Systèmes informatiques répartis

GUIDE DE L'ÉTUDIANTE ET DE L'ÉTUDIANT
S3 Génie Informatique – APP3

Été 2023 – Semaines 5, 6 et 7

Département de génie électrique et de génie informatique
Faculté de génie
Université de Sherbrooke

Historique des modifications

Date	Responsables	Description
mai 2004	Bernard Beaulieu	Version 1.0
mai 2005	Bernard Beaulieu	Version 1.1
mai 2006	Bernard Beaulieu	Version 1.2
mai 2007	Bernard Beaulieu	Version 1.3
mai 2008	Bernard Beaulieu	Version 1.4
mai 2009	Bernard Beaulieu	Version 1.5
mai 2010	Bernard Beaulieu	Version 1.6
mai 2011	Bernard Beaulieu	Version 1.7
mai 2012	Bernard Beaulieu	Version 1.8
mai 2013	Bernard Beaulieu	Version 1.9
mai 2014	Bernard Beaulieu	Version 1.10
mai 2015	Bernard Beaulieu	Version 1.11
mai 2016	Bernard Beaulieu	Version 1.12
mai 2017	Bernard Beaulieu	Version 1.13
mai 2018	Bernard Beaulieu	Version 1.14
mai 2019	Domingo Palao Munoz et Bernard Beaulieu	Version 1.15
mai 2020	Domingo Palao Munoz et Bernard Beaulieu	Version 2.0
mai 2021	Domingo Palao Munoz et Bernard Beaulieu	Version 2.1
mai 2022	Bernard Beaulieu et Frédéric Mailhot	Version 3.0

Auteur : Bernard Beaulieu et Frédéric Mailhot
Version : 3.1 (mai 2023)

Ce document est réalisé avec l'aide de LaTeX et de la classe `gegi-app-guide`.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

1 ÉNONCÉ DE LA PROBLÉMATIQUE

Back to the To-do

Vous vous réveillez dans un univers parallèle, où vous êtes maintenant Dominique McFly, jeune "geek" dont le frère est Marty McFly. Doc Emmett Brown vient tout juste de sortir précipitamment de sa DeLorean et vous demande de le suivre pour pouvoir vous aider vous-mêmes à surmonter des défis dans votre future vie professionnelle. Doc vous fait comprendre que votre logiciel personnel de gestion de choses à faire (liste "to-do") sera possiblement très populaire dans le futur, mais vous devrez traverser plusieurs étapes pour y arriver. Il vous propose de vous accompagner dans six moments clés de votre carrière, où vous devrez utiliser les systèmes docker, docker-compose et kubernetes pour faire fonctionner un système logiciel de complexité croissante, passant d'un programme monolithique à un système distribué robuste, multi-serveurs, multi-usagers, gérant l'historique et l'archivage des tâches. En faisant évoluer votre système, vous aurez à mettre en oeuvre un système transparent et hétérogène, en réduisant le couplage à son minimum. L'architecture du système qui sera proposée pourrait toucher les systèmes en couches, structurés par événements, micro-noyau (*microkernel*), et nuagique).

1. Premier arrêt : Votre moi futur a développé "dans son sous-sol" un système minimaliste comportant simplement un serveur monolithique (todo) qu'on peut démarrer (à l'aide de la commande *docker* avec une image pré-définie (*arret01*, qui se trouve dans la machine virtuelle Lubuntu), le tout en utilisant les commandes et paramètres appropriés (docker ps, docker images, docker run, docker exec, docker rm, docker stop, docker rmi etc.)).

   Il est alors possible de donner l'une des quatre commandes suivantes :
   — nouveau
   — retrait
   — liste
   — aide
   pour interagir avec le système de gestion de liste to-do (d'autres commandes plus complexes sont disponibles, telles que décrites lorsqu'on évoque la commande *aide*).

L'utilisation d'un conteneur se fera par *Command Line Interface* (*CLI*). Vous devrez accéder à l'intérieur du conteneur à l'aide d'un *shell* (*sh*, *bash* ou autre). Un certain nombre de commandes Linux devraient vous être utiles : *ls*, *ps*, *host*, parmi une multitudes d'autres commandes qui pourraient vous aider grandement dans votre travail.

2. Deuxième arrêt : Utilisation du serveur monolithique précédent (todo), avec un "client" simulé (usager) qui produit des requêtes périodiques au serveur todo. Les deux systèmes sont disponibles sous forme d'images docker décrites dans des Dockerfiles.

