UNIVERSITÉ DE SHERBROOKE

Faculté de génie

Département de génie informatique

Rapport APP3

Systèmes répartis

GIF391

Présenté à

Équipe de formateurs de la session S3

Présenté par

Raphael Bouchard – bour0703

Alexis Guérard – guea0902

Sherbrooke – 12 juin 2023

Table des matières

[1. Description de la solution pour les six étapes de la problématique 4](#_Toc137217878)

[1.1 Exemples d’utilisation des commandes nouveau, retrait et liste 4](#_Toc137217879)

[1.2 Fichier de configuration fonctionnel 5](#_Toc137217880)

[1.3 Fichier yml utilisé pour docker-compose 6](#_Toc137217881)

[1.4 Explication des problèmes inhérents à cette étape 7](#_Toc137217882)

[1.5 Fichier yml utilisé pour docker-compose 7](#_Toc137217883)

[1.6 Fichier yml utilisé pour l’orchestration avec kubernetes 8](#_Toc137217884)

[2. Discussion de la structure, des avantages et des inconvénients de la version ultime du système 9](#_Toc137217885)

[2.1 Technologies Linux sous-jacentes utilisées 9](#_Toc137217886)

[2.2 Identification des ressources 9](#_Toc137217887)

[2.3 Contrôle de l’accès aux ressources 9](#_Toc137217888)

[2.4 Gestion des accès aux fichiers utilisés 10](#_Toc137217889)

[2.4.1 pilotes utilisables pour la persistance du système de fichiers à union 10](#_Toc137217890)

[2.4.2 pilote de persistence a été utilisé et pourquoi ? 10](#_Toc137217891)

[2.5 Discussion de la configuration réseau permettant aux différents conteneurs d’interagir 10](#_Toc137217892)

[2.6 Discussion de la duplication mise en place pour les ressources 11](#_Toc137217893)

[2.6.1 Quelles ressources doivent être dupliquées, et pourquoi ? 11](#_Toc137217894)

[2.6.2 Dans quel cas la duplication peut se faire sur une machine réelle unique, dans quel cas elle doit être distribuée sur plusieurs machines réelles ? 11](#_Toc137217895)

Liste des figures

[Figure 1 : Exemples d'utilisation des commandes nouveau, retrait et liste 4](#_Toc137401865)

[Figure 2 : Fichier de configuration 5](#_Toc137401866)

[Figure 3 : Fichier YML de l'arret03 6](#_Toc137401867)

[Figure 4 : Fichier YML de l'arret05 8](#_Toc137401868)

[Figure 5 : client\_pod.yaml 9](#_Toc137401869)

[Figure 6 : db\_pod.yaml 9](#_Toc137401870)

[Figure 7 : gest\_pod.yaml 10](#_Toc137401871)

[Figure 8 : serveur\_pod.yaml 11](#_Toc137401872)

# Description de la solution pour les six étapes de la problématique

## Exemples d’utilisation des commandes nouveau, retrait et liste

Pour la première étape, nous avons utilisé Docker pour créer un système monolithique minimaliste. À l'aide des commandes Docker appropriées, comme docker ps, docker images, docker run, docker exec, docker stop et docker build, nous avons géré un conteneur basé sur l'image construite "arret01" dans la machine virtuelle à l’aide de la commande « docker build -t arret01 . » . Nous avons vérifié le système en faisant la commande « docker run -it arret01 ». Les commandes "nouveau", "retrait" et "liste" étaient disponibles pour ajouter, supprimer et afficher les citations dans le serveur. La commande "nouveau" permet d'ajouter une nouvelle citation générée aléatoirement à l'aide de la commande "fortune" dans le serveur monolithique. La commande "retrait" permet de supprimer une citation spécifique. La commande "liste" affiche toutes les citations présentes. La commande "aide" fournit des informations détaillées sur les différentes commandes disponibles.

Une image contenant texte, capture d’écran

Description générée automatiquement

Figure 1 : Exemples d'utilisation des commandes nouveau, retrait et liste

## Fichier de configuration fonctionnel

Pour la deuxième étape de la problématique, nous avons corrigé le fichier de configuration pour assurer le bon fonctionnement du système. Nous avons ajouté la configuration du réseau pour le serveur, le client et le gestionnaire, ainsi que les environnements nécessaires. De plus, nous avons utilisé un volume pour permettre de voir les modifications dans le fichier "todo.txt". Pour démarrer le système, nous avons construit les trois images à l'aide des Dockerfiles fournis, puis nous avons exécuté la commande "docker-compose up" pour lancer les conteneurs. Ensuite, nous avons exécuté le conteneur du client et utilisé la commande "call" pour créer des citations automatiquement et les ajouter dans notre fichier "todo.txt".

