

DOCKER COMPOSE – RÉCAPITULATIF DES COMMANDES DE BASE

TABLE DES MATIÈRES

A. Nomenclature du fichier Docker Compose.....	2
B. Commandes Docker Compose.....	3
C. Compléments sur Docker Compose.....	4
1. Nom du projet, du dossier et du fichier Compose par défaut.....	4
2. Préservation des données de volume lors de la création des conteneurs.....	4
3. Création uniquement des conteneurs qui ont changé.....	4
4. Prise en charge des variables dans le fichier compose.yaml.....	4
5. Utilisation des « secrets » dans Docker Compose.....	5

Le contenu du fichier docker-compose.yml utilise la [syntaxe yaml](#) :

- la syntaxe **impose une indentation** (nombre de caractères d'espacement au début de la ligne – j'utilise de manière arbitraire 2 caractères) qui marque une arborescence. **Cette indentation doit être absolument respectée pour que le fichier soit interprété correctement ;**
- les commentaires sont signalés par le signe dièse (#) (si par contre le dièse apparaît dans une chaîne, il signifie alors un nombre littéral) ;
- les éléments de listes sont dénotés par le tiret (-), suivi d'une espace ;
- les tableaux sont de la forme clé: valeur, à raison d'un couple par ligne ;
- les scalaires peuvent être entourés de guillemets doubles ("), ou simples ('), sachant qu'un guillemet s'échappe avec un antislash (\), alors qu'un apostrophe s'échappe avec un autre apostrophe.



Utiliser un outil comme VS Code peut être d'une grande aide pour vérifier la validité du fichier.



Vous trouverez sur Internet un nombre conséquent de ressources sur Docker-Compose (avec un tiret au lieu de l'espace). Il s'agit de la version 1 de Docker Compose développée en Python qui n'est plus maintenue depuis le mois de juillet 2023.

La dernière version de Docker Desktop regroupe la plate-forme Docker Engine et Docker CLI, y compris Compose v2 a été développée en Go.

Une forte rétro-compatibilité existe. La plupart des fichiers YAML que vous trouverez sur Internet fonctionneront avec Docker Compose v2.

A. NOMENCLATURE DU FICHIER DOCKER COMPOSE

La nomenclature est très proche de celle qui est utilisée dans les commandes docker run.

Pour visualiser des exemples avec des explications :

- <https://blog.stephane-robert.info/docs/conteneurs/orchestrateurs/docker-compose/>

La meilleure référence reste la documentation officielle dans laquelle de nombreux exemples sont également donnés :

- <https://docs.docker.com/compose/compose-file/>

Les éléments essentiels sont les suivants :

- service :
- image : image Docker à utiliser.
- container_name : nom du conteneur qui va apparaître dans la liste (plutôt que de générer un nom au hasard).
- environment : variables d'environnement à passer au conteneur.
- ports : correspondance des ports ouverts : il est recommandé de les entourer de quotes " sinon leur valeur pourrait être mal interprétée.
- volumes : volumes à créer entre la machine hôte et le conteneur.
- build : si l'image doit être construite à partir d'un fichier Dockerfile.
- depends_on : si le conteneur dépend d'un autre pour son exécution (ex : une base de données).
- links : liens entre services (l'un « verra » l'autre dans son réseau avec son nom propre), on peut utiliser service:alias (ici db:db) ou juste service (ici db) puisqu'il s'agit du même nom.

B. COMMANDES DOCKER COMPOSE

docker compose config vérifie la configuration du fichier docker compose.yml.

docker compose up crée les conteneurs et démarre les services décrits dans le fichier docker compose.yml et ne rend pas la main.

docker compose up -d démarre les services décrits dans le fichier docker compose.yml et rend la main une fois que les services sont démarrés.

docker compose build reconstruit les services avant de les lancer. Elle n'est utile que si l'on souhaite qu'un container utilise une configuration définie dans un Dockerfile. Pour utiliser une image existante (comme dans notre cas ici), il suffit d'utiliser la syntaxe image suivie du nom de l'image.

docker compose logs retourne l'ensemble des logs des services depuis le dernier démarrage et rend la main.

docker compose logs <nom_d'un_service> retourne les logs du service correspondant.

docker compose ps affiche des informations sur les conteneurs créés.

docker compose logs -f affiche les logs des services et continue à les « écouter » sans rendre la main.

docker compose logs -f <nom_d'un_service> retourne les logs du service correspondant et continue à les « écouter » sans rendre la main.

docker compose stop arrête l'ensemble des conteneurs.

docker compose stop <nom_d'un_service> arrête le conteneur correspondant.

docker compose start démarre l'ensemble des conteneurs.

docker compose start <nom_d'un_service> démarre le conteneur correspondant.

docker compose restart redémarre l'ensemble des services.

docker compose restart <nom_d'un_service> redémarre le service.

docker compose down stoppe et supprime l'ensemble des conteneurs.

docker compose down <nom_d'un_service> stoppe le service et supprime le conteneur correspondant.

docker compose rm supprime l'ensemble des conteneurs (le ou les conteneurs doivent avoir été arrêtés au préalable).

docker compose rm <nom_d'un_service> supprime le conteneur correspondant (le conteneur doit avoir été arrêté au préalable).

docker compose exec <nom_d'un_service> bash fournit une console bash au sein du conteneur correspondant.

docker compose run exécute une commande dans un conteneur.

C. COMPLÉMENTS SUR DOCKER COMPOSE

1. NOM DU PROJET, DU DOSSIER ET DU FICHIER COMPOSE PAR DÉFAUT

Le nom du projet par défaut est le nom de base du répertoire du projet. Il est possible de définir un nom de projet personnalisé à l'aide de l'option « - » en ligne de commande ou de la variable d'environnement COMPOSE_PROJECT_NAME .

Le répertoire du projet par défaut est le répertoire de base du fichier Compose. Une valeur personnalisée peut être définie avec l'option de ligne de commande « --project-directory ».

Le fichier « compose.yaml » de base définit la structure de l'application. Il est possible de spécifier d'autres fichiers pour le surcharger en modifiant ou ajoutant des propriétés sur des services existants et aussi d'ajouter des services supplémentaires. Par défaut, Docker Compose lit les deux fichiers suivants :

- compose.yaml ;
- compose.override.yaml (optionnel).

Il est également possible d'utiliser des noms de fichiers différents avec l'option « -f », par exemple :

docker compose -f compose-kali.yaml up

2. PRÉSERVATION DES DONNÉES DE VOLUME LORS DE LA CRÉATION DES CONTENEURS

Compose préserve tous les volumes utilisés par les services. Lors de l'exécution de « docker compose up », s'il trouve des conteneurs issus des exécutions précédentes, il copie les volumes de l'ancien conteneur vers le nouveau conteneur. Ce processus garantit que les données créées en volumes ne sont pas perdues.

3. CRÉATION UNIQUEMENT DES CONTENEURS QUI ONT CHANGÉ

Compose met en cache la configuration utilisée pour créer un conteneur. Lorsque vous redémarrez un service qui n'a pas changé, Compose réutilise les conteneurs existants. La réutilisation des conteneurs signifie que vous pouvez apporter des modifications à votre environnement très rapidement.

4. PRISE EN CHARGE DES VARIABLES DANS LE FICHIER COMPOSE.YAML

Compose prend en charge les variables qui permettent de personnaliser le déploiement pour différents environnements ou différents utilisateurs.

Voir ici : <https://docs.docker.com/compose/environment-variables/set-environment-variables/>

5. UTILISATION DES « SECRETS » DANS DOCKER COMPOSE

Un secret est toute donnée, telle qu'un mot de passe, un certificat ou une clé API, qui ne doit pas être transmise sur un réseau ou stockée en clair dans un Dockerfile ou dans le code source de votre application.

Docker Compose permet d'utiliser des secrets sans avoir à utiliser de variables d'environnement pour stocker des informations. En effet, si on injecte des mots de passe et des clés API en tant que variables d'environnement, on risque une exposition involontaire d'informations. Les variables d'environnement sont souvent disponibles pour tous les processus et il peut être difficile d'en suivre l'accès. Ils peuvent également être imprimés dans des journaux lors du débogage d'erreurs à votre insu. L'utilisation de secrets atténue ces risques.

La procédure est ici : <https://docs.docker.com/compose/use-secrets/>.