   Le serveur utilise un fichier (todo.txt) pour modifier la liste des choses à faire, ajoutant, retirant et produisant une liste des éléments inclus à un instant donné. Il existe un fichier yml pour docker-compose et destiné à démarrer le tout. Malheureusement, il semble que la configuration ne fonctionne pas. Vous devez la corriger, sous peine d'affecter négativement votre vie future.

3. Troisième arrêt : Quelques mois après le premier arrêt, votre moi futur travaille maintenant dans une petite équipe de trois (3) développeurs. Ayant vu votre système, les développeurs de l'équipe veulent chacun l'utiliser séparément, la gestionnaire de l'équipe ayant accès en lecture aux listes "todo" des trois développeurs. Vous devriez réutiliser les images du premier arrêt, en utilisant trois paires client-serveur (c1 avec s1, c2 avec s2, c3 avec s3), chaque client étant lié à un serveur distinct. Une troisième image (gestionnaire) doit se connecter aux trois serveurs, le conteneur associé ne faisant que des lectures. Les fichiers Dockerfile existent pour les trois types d'images, mais malheureusement il faut créer le fichier yml pour docker-compose, qui n'existe pas.

4. Quatrième arrêt : Le patron de votre gestionnaire, voyant votre système à l'oeuvre, propose de n'utiliser qu'un seul serveur (todo) pour les trois développeurs, avec une configuration incluant c1, c2 et c3 avec un seul serveur, le tout avec un gestionnaire g. Considérant que le serveur ne garantit pas l'atomicité des opérations dans la ressource partagée (le fichier todo.txt), croyez-vous pouvoir faire fonctionner le tout ? Le patron de votre gestionnaire désire voir une réalisation concrète pour démontrer que son idée est excellente.

5. Cinquième arrêt : Votre moi futur a créé un nouveau serveur (todo-bd), qui fonctionne maintenant avec une instance de l'image de la base de données PostgreSQL (postgres), qui assure l'atomicité des requêtes. L'image postgres est tirée directement du registre (*registry*) utilisé par défaut par docker. Malheureusement, votre moi futur n'a pas encore réussi à mettre tous les éléments ensemble (3 clients, 1 gestionnaire, 1 serveur

todo-bd et 1 postgres) dans un fichier yml unique pour docker-compose. Vous devez l'aider à tout prix !

6. Sixième arrêt : Le Vice-président Technologies, qui a entendu parler de votre système, désire qu'il soit maintenant utilisé à grande échelle, par l'ensemble des mille (1000) développeurs de l'entreprise. Anticipant de la congestion sur une base de données unique, il recommande d'en faire la duplication et d'associer un maximum de cent (100) développeurs par instance de base de données. À ce stade, il est clair qu'il faudra répartir les bases de données dans de multiples pods kubernetes, pour permettre le déploiement sur un ensemble de machines matérielles distinctes. Pour simplifier la préparation du système, le VP Technologies vous indique que *minikube* pourrait s'avérer utile. Pouvez-vous aider votre futur moi à développer et faire fonctionner un fichier yml à être utilisé avec kubernetes ?

Au retour de votre périple, et sans considérer les conséquences d'un éventuel paradoxe temporel, vous décidez d'analyser ce que vous avez fait à chacun des arrêts, pour faciliter le travail de votre futur moi. Qu'est-ce qui a fonctionné et quelles fonctionnalités sous-jacentes de docker ont été importantes ? Comment le système de fichiers à union a-t-il été utile ? Pensez-vous que les permissions gérées par les *cgroups* ont joué un rôle ? Qu'en est -il des *namespaces* ? En quoi l'atomicité des requêtes PostgreSQL a été importante ?