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Figure 2 : Fichier de configuration

## Fichier yml utilisé pour docker-compose

Pour la troisième étape de la problématique, nous avons modifié le fichier .yml pour docker-compose afin de gérer la configuration du système avec trois paires client-serveur distinctes et une troisième image pour le gestionnaire qui se connecte aux trois serveurs. Dans le fichier .yml, nous avons défini les services pour chaque client et serveur en utilisant les fichiers Dockerfiles. Chaque client a été lié à un serveur spécifique en spécifiant les conteneurs correspondants dans la configuration du réseau. Nous avons également utilisé des volumes pour chaque serveur afin de voir les modifications apportées aux listes "todo" des clients. Cela a permis aux clients d'utiliser le système individuellement, tout en permettant au gestionnaire d'accéder en lecture aux listes "todo" des trois clients.

Une image contenant texte, capture d’écran, Police

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 3 : Fichier YML de l'arret03

## Explication des problèmes inhérents à cette étape

À cette quatrième étape, il fallait faire fonctionner un seul serveur pour les trois clients, avec également un gestionnaire. Cependant, le serveur ne garantit pas l'atomicité des opérations dans le fichier todo.txt. Cela soulève des problèmes potentiels, car plusieurs clients pourraient tenter d'accéder et de modifier simultanément le fichier, ce qui peut entraîner des conflits, des incohérences ou des pertes de données.

La problématique principale réside dans la synchronisation des opérations effectuées par les différents développeurs sur le fichier partagé. En l'absence de garantie d'atomicité, il peut y avoir des situations où deux clients tentent de modifier le fichier en même temps, ce qui peut entraîner des conflits de données ou des écritures incomplètes. Il peut également y avoir des problèmes de cohérence lorsque des opérations simultanées entraînent des incohérences dans le contenu du fichier.

Un autre défi est de gérer efficacement les accès concurrents au fichier partagé. Si plusieurs développeurs tentent de lire, ajouter ou supprimer des éléments simultanément, il peut y avoir des problèmes de concurrence qui peuvent compromettre l'intégrité et la cohérence des données.

ll faudrait donc mettre en place des mécanismes de synchronisation et de verrouillage appropriés pour garantir la cohérence des opérations effectuées.

## Fichier yml utilisé pour docker-compose

Au cinquième arrêt, nous avions mis en place une solution où le serveur principal (todo) avait été amélioré pour fonctionner avec une instance de l'image de la base de données PostgreSQL (todo-bd). Cette base de données assurait l'atomicité des requêtes, garantissant ainsi la cohérence des opérations effectuées sur les données. L'image PostgreSQL était tirée du registre par défaut de Docker, ce qui facilitait son intégration dans le système. Nous avions donc modifié le fichier docker-compose.yml qui regroupait les trois clients, le gestionnaire, le serveur todo-bd et la base de données PostgreSQL, permettant ainsi le déploiement et l'exécution de l'ensemble du système avec Docker Compose.

Une image contenant texte, capture d’écran, Police

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 4 : Fichier YML de l'arret05

## Fichier yml utilisé pour l’orchestration avec kubernetes

Au sixième arrêt, nous avons évolué notre système pour le rendre utilisable à grande échelle. Pour éviter les problèmes de congestion sur une seule base de données, nous avons dupliquer la base de données. Pour cela, nous avons utilisé Kubernetes pour répartir les bases de données dans plusieurs pods, permettant ainsi le déploiement sur différentes machines physiques. En utilisant des fichiers yml adaptés à Kubernetes, nous avons pu développer et faire fonctionner efficacement le système à grande échelle, en exploitant les fonctionnalités de réplication offertes par Kubernetes.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 5 : client\_pod.yaml

Une image contenant texte, capture d’écran, Police, document

Description générée automatiquement

Figure 6 : db\_pod.yaml

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Figure 7 : gest\_pod.yaml

Une image contenant texte, Police, capture d’écran, menu

Description générée automatiquement

Figure 8 : serveur\_pod.yaml

# Discussion de la structure, des avantages et des inconvénients de la version ultime du système

## Technologies Linux sous-jacentes utilisées

Les technologies Linux sous-jacentes utilisées pour permettre la mise en œuvre des conteneurs sont les Cgroups (Control Groups), les Namespaces et les systèmes de fichiers à union. Les Cgroups permettent de définir et de limiter les ressources utilisées par les conteneurs, garantissant une utilisation équilibrée des ressources système. Les Namespaces fournissent un environnement isolé pour chaque conteneur, séparant les processus, les utilisateurs, les réseaux et les systèmes de fichiers. Les systèmes de fichiers à union superposent des vues cohérentes de plusieurs systèmes de fichiers, permettant aux conteneurs d'avoir leur propre espace de stockage tout en partageant une base commune. Ces technologies Linux sont essentielles pour l'isolation, la gestion des ressources et la gestion des systèmes de fichiers nécessaires au bon fonctionnement des conteneurs et à la création d'environnements conteneurisés performants et sécurisés.

