Commandes Docker Essentielles pour les Développeurs

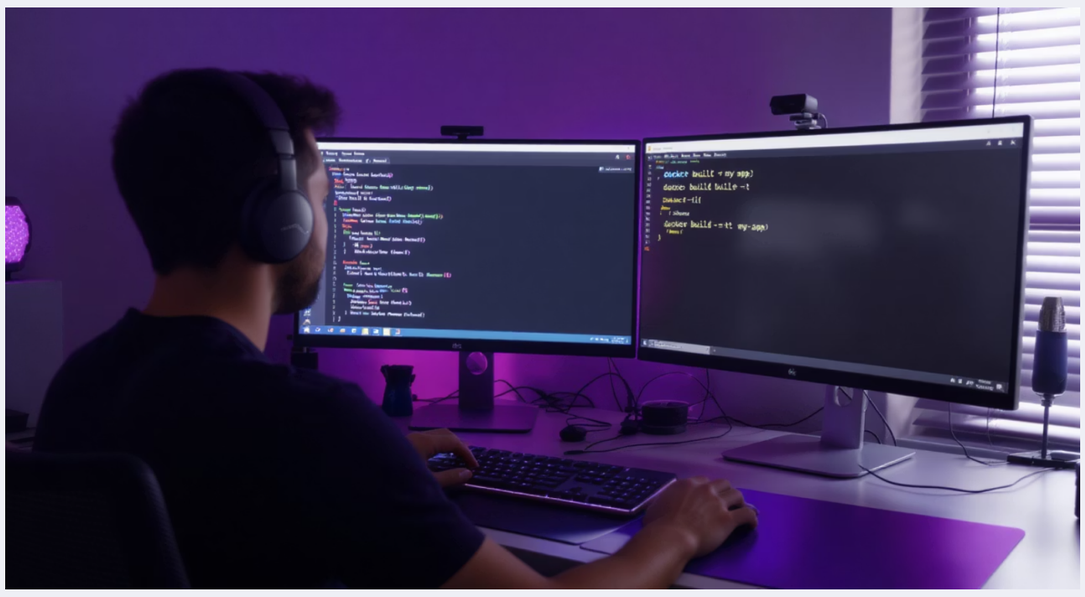
Ce document présente les commandes Docker fondamentales que tout développeur devrait connaître pour optimiser son flux de travail. Des instructions de construction d'images aux techniques avancées de gestion des conteneurs, ce guide complet couvre les aspects essentiels de l'écosystème Docker pour faciliter le développement, le déploiement et la maintenance d'applications conteneurisées.

Philémon Globléhi

1. Gestion des Images : `docker build`

La commande `docker build` est au cœur de la création d'images Docker personnalisées. Elle permet de transformer un Dockerfile en une image prête à l'emploi, encapsulant votre application et toutes ses dépendances.

- Utiliser `docker build -t nomimage:tag .` pour créer une image à partir d'un Dockerfile situé dans le répertoire courant
- Exploiter la prise en charge des sources distantes (Git ou archives) pour automatiser davantage le processus de construction
- Appliquer l'option `-t` pour taguer l'image, facilitant ainsi son organisation et son déploiement ultérieur



Un Dockerfile bien structuré et des commandes de construction appropriées constituent la base d'une stratégie de conteneurisation efficace.

Philémon Globléhi

2. Exécution des Conteneurs : `docker run`

La commande `docker run` transforme une image statique en un conteneur fonctionnel. C'est l'équivalent du lancement d'une application, mais dans un environnement isolé et reproductible.

Lancement en arrière-plan

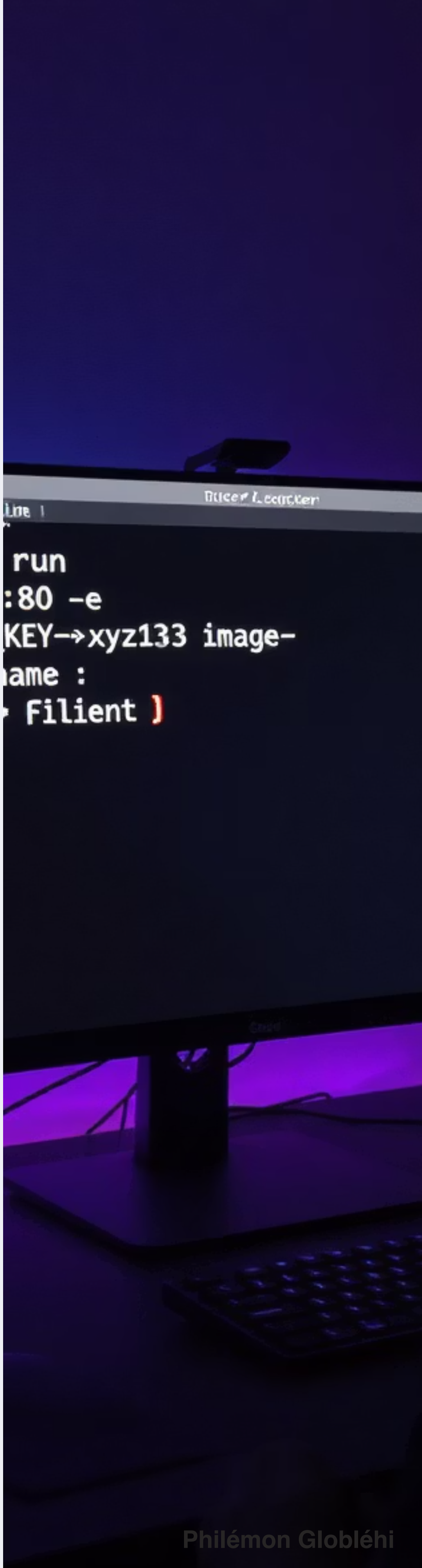
`docker run -d -p 8080:80 --name monapp nomimage` pour démarrer un conteneur en tâche de fond avec un mappage de port et un nom personnalisé