2 GUIDE DE LECTURE

2.1 Livres et documents à consulter

2.1.1 Lectures en lien avec la théorie des systèmes distribués

- Systems programming Designing and developing Distributed Applications. Richard John Anthony (disponible sur le site web de l'unité).
- UML 2. Pratique de la modélisation. Benoît Charroux, Aomar Osmani, Yann Thierry-Mieg. (Utilisé en S2)

2.1.2 Lectures en lien avec la conteneurisation

- Série modulaire sur les Systèmes distribués (disponible sur le site web de l'unité)
  - Fascicule 01 : Le protocole HTTP (disponible sur le site web de l'unité)
  - Fascicule 02 : Les conteneurs : Théorie (disponible sur le site web de l'unité)
- Jeff Nickoloff, Stephen Kuenzli, *Docker in Action, 2nd Edition*, Manning Publications, 310 p., 2019
- Marko Luksa, *Kubernetes in Action, 2nd Edition*, Manning Publications, 206 p., 2020
- Gabriel Shenker, *Learn Docker, 2nd Edition*, Packt, 558 p., 2020

2.2 Séquence d'étude suggérée

2.2.1 Lectures à faire avant le procédural

- Systems Programming and Developing Distributed Applications
  - Chapitre 5 : The Architecture View
  - Chapitre 6 : Transparency
- Fascicule 02 : Les conteneurs : Théorie
- UML 2. Pratique de la modélisation
  - Annexe D. Diagramme de déploiement
- Docker in Action : Chapitre 1

2.2.2 Lectures à faire avant le premier laboratoire

- Fascicule 01 : Le protocole HTTP
- Martin Heinz, Deep Dive into Docker Internals - Union Filesystem, 2021
- Docker in Action : Chapitres 2 et 3

2.2.3 Lectures à faire avant le deuxième laboratoire

- Docker in Action : Chapitres 11 et 12
- Learn Docker : Chapitre 11

2.3 Lectures supplémentaires (non essentielles, mais utiles), à faire avant les laboratoires

— Introduction à la ligne de commande Linux (si vous ne connaissez pas du tout Linux, c'est un bon point de départ)
— 20 commandes Linux très utiles pour l'administration de système
— Liste concise de commandes Linux utiles (*Cheat Sheet*)

3 LOGICIELS ET MATÉRIEL

— L'environnement de travail sera une machine virtuelle Linux utilisant le logiciel VMWare (Workstation Pro (Windows ou Linux) ou Fusion (macOS)). La machine virtuelle (contenant le système d'exploitation Ubunbtu) est disponible sur le site de l'APP.

— https://www.vmware.com/ca-fr/products/workstation-pro.html
— https://www.vmware.com/ca-fr/products/fusion.html

4 SANTÉ ET SÉCURITÉ

Dispositions particulières

Aucune.

5 SOMMAIRE DES ACTIVITÉS

Semaine 1

— Première rencontre de tutorat
— Étude personnelle
— Formation à la pratique procédurale 1
— Formation à la pratique en laboratoire 1
— Formation à la pratique en laboratoire 2

Semaine 2

— Consultation facultative
— Étude personnelle et exercices
— Validation pratique de la solution

Semaine 3

— Rédaction du rapport d'APP
— Remise des livrables d'APP
— Deuxième rencontre de tutorat
— Évaluation formative théorique
— Évaluation sommative théorique

6 PRODUCTIONS À REMETTRE

- Les productions se font par équipe de 2, sauf avis contraire.
- L'identification des membres des équipes doit être faite sur la page web de l'unité avant 16h30, le lendemain de votre premier tutorat. Les personnes qui n'ont pas d'équipe d'APP à ce moment seront mises en équipe par le tuteur.
- La date limite pour le dépôt électronique est le jour de votre deuxième tutorat avant le début du premier groupe. Les retards seront pénalisés.
- Les productions soumises à l'évaluation doivent être originales pour chaque équipe, sinon l'évaluation sera pénalisée en cas de non-respect de cette consigne, suivant le règlement des études. Pour plus de renseignements, voir la page sur l'intégrité intellectuelle.

Rapport d'APP

Le rapport, nommé CIP1_CIP2.pdf, doit contenir les éléments suivants :

- Identification de l'ensemble des membres de l'équipe (Prénom, nom et CIP)
- Une description de la solution utilisée pour chacune des six étapes de la problématique :
  1. Exemples d'utilisation des commandes *nouveau*, *retrait* et *liste*
  2. Fichier de configuration fonctionnel
  3. Fichier *yml* utilisé pour *docker-compose*
  4. Explication des problèmes inhérents à cette étape
  5. Fichier *yml* utilisé pour *docker-compose*
  6. Fichier *yml* utilisé pour l'orchestration avec *kubernetes*
- Discussion de la structure, des avantages et des inconvénients de la version ultime du système :
  - Quelles technologies Linux sous-jacentes ont été utilisées pour permettre la mise en oeuvre de conteneurs ? Expliquez brièvement (en un paragraphe) ce que chacune permet de faire :
    - Identification des ressources
    - Contrôle de l'accès aux ressources
    - Gestion des accès aux fichiers utilisés :
      - Quels sont les pilotes utilisables pour la persistance du système de fichiers à union ?
      - Quel pilote de persistence a été utilisé et pourquoi ?