## Identification des ressources

L’identification des ressources se fait grâce aux Cgroups (Control Groups). Les Cgroups permettent de définir et contrôler les ressources allouées à chaque conteneur, tel que le CPU, la mémoire, les E/S (entrées/sorties) et le réseau. Ils assurent une gestion fine des ressources en attribuant des limites, des priorités et des quotas à chaque groupe de conteneurs. Ainsi, les ressources sont efficacement identifiées et isolées, ce qui permet de prévenir les conflits et d'optimiser l'utilisation des ressources disponibles dans le système.

## Contrôle de l’accès aux ressources

Le contrôle de l'accès aux ressources est assuré par les mécanismes de sécurité et de gestion des autorisations du système d'exploitation Linux. Chaque conteneur fonctionne dans son propre environnement isolé grâce à l'utilisation des Namespaces. Les Namespaces garantissent que les ressources attribuées à un conteneur sont uniquement accessibles par celui-ci et ne peuvent pas être utilisées par d'autres conteneurs ou par le système hôte. De plus, les politiques de contrôle d'accès au niveau du système d'exploitation, telles que les permissions de fichiers, les utilisateurs et les groupes, sont appliquées pour garantir que seules les ressources autorisées peuvent être accédées par les conteneurs et les processus qui y sont exécutés. Cela permet de maintenir l'isolation et la sécurité du système tout en contrôlant l'accès aux ressources.

## Gestion des accès aux fichiers utilisés

### pilotes utilisables pour la persistance du système de fichiers à union

Dans le cadre de l'arrêt 6 où le système doit être déployé à grande échelle, plusieurs pilotes peuvent être utilisés pour la persistance du système de fichiers à union. Parmi les options courantes figurent OverlayFS, AUFS, Device Mapper, et ZFS. Ces pilotes permettent de superposer les systèmes de fichiers, fournissant une abstraction qui combine les fichiers et les répertoires provenant de différentes sources en une seule vue cohérente.

### pilote de persistance a été utilisé et pourquoi ?

Pour répondre aux besoins de la problématique présentée à l'arrêt 6, le pilote de persistance OverlayFS aurait pu être utilisé. OverlayFS est un pilote léger et efficace, intégré nativement au noyau Linux. Il offre une bonne performance, une faible surcharge et une facilité d'utilisation. De plus, il prend en charge la superposition de plusieurs systèmes de fichiers en utilisant des couches, ce qui facilite la gestion des fichiers et des répertoires dans un environnement distribué. En utilisant OverlayFS, il est possible de mettre en place une solution de persistance du système de fichiers à union pour les conteneurs dans Kubernetes, permettant ainsi la gestion efficace des données partagées entre les conteneurs tout en minimisant la complexité et les problèmes de performance.

## Discussion de la configuration réseau permettant aux différents conteneurs d’interagir

Pour permettre l'interaction entre les conteneurs dans Kubernetes, une configuration réseau appropriée est mise en place. Chaque conteneur se voit attribuer une adresse IP unique à l'intérieur d'un réseau virtuel. Les conteneurs sont regroupés en pods, et la communication à l'intérieur d'un pod se fait en utilisant des adresses IP locales. Pour permettre l'interaction entre les pods, Kubernetes utilise des services qui fournissent des adresses IP stables et des noms DNS. Cela permet une communication fluide entre les conteneurs et services dans un cluster Kubernetes.

## Discussion de la duplication mise en place pour les ressources

### Quelles ressources doivent être dupliquées, et pourquoi ?

Dans la version ultime du système, où celui-ci doit être utilisé à grande échelle par mille développeurs, la ressource clé à dupliquer est la base de données. Étant donné qu'un grand nombre d'utilisateurs interagiront avec le système et effectueront des opérations sur la base de données, la duplication est nécessaire pour répartir la charge et éviter la congestion sur une seule base de données. En associant un maximum de cent développeurs par instance de base de données, chaque instance peut gérer un sous-ensemble de requêtes et garantir des performances optimales.

### Dans quel cas la duplication peut se faire sur une machine réelle unique, dans quel cas elle doit être distribuée sur plusieurs machines réelles ?

Dans l'optique d'une utilisation à grande échelle avec mille développeurs, il est préférable de distribuer les bases de données sur plusieurs machines réelles. Cela permet de bénéficier de la puissance de calcul et de la mémoire de plusieurs machines, ainsi que de répartir la charge de manière équilibrée. En utilisant Kubernetes et en configurant des pods pour chaque instance de base de données, il est possible de déployer les bases de données sur un ensemble de machines distinctes. Cette distribution garantit une meilleure extensibilité, une résilience accrue et une tolérance aux pannes, assurant ainsi la continuité du service même en cas de défaillance matérielle d'une machine.