Options essentielles

Utiliser `-d` (détacher) pour exécuter en arrière-plan, `-p` pour mapper les ports entre l'hôte et le conteneur, et `--name` pour nommer votre conteneur

Configuration avancée

Possibilité de passer des variables d'environnement avec `-e VAR=valeur` et de monter des volumes avec `-v chemin_hote:chemin_conteneur` pour la persistance des données



3. Manipulation des Conteneurs : `start`, `stop`, `restart`

Une fois que vos conteneurs sont créés, Docker fournit un ensemble de commandes simples mais puissantes pour gérer leur cycle de vie. Ces commandes vous permettent de contrôler l'état d'exécution de vos applications conteneurisées sans avoir à les recréer.

docker start

Utilisez `docker start <nom|id>` pour relancer un conteneur précédemment arrêté, conservant ainsi sa configuration et ses volumes associés.

docker stop

Appliquez `docker stop <nom|id>` pour arrêter proprement un conteneur actif, en envoyant un signal SIGTERM suivi d'un SIGKILL si nécessaire.

docker restart

Exécutez `docker restart <nom|id>` pour redémarrer rapidement un conteneur, combinant efficacement les opérations stop et start en une seule commande.

Ces commandes peuvent être appliquées à plusieurs conteneurs simultanément en spécifiant plusieurs noms ou identifiants, séparés par des espaces.

4. Inspection des Conteneurs et Images

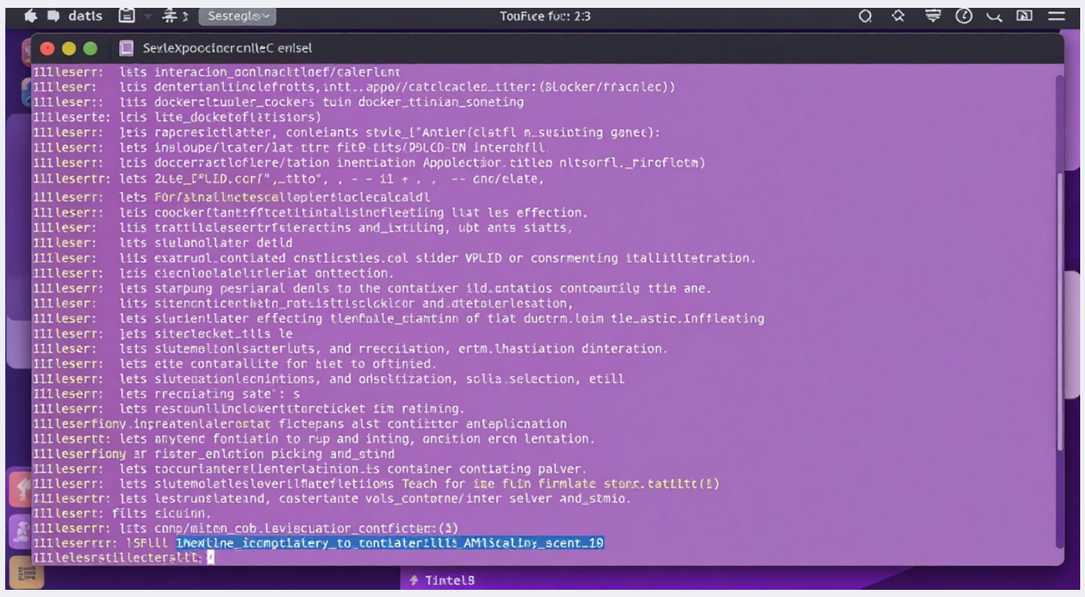
Pour un développement efficace avec Docker, il est crucial de pouvoir examiner l'état de votre environnement. Docker propose plusieurs commandes d'inspection qui vous permettent d'obtenir des informations détaillées sur vos conteneurs et images.