- Discussion de la configuration réseau permettant aux différents conteneurs d'interagir
- Discussion de la duplication mise en place pour les ressources :
  - Quelles ressources doivent être dupliquées, et pourquoi ?
  - Dans quel cas la duplication peut se faire sur une machine réelle unique, dans quel cas elle doit être distribuée sur plusieurs machines réelles ?

7 ÉVALUATIONS

7.1 Grille d'évaluation

La note attribuée aux activités pédagogiques de l'unité est une note individuelle. L'évaluation portera sur l'élément de compétence figurant dans la description des activités pédagogiques. Cette compétence, ainsi que sa pondération dans l'évaluation de cette unité, apparaît au tableau 7.1.

Activités et éléments de compétence		*Validation d'APP*	*Rapport d'APP*	*Examen sommatif*	*Examen final*
1	Compétence 1	10	20	55	65
1	Compétence 2	10	20	55	65
	Total : GIF390	20	40	110	130

TABLEAU 7.1 Évaluation de l'unité d'APP

Compétence 1 : Comprendre et appliquer les techniques essentielles aux systèmes distribués.
Compétence 2 : Comprendre, concevoir et déployer des systèmes en nuage.

7.2 Qualités de l'ingénieur

La grille d'indicateurs utilisée aux fins de l'évaluation est donnée au tableau 7.2. Il est à noter qu'un niveau d'atteinte d'une qualité et d'un indicateur dans cette grille n'a pas la même signification qu'un niveau d'atteinte d'une compétence dans le tableau 7.1. Cela est normal, un indicateur, une qualité et une compétence sont des éléments d'évaluation différents.

Tableau 7.2 Grille d'indicateurs utilisée pour les évaluations

Indicateur	Qualité	Aucun (N0)	Insuffisant (N1)	Seuil (N2)	Cible (N3)	Excellent (N4)
Démontrer, à un niveau universitaire, l'acquisition de connaissances en mathématiques	Q01.1	Ne résout pas ou très peu de problèmes mathématiques en génie	Résout correctement peu de problèmes mathématiques en génie	Résout correctement certains des problèmes mathématiques en génie	Résout aisément les problèmes mathématiques en génie	Résout aisément et efficacement les problèmes mathématiques en génie
Démontrer, à un niveau universitaire, l'acquisition de connaissances en sciences naturelles	Q01.2	N'applique pas ou très peu de concepts fondamentaux en sciences naturelles	Applique correctement peu de concepts fondamentaux en sciences naturelles	Est capable d'appliquer correctement certains des concepts fondamentaux en sciences naturelles	Applique aisément les concepts fondamentaux en sciences naturelles	Applique aisément et efficacement les concepts fondamentaux en sciences naturelles
Démontrer, à un niveau universitaire, l'acquisition de connaissances en sciences du génie	Q01.3	N'applique pas ou très peu de concepts fondamentaux en sciences du génie	Applique correctement peu de concepts fondamentaux en sciences du génie	Est capable d'appliquer correctement certains des concepts fondamentaux en sciences du génie	Applique aisément les concepts fondamentaux en sciences du génie	Applique aisément et efficacement les concepts fondamentaux en sciences du génie
Élaborer une procédure de résolution	Q02.2	Élabore une procédure de résolution de problème incomplète ou inappropriée	Élabore une procédure de résolution de problème minimalement appropriée	Élabore une procédure convenant à la résolution du problème, sans être parmi les meilleures	Élabore une procédure générale de résolution appropriée	Élabore une procédure ingénieuse de résolution
Appliquer la procédure de résolution	Q02.3	Est incapable de mettre en uvre la procédure de résolution choisie	Commet des erreurs mineures dans l'application de la procédure de résolution choisie	Commet peu d'erreurs mineures dans l'application de la procédure de résolution choisie	Applique correctement la procédure de résolution choisie	Applique efficacement et rigoureusement la procédure de résolution choisie
Analyser et interpréter les résultats obtenus	Q02.4	Est incapable d'analyser les résultats obtenus et d'en identifier les limites et la portée	Est capable d'analyser partiellement les résultats obtenus et d'en identifier les limites et la portée	Est capable d'analyser sommairement les résultats obtenus et d'en identifier les limites et la portée	Est capable d'analyser correctement les résultats obtenus et d'en identifier les limites et la portée	Est capable d'analyser correctement les résultats obtenus et témoigne d'une compréhension fine de leurs limites et de leur portée
Rechercher plusieurs solutions et en sélectionner une	Q04.3	Identifie peu de solutions et éprouve de la difficulté à sélectionner celle qui semble convenir	Identifie quelques solutions et sélectionne celle qui semble convenir, sans toutefois en faire la validation	Identifie plusieurs solutions, fait une analyse sommaire et sélectionne celle qui semble être la meilleure. Valide sommairement le potentiel de la solution retenue	Identifie plusieurs solutions et en crée de nouvelles, fait une analyse critique, basée sur des critères de sélection pertinents et à l'aide d'outils servant à la prise de décision. Valide le potentiel de la solution retenue	Identifie plusieurs solutions et en crée de nouvelles, fait une analyse critique et applique un processus rigoureux et rationnel à l'aide d'outils servant à la prise de décision. Valide le potentiel de la solution retenue et cherche à l'améliorer
Sélectionner les techniques, ressources et outils appropriés pour réaliser une tâche donnée	Q05.1	Sélectionne des techniques, ressources et outils qui ne sont pas appropriés pour réaliser une tâche donnée	Sélectionne des techniques, ressources et outils qui sont minimalement appropriés pour réaliser une tâche donnée	Sélectionne les techniques, ressources et outils appropriés, sans toutefois pouvoir justifier ses choix	Sélectionne les techniques, ressources et outils appropriés en pouvant justifier ses choix (en connaît la portée et les limites)	Sélectionne les techniques, ressources et outils appropriés en pouvant justifier ses choix (en connaît la portée et les limites), de même qu'en pouvant inférer la nature du travail à accomplir ou les données à colliger

