be executed on container startup.

- **Networks** : Defines custom networks for communication between services.

- **Volumes** : Define volumes for data persistence outside containers.

YML expemple :



Basics commands

- Starts services defined in the Docker Compose file

```
docker-compose up
```

- Stops and deletes resources created by up

```
docker-compose down
```

Advanced management

- **Volume types**
- Named volumes offer better management of persistent data. They are managed by Docker and separated from host data.

```
services:
    db:
        image: postgres
        volumes:
            - dbdata:/var/lib/postgresql/data
volumes:
    dbdata:
```

Advanced management

- **Volume sharing between containers**
- Sharing the same volume between different services enables common access to data, useful for distributed applications.

```
services:
    service1:
        volumes:
            - shared_data:/data
    service2:
        volumes:
            - shared_data:/data
volumes:
    shared_data:
```

Advanced management

- **Backup and Restore**
- Backing up volume data on a regular basis is crucial to avoid losses in the event of problems.

```
docker run --rm --volumes-from db_container -v $(pwd):/backup ubuntu tar cvf /backup/backup.tar /dbdata
```

  - ‘`--rm`’ automatically deletes the container after execution.
  - ‘`--volumes-from db_container`’ uses the volumes of the container named db_container.
  - ‘`-v $(pwd):/backup`’ mounts the current host directory ($(pwd)) in the container at /backup.
  - ‘`ubuntu`’ is the temporary Docker image used.
  - ‘`tar cvf /backup/backup.tar /dbdata`’ is the command executed in the container.

Advanced management

- **Data security**
- Volume permissions help to secure data, limiting write or read access as required.

```
services:
        app:
                volumes:
                        - type: volume
                        source: my_volume
                        target: /app/data
                        volume:
                                nocopy: true
                                read_only: true
volumes:
        my_volume:
```

Advanced management

- **Performance Optimization**
- Bind mounts, which link a host folder to a container, can improve file access for better performance.

```
services:
    app:
        volumes:
            - type: bind
            source: ./myapp
            target: /usr/src/app
```

Solving common problems

- **Configuration problems**
- **Point:** Incorrect syntax in YAML file
- **Example:** A service is not started because the indentation is incorrect
- **Solution:** Use an online YAML validator to identify and correct errors

- **Network problems**
- **Issue:** Port conflict
- **Example:** Two services try to use the same port on the host
- **Solution:** Assign different ports for each service in the docker-compose.yml file
- Scan port : sudo netstat -tuln | grep [PORT]

- **Service Dependency Problems**
- **Point:** Cyclical dependency between services
- **Example:** Service A waits until Service B is up and running
- **Solution:** Use depends_on option in docker-compose.yml to explicitly define startup order

Solving common problems

- **Performance problems**
- **Point:** Excessive CPU or memory usage
- **Example:** A container uses all available memory
- **Solution:** Limit container resources using cpus and mem_limit in docker-compose.yml

- **Security Issues**
- **Point:** Unnecessary port exposure
- **Example:** A database port is accessible from the outside
- **Solution:** Ensure that only the necessary ports are mapped to the host

- **Debugging and Logging**
- **Point:** Analyze logs to identify errors
- **Example:** A service fails without clear indication
- **Solution:** Use docker-compose logs [service_name] to inspect the logs of the service concerned

Best Practices in Docker Compose

- **Code Organization**:

Split large docker-compose.yml files into smaller, manageable files (e.g., docker-compose.dev.yml for development).

- **Environment Separation**:

Use .env files to separate environment variables for different stages (development, production).

```
docker-compose --env-file .env.prod up
```

- **Service Scalability**:

Use scale command to dynamically change the number of running instances.

```
docker-compose up --scale web=3
```

Best Practices in Docker Compose

- **Security Considerations**:
- Avoid hardcoding sensitive data; use Docker secrets or environment variables.

```
POSTGRES_PASSWORD: ${DB_PASS}
```

- **Performance Optimization**:
- Monitor and limit resource usage (CPU, memory) per service.

```
services:
 web:
  deploy:
   resources:
    limits:
     cpus: '0.50’
     memory: 50M
```

Exemple:

- Find an exemple:
  - https://github.com/sebDtSci/iris_deployement

Here you'll find a simple docker compose deployment project. You can explore, modify and try new things on this basis.

If you have any questions, don't hesitate to contact me on linkedin.