Commande	Description	Utilisation courante
`docker ps -a`	Affiche tous les conteneurs	Vérifier quels conteneurs sont en cours d'exécution ou arrêtés
`docker images`	Liste toutes les images disponibles	Identifier les images obsolètes ou volumineuses
`docker inspect <nom|id>`	Fournit des informations détaillées	Analyser la configuration, les volumes et les réseaux

L'option `--format` peut être utilisée avec `docker inspect` pour extraire spécifiquement les informations nécessaires au format JSON ou Go template, rendant l'automatisation plus simple.

5. Accès à l'Intérieur d'un Conteneur : `docker exec`

La commande `docker exec` est indispensable pour le débogage et la maintenance des conteneurs en cours d'exécution. Elle vous permet d'ouvrir un shell interactif à l'intérieur d'un conteneur actif, vous donnant ainsi un accès direct à son système de fichiers et à ses processus.



L'option `-i` maintient STDIN ouvert, tandis que `-t` alloue un pseudo-TTY, créant ainsi une expérience interactive similaire à une connexion SSH.

> `docker exec -it <nom|id> /bin/bash` pour ouvrir un terminal Bash interactif dans le conteneur spécifié

Pour les conteneurs basés sur des images légères comme Alpine ou BusyBox, qui n'incluent pas Bash par défaut, utilisez plutôt `/bin/sh` :

> `docker exec -it <nom|id> /bin/sh` pour les distributions Linux minimalistes

Philémon Globléhi

6. Nettoyage du Système Docker

Au fil du temps, les systèmes Docker peuvent accumuler des conteneurs arrêtés, des images obsolètes et des volumes non utilisés. Voici les commandes essentielles pour maintenir votre environnement Docker propre et optimisé.

Suppression de conteneurs

`docker rm <nom|id>` pour éliminer un conteneur arrêté. Ajoutez l'option `-f` pour forcer la suppression d'un conteneur en cours d'exécution.

Suppression d'images

`docker rmi <image>` pour supprimer une image inutilisée du système. Assurez-vous qu'aucun conteneur n'utilise cette image avant de la supprimer.

Nettoyage complet

`docker system prune` pour éliminer en une seule commande tous les conteneurs arrêtés, les réseaux non utilisés, les images pendantes et les caches de construction.

Pour une gestion encore plus approfondie, vous pouvez utiliser `docker system prune -a --volumes` qui supprimera également toutes les images non utilisées et les volumes, libérant ainsi un maximum d'espace disque.

7. Gestion des Volumes et Réseaux

Les volumes et les réseaux Docker sont essentiels pour la persistance des données et la communication entre conteneurs. Une bonne gestion de ces ressources garantit des applications robustes et bien organisées.

Commandes pour les volumes

- `docker volume ls` pour afficher tous les volumes créés
- `docker volume create monvolume` pour créer un nouveau volume nommé
- `docker volume rm <nom>` pour supprimer un volume non utilisé
- `docker volume inspect <nom>` pour examiner les détails d'un volume

Commandes pour les réseaux

- `docker network ls` pour lister tous les réseaux disponibles
- `docker network create --driver bridge monreseau` pour créer un réseau personnalisé
- `docker network connect monreseau monconteneur` pour ajouter un conteneur à un réseau
- `docker network rm monreseau` pour supprimer un réseau non utilisé

Philémon Globléhi

8. Utilisation de Docker Compose

Docker Compose simplifie considérablement la gestion d'applications multi-conteneurs en permettant de définir et d'exécuter tous les services dans un fichier YAML unique.

Lancement des services

`docker-compose up -d` pour démarrer tous les services définis dans le fichier `docker-compose.yml` en mode détaché (arrière-plan)

Arrêt et nettoyage

`docker-compose down` pour arrêter tous les conteneurs et supprimer les réseaux créés. Ajoutez `--volumes` pour supprimer également les volumes associés

Options avancées

Utiliser `--build` pour forcer la reconstruction des images, `-f fichier.yml` pour spécifier un fichier autre que celui par défaut, et `--scale service=n` pour ajuster le nombre d'instances

```
# Exemple de docker-compose.ymlversion: '3'services:  web:    build: ./frontend
ports:      - "80:80"  api:    image: backend:latest    environment:      -
DB_HOST=db  db:    image: postgres:13    volumes:      - db-data:/var/lib/
postgresql/datavolumes:  db-data:
```

Philémon Globléhi

9. Commandes Avancées et Bonnes Pratiques

Pour une utilisation professionnelle de Docker, ces commandes avancées et ces bonnes pratiques vous aideront à améliorer votre workflow de développement et à résoudre efficacement les problèmes.

Surveillance des logs

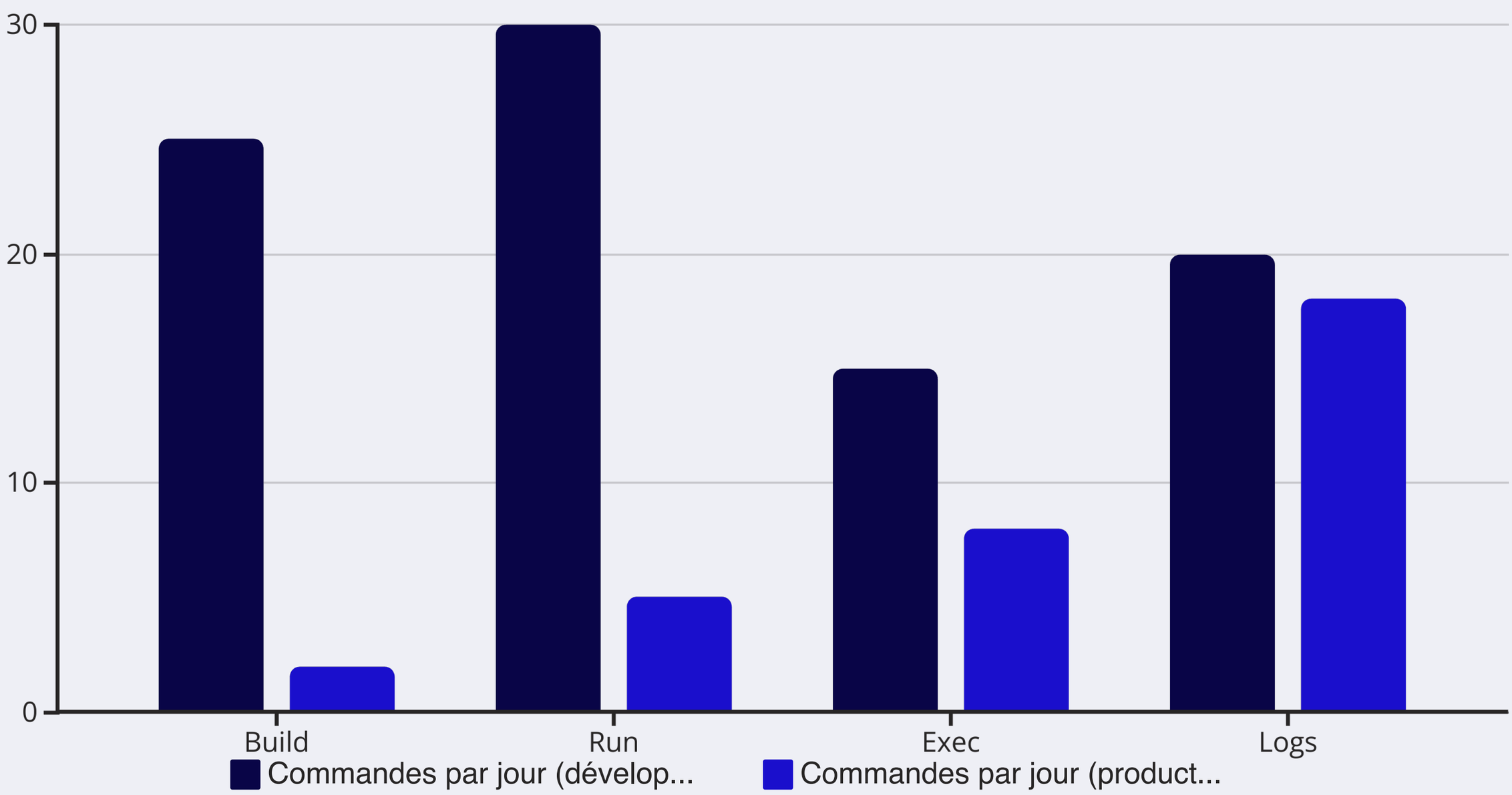
Utilisez `docker logs <nom|id>` pour consulter les sorties standard et d'erreur d'un conteneur. Ajoutez l'option `-f` pour suivre les logs en temps réel, similaire à `tail -f`.

Monitoring des ressources

Exécutez `docker stats` pour obtenir une vue en temps réel de la consommation CPU, mémoire et réseau de vos conteneurs actifs, facilitant ainsi l'identification des goulots d'étranglement.

Versionnement des images

Privilégiez toujours le versionnement explicite de vos images (`app:1.0.3` plutôt que `app:latest`) pour garantir la reproductibilité et faciliter les rollbacks dans vos pipelines CI/CD.



Ce graphique illustre la différence d'utilisation des commandes Docker entre les environnements de développement et de production, montrant l'importance relative de chaque commande selon le contexte.

Philémon Globléhi