8 PRATIQUE PROCÉDURALE

graphicx

But de l'activité

— Cette activité a pour but de vous familiariser avec les concepts et les architectures de systèmes répartis, de même qu'avec les technologies de conteneurisation.

8.1 EXERCICES PRÉPARATOIRES

P.P1 Questions.

Répondez aux questions suivantes avant de vous présenter au Procédural.

1. Qu'est-ce qu'une application centralisée ?
2. Qu'est-ce qu'une application distribuée ?
3. Qu'est-ce qu'une liaison synchrone ?
4. Qu'est-ce qu'une liaison asynchrone ?
5. Quelle est la différence entre une machine virtuelle (par exemple, VMware) et la virtualisation par conteneurs ?

8.2 EXERCICES

P.E1 Transparence dans le paradigme client-serveur

Un message de demande de données unique est envoyé par le client au serveur d'application, qui est en contact avec deux bases de données distinctes. Le serveur d'application utilise le contenu du message pour créer une paire de messages de type requête, un pour chaque base de données spécifique.

Les réponses à ces messages sont renvoyées au serveur d'applications par les deux bases de données. Elles sont ensuite transmises par le serveur d'applications au client, en deux réponses distinctes, l'une par base de données.

1. Faire un schéma pour décrire la situation
2. Expliquer les implications en matière de transparence de cette décision de conception.
3. Quelles seraient les conséquences de la combinaison des deux réponses (une réponse en provenance de chacune des bases de données) en une seule réponse transmise au client ?

P.E2 Couplage des composants.

Identifier le(s) type(s) de couplage se produisant dans chacun des scénarios suivants :

1. Une application dans une architecture en couche et fonctionnant en utilisant le *middleware*
2. Un prototype du système à trois niveaux qui utilise des adresses établies et figées dans le code (*hardcoded*) pour identifier les emplacements des composants.
3. Une demande peer-to-peer dans lequel les composants se connectent les uns aux autres lors de sa découverte dans le réseau local (le processus de découverte est automatique, basé sur l'utilisation de messages de diffusion)

P.E3 Identifier l'hétérogénéité.

Identifier les classes d'hétérogénéité qui peuvent se présenter dans chacun des scénarios suivants :

1. Un laboratoire informatique avec des ordinateurs achetés en lots sur une période de deux ans, installés avec le système d'exploitation Windows et la même suite d'applications logicielles sur tous les des ordinateurs.
2. Un petit système de réseau d'entreprise comprenant quelques ordinateurs de bureau différents, exécutant différentes versions du système d'exploitation Windows et quelques ordinateurs portables fonctionnant également avec le système d'exploitation Windows.
3. Un système qui prend en charge une application commerciale client-serveur. Un puissant ordinateur exécutant le système d'exploitation Linux est utilisé pour héberger le processus serveur. Les utilisateurs accèdent au service par le biais d'ordinateurs de bureau à faible coût de fonctionnement du système d'exploitation Windows.

P.E4 Transparence de concurrence

1. Quel est le principal défi quand on veut faciliter l'accès simultané aux ressources partagées ?
2. Comment un système transactionnel contribue à la transparence de la concurrence ?
3. Expliquez les proprietés ACID d'une transaction.

P.E5 Transparence de localisation

1. Pourquoi la transparence de localisation est l'une des exigences les plus courantes des applications distribuées ?
2. Comment un service de nom contribue à la réalisation de la transparence de localisation ?
3. Expliquez le principe d'un serveur de *DNS*

P.E6 Transparence de réplication

1. Quelles sont les principales motivations pour la réplication des données et/ou des services ?
2. Quel est le principal défi lors de l'implémentation de la réplication ?
3. Expliquez le fonctionnement du protocole en deux phases *Two Phases Commit*
4. Comment le protocole de la validation en deux phases (*Two Phases Commit*) contribue à la réalisation des mécanismes de réplication robustes ?

P.E7 Microkernel et Microservices

Quelle est la différence entre les modèles d'architecture micro-noyau (*microkernel*) et micro-services

P.E8 Technologies sous-jacentes aux conteneurs

1. Quelles sont les trois éléments sous-jacents du système d'exploitation Linux qui ont permis l'émergence des conteneurs ?
2. Quels sont leurs rôles respectifs ?
3. Quelle(s) fonctionnalité(s) additionnelle(s) facilite(nt) l'utilisation des conteneurs ?

9 PRATIQUE EN LABORATOIRE 1

Dans cette activité, on collabore par équipe de deux.

But de l'activité

Le but de cette activité est de se familiariser avec les conteneurs et les technologies sous-jacentes. En particulier, nous explorerons tout d'abord les commandes et éléments sous-jacents à l'opération de conteneurs, pour ensuite utiliser la commande docker et créer facilement un certain nombre de conteneurs de complexité grandissante. L'objectif est d'à la fois comprendre les fondations des technologies de conteneurisation et d'en maîtriser l'usage isolé. Nous explorerons comment lier ensemble un groupe de conteneurs (l'orchestration) dans le deuxième laboratoire.

Description du laboratoire :

Au laboratoire, à l'aide des commandes 'id', 'unshare', 'lsns', 'docker' :

> Définir et utiliser un *nameshare*, identifier l'usager, explorer et utiliser un nouveau *nameshare* ;
> Définir et utiliser un *cgroup*, explorer et valider ce que permet un *cgroup* ;
> Utiliser *docker* pour créer automatiquement un conteneur :
> — Créer un nouveau conteneur à partir d'une image "vanille" ;
> — Configurer et créer un conteneur à partir d'une image existante ;
> — Définir un fichier *Dockerfile* pour personnaliser une image ;
> — Exécuter un conteneur à partir d'une image nouvellement créée ;
> — Explorer "l'historique" d'une image ;
> — Produire et utiliser un *volume* Docker ;
> — Lier et accéder à un répertoire local à partir d'un conteneur

9.1 Utilisation de commandes sous-jacentes aux conteneurs

— Référence utile pour la première question
— Autre référence utile pour la première question, en particulier le paragraphe 2.4.2

Dans cet exercice, vous utiliserez les commandes Linux *id, unshare, lsns* pour explorer les possibilités offertes par les services *namespace*, ainsi que les commandes pour explorer ce que permettent les services *cgroup*. Enfin, vous explorerez ce que permet un système de fichier en couches (*union file system*).

a. Démarrez la machine virtuelle App3S3i_Lubuntu.vmdk. L'usager est *app3s3i*, avec le mot de passe *app3s3i* (le mot de passe est tout en minuscules). Notez que le système est configuré pour ne pas demander le mot de passe lorsqu'on le démarre. Démarrez un terminal dans l'interface graphique (il y a un raccourci sur le bureau). Lorsque nécessaire, vous pouvez devenir *root* en utilisant la commande : *su root*, avec le mot de passe : *app3s3i*.

b. Lorsque vous êtes l'usager *app3s3i*, donnez la commande *id*. Ensuite, faites *su root*, et après avoir donné le mot de passe, utilisez de nouveau la commande *id*. Qu'est-ce qui est différent ? Pourquoi ?

c. Démarrez un nouveau terminal, (vous pourriez aussi revenir à l'usager *app3s3i* (à partir de l'usager *root*) en utilisant la commande *exit*, mais ne le faites pas). Observez la liste complète de processus s'exécutant sur la machine virtuelle, avec la commande *ps aux*. Vous devriez observer qu'il y a un grand nombre de processus qui s'y exécutent en parallèle. Utilisez la commande *id* pour valider que vous êtes bien l'usager *app3s3i*.

d. Définissez maintenant un nouveau *namespace* et démarrez un nouveau "shell", à l'aide de la commande :

```
unshare --user --pid --map-root-user --mount-proc --fork bash
```

e. Utilisez la commande *id*. Que s'est-il passé ?

f. Utilisez de nouveau la commande *ps aux*. Quelle est maintenant la liste des processus ? Que s'est-il passé ?

g. Utilisez la commande *lsns*, qui indique les caractéristiques du *namespace* courant.

h. Allez dans le premier terminal, et utilisez de nouveau la commande *lsns*. Que voyez-vous ? Pouvez-vous l'expliquer ?

i. Dans le premier terminal, utilisez la commande *systemd-cgls*, pour obtenir la liste des cgroups (groupes de contrôle) s'exécutant sur le système. Il y aura beaucoup d'information, alors vous pouvez utiliser la commande *less* et le signe de *pipe* (|) pour naviguer dans les lignes apparaissant à l'écran : *systemd-cgls | less*

j. Retournez dans le deuxième terminal, et utilisez la même commande *systemd-cgls*. Est-ce que l'information est différente ? Pouvez-vous expliquer pourquoi ?

9.2 Introduction à docker

Dans cet exercice, vous utiliserez les commandes docker pour explorer le fonctionnement de ce système de virtualisation et son impact sur le système d'exploitation.

a. Dans le premier terminal, observez l'état du système *docker*, à l'aide des commandes :

— *docker ps* (pour la liste des conteneurs qui sont en exécution)
— *docker ps -a* (pour la liste complète des conteneurs du système, incluant ceux qui sont arrêtés ou en pause)
— *docker images* (pour connaître la liste d'images disponibles sur le système local)

b. Dans le deuxième terminal, utilisez l'image *hello-world* pour démarrer un conteneur simple :

```
docker run hello-world
```

c. Reprenez les commandes de la partie (a). Les résultats sont-ils différents ?
d. Retirez le conteneur inutilisé, avec la commande

```
docker rm nom-du-conteneur
```

ou

```
docker rm ID-du-conteneur
```

où le nom du conteneur apparaît à la fin de la ligne, et l'Identificateur du conteneur apparaît au tout début de la ligne de la liste de conteneurs obtenue par la commande *docker ps -a*.
e. Retirez l'image *hello-world* :

```
docker rmi hello-world
```

9.3 Utilisation d'un conteneur de l'intérieur

Dans cet exercice, vous utiliserez l'image *alpine* pour explorer l'intérieur d'un conteneur et observer son impact sur le système hôte.

a. Démarrez un conteneur interactif avec l'image *alpine* :

```
docker run -it alpine sh
```

b. Vous avez maintenant un "*shell*" qui s'exécute à l'intérieur du conteneur
c. Déterminez qui est l'usager avec les commandes *id* ou *whoami*

d. Pouvez-vous accéder au réseau à partir du conteneur ? Vérifier avec la commande *ping google.ca*. Utilisez CTL-C (les touches "contrôle" et "c" simultanément) pour arrêter le processus *ping*

e. Vérifiez quels processus s'exécutent, avec *ps aux*

f. Dans l'autre terminal, utilisez la commande *systemd-cgls|less*. Vous devriez observer une ou deux entrées pour *docker*

g. À partir du terminal où vous avez accès à l'intérieur du conteneur qui s'exécute avec *alpine*, installez le programme *fortune* : *apk add fortune* (sous *Alpine*, on installe à l'aide de la commande *apk*, l'équivalent de *apt* ou *apt-get* sous *Ubuntu*).

h. Vérifier que le programme est bien installé, en donnant la commande *fortune*. Vous devriez voir apparaître une vérité profonde sortie tout droit d'un biscuit chinois...

i. Sortez du conteneur *alpine* : *exit*. La commande *docker ps -a* vous indiquera que le conteneur est arrêté, mais toujours présent. Redémarrez le conteneur, en utilisant *docker start nom-du-conteneur* ou *docker start ID-du-conteneur*

j. Pour entrer de nouveau dans le conteneur, utiliser *docker attach nom-du-conteneur* ou *docker attach ID-du-conteneur*. Vous aurez de nouveau accès au *shell* à l'intérieur du conteneur *alpine*

k. Sortez du conteneur *alpine*, et effacez-le du système. Vous ne devriez plus le voir en faisant la liste complète de tous les conteneurs.

l. Démarrez de nouveau un conteneur avec l'image *alpine*. Pouvez-vous exécuter la commande *fortune* ? Que s'est-il passé ? Installez de nouveau *fortune*. Puis, dans l'autre terminal, démarrer un conteneur *alpine*. Avez-vous accès au programme *fortune* ? Pouvez-vous expliquer ce qui se passe ?

m. Arrêtez les deux conteneurs et effacez-les.

n. Redémarrez un conteneur *alpine*, cette fois avec la commande :

```
docker run -it ––rm alpine sh
```

Sortez du conteneur avec la commande *exit*. Est-ce que le conteneur *alpine*est toujours présent ?

o. Redémarrez un conteneur *alpine, avec la commande :* docker run -it –rm –name app3s3i alpine sh (il y a deux "-" avant *name*).

p. Dans l'autre terminal, utilisez la commande *docker ps*. Vous pouvez observer le nom du conteneur que vous venez tout juste de démarrer. Retournez dans le premier terminal et mettez fin au conteneur.

9.4 Configuration d'une image à l'aide d'un *Dockerfile*

Dans cet exercice, vous utiliserez l'image *alpine* pour configurer une image personnalisée, que vous utiliserez par la suite.

a. Complétez le fichier *Dockerfile* se trouvant dans le répertoire Labo1. Si vous voulez obtenir le chemin complet vers l'application *fortune*, vous pouvez refaire l'exercice précédent où vous avez installé interactivement l'application *fortune* dans le conteneur *alpine*. Utilisez la commande *which fortune* pour connaître le chemin complet vers cette application.

b. Créez une image locale à partir du *Dockerfile* :

```
docker build -t fortune .
```

c. Utilisez l'image créée :

```
docker run --rm --name test_fortune fortune
```

Vous devriez voir apparaître un texte issu de l'application *fortune*

d. Vous pouvez aussi utiliser l'image pour démarrer un conteneur interactif :

```
docker run -it --rm --name test_fortune fortune sh
```

Vous pouvez appeler l'application *fortune* de l'intérieur du conteneur. Arrêtez le conteneur.

9.5 Observation d'une image locale

Dans cet exercice, vous utiliserez l'image locale créée au numéro précédent. Nous observerons la stucture de cette image, ainsi que la structure du conteneur qui l'utilise

a. Démarrez un conteneur en utilisant l'image *fortune* que vous avez créée auparavant.

b. Dans un autre terminal, regardez ce qui se trouve dans l'image *fortune* :

```
docker image inspect fortune
```

Vous verrez :

— son identificateur (une signature obtenue à l'aide de *SHA256*) ,

— l'auteur (APP3S3i) ,

— la commande par défaut (l'entrée "Cmd") ,
— le type de processeur ciblé (l'entrée "Architecture") ,
— les paramètres du système de fichiers à union
  — l'entrée "GraphDriver":"Data",
  — l'entrée "GraphDriver":"Name" (le pilote utilisé pour le système de fichiers à union)

Dans l'entrée "GraphDriver":"Data", vous verrez quatre (4) entrées :
— "LowerDir"
— "MergeDir"
— "UpperDir"
— "WorkDir"

Ces quatre entrées pointent vers des sous-répertoires de */var/lib/docker/overlay2*. Observez bien le répertoire pointé par l'entrée "LowerDir", et comparez-le avec les répertoires des trois autres entrées. Les trois autres entrées pointent vers les sous-répertoires du même répertoire *.../overlay2* :
— *merged*
— *diff*
— *work*

Pour sa part, l'entrée "LowerDir" pointