Docker Compose

Docker compose la base

Docker Compose permet de faire tourner une application composée de **plusieurs containers**.

Docker Compose (V2) consiste en

- **Compose file**: un fichier yaml de definition et de gestion des services (volumes et et réseaux) qui définit intégralement l'application.
  - Le fichier est `compose.yaml`.
  - on peut aussi utiliser `docker-compose.yaml` mais c'est un nom lié à la version 1 de Docker Compose.
- **Compose CLI**: une série de commandes pour démarrer, stopper, surveiller (log et status des services) etc l'application

Docker Compose est tres adapté pour l'intégration continue (CI/CD).

Exemples d'applications

- un site Wordpress adossé à une base de donnée Mysql
- Une application NGinx + Redis pour le cache
- Un stack machine learning avec MLops, Airflow, ...
- Un LLM en local avec Ollama et une interface web

Pourquoi ?

Docker Compose permet de définir et de gérer des applications multi-conteneurs à partue d'**un seul fichier** YAML.

C'est puissant!!!

- simplifier la tâche complexe d'orchestration et de coordination de divers services,
- faciliter la gestion et la réplication de votre environnement applicatif.
- En **développement** : docker compose remplace plusieurs page de "getting started" en un seul fichier "machine readable" et quelques commandes.
- Pour les **tests**: docker compose permet de créer et de supprimer tout un environnement de test avec quelques ligne de commandes
- pour la **production**: l'organisation de la mise en production des briques applicatives est

entièrement automatisée

Spécifier tout une application à partir d'un unique fichier permet aussi le contrôle des versions. Le `compose.yaml` est dans git.

Comment ?

Structure et éléments du fichier compose.yaml

Le fichier `compose.yaml` (ou `docker-compose.yml` pour les versions plus anciennes) est placé à la racine de votre répertoire de projet.

Structure du fichier et les éléments principaux d'un fichier `compose.yaml` :

a. Version : (optionnel)(obsolete)

- Spécifie la version du format du fichier Compose.
- Exemple : `version: '3.8'`

b. Services :

- Définit les conteneurs qui doivent être exécutés.
- Chaque service reçoit un nom et peut avoir diverses options de configuration.

Par exemple: postgres, nginx, worker, appel a api, cache, etc ....

c. Réseaux (Networks) :

- Définit les réseaux à créer pour la communication entre les conteneurs.

d. Volumes :

- Définit les volumes nommés qui peuvent être réutilisés par plusieurs services.

Voici la structure de base d'un fichier `compose.yaml` :

```yaml
services:
  service1:
    # configuration pour service1
  service2:
    # configuration pour service2

networks:
  network1:
    # configuration pour network1

volumes:
  volume1:
    # configuration pour volume1
```

Illustration

1. Le quickstart de la doc https://docs.docker.com/compose/gettingstarted/
2. Une application Python/FastAPI application https://github.com/docker/awesome-compose/tree/master/fastapi

C'est simple et très puissant

CLI de base: `up`, `down`, `logs`, `ps`

- Pour **démarrer** tous les services définis dans votre fichier compose.yaml : `docker compose up`
- Pour **arrêter** et supprimer les services en cours d'exécution : `docker compose down`
- Si vous souhaitez surveiller la sortie de vos conteneurs en cours d'exécution et débugger des problèmes, on affiche les logs avec : `docker compose logs`
- Pour lister tous les services : `docker compose ps`

La liste complète des commandes est ici : https://docs.docker.com/reference/cli/docker/compose/

Command	Description
docker compose **build**	Build or rebuild services
docker compose config	Parse, resolve and render compose file in canonical format
docker compose cp	Copy files/folders between a service container and the local filesystem
docker compose create	Creates containers for a service
docker compose events	Receive real time events from containers
docker compose	Execute a command in a running container

exec	
docker compose images	List images used by the created containers
docker compose kill	Force stop service containers
docker compose ls	List running compose projects
docker compose pause	Pause services
docker compose port	Print the public port for a port binding
docker compose pull	Pull service images
docker compose push	Push service images
docker compose restart	Restart service containers
docker compose rm	Removes stopped service containers
docker compose **run**	Run a one-off command on a service
docker compose start	Start services
docker compose stop	Stop services
docker compose top	Display the running processes
docker compose unpause	Unpause services
docker compose version	Show the Docker Compose version information
docker compose wait	Block until the first service container stops
docker compose watch	Watch build context for service and rebuild/refresh containers when files are updated

Flags

Voici la traduction en français :

- `--dry-run` Exécute la commande en mode simulation
- `--env-file` Spécifie un fichier d'environnement alternatif
- `-f`, `--file` Fichiers de configuration Compose

et

- `--all-resources` Inclut toutes les ressources, même celles non utilisées par les services
- `--ansi` Contrôle quand imprimer les caractères de contrôle ANSI ("never"|"always"|"auto")
- `--compatibility` Exécute compose en mode de compatibilité descendante
- `--parallel -1` Contrôle le parallélisme maximal, -1 pour illimité
- `--profile` Spécifie un profil à activer
- `--progress` Définit le type de sortie de progression (auto, tty, plain, json, quiet)
- `--project-directory` Spécifie un répertoire de travail alternatif (par défaut : le chemin du premier fichier Compose spécifié)
- `-p`, `--project-name` Nom du projet

Services

C'est la partie la plus importante, où sont définis les containers de l'application.

A chaque service correspond un container.

Principales options de configuration d'un service:

a) `image`: L'image Docker a utiliser (si on ne build pas l'image) b) `build`: Le chemin vers le Dockerfile (si on utilise pas une image déjà existante) c) `ports`: Mapping des ports du host vers le container d) `volumes`: Mounts paths or named volumes. e) `environment`: Sets environment variables. f) `depends_on`: Expresses dependency between services. g) `restart`: Defines the restart policy.

Example:

YAML
```yaml
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html
    environment:
      - NGINX_HOST=example.com
    depends_on:
      - db
    restart: always

  db:
    image: postgres:13
    volumes:
      - db-data:/var/lib/postgresql/data
```

On retrouve là, la plupart des flags utilisés en ligne de commande pour run un container!

Volumes

Voici la traduction en français :

Les volumes sont utilisés pour conserver les données générées et utilisées par les conteneurs Docker. Les options de configuration principales comprennent :

a) `driver` : Spécifie le pilote de volume à utiliser. b) `driver_opts` : Fournit des options spécifiques au pilote. c) `external` : Spécifie si le volume a été créé en dehors de Compose.

Exemple:

```yaml
volumes:
  db-data:
    driver: local
  cached-data:
    external: true
```

Networks

Voici la traduction en français :

Les réseaux définissent comment les conteneurs communiquent entre eux et avec le monde extérieur. Les options de configuration principales comprennent : a) `driver` : Spécifie le pilote réseau à utiliser. b) `driver_opts` : Fournit des options spécifiques au pilote. c) `ipam` : Personnalise la gestion des adresses IP.

Par exemple :

```yaml
networks:
  frontend:
    driver: bridge
  backend:
    driver: overlay
    driver_opts:
      encrypted: "true"
```

Configs and Secrets

These are used to manage configuration files and sensitive data.

Example:

```yaml
configs:
  my_config:
    file: ./my_config.txt

secrets:
  my_secret:
    file: ./my_secret.txt
```

Deploy

Used to specify configuration related to the deployment and running of services.

Key configuration options include:

a) `replicas`: Number of containers to run for the service. b) `update_config`: How service updates should be applied. c) `restart_policy`: How to restart containers when they exit.

Example:

```yaml
services:
  web:
    image: nginx
    deploy:
      replicas: 3
      update_config:
        parallelism: 1
        delay: 10s
      restart_policy:
        condition: on-failure
```

These elements form the core of a `compose.yaml` file, allowing you to define complex, multi-container applications in a declarative way.

Exercice

Faisons le tutorial d'intro de la doc https://docs.docker.com/compose/gettingstarted/

Resources

- repo awesome compose - exemples de compose.yaml files https://github.com/docker/awesome-compose?tab=readme-ov-file
  - postgres + pgAdmin : https://github.com/docker/awesome-compose/tree/master/postgresql-pgadmin
  - minecraft : https://github.com/docker/awesome-compose/tree/master/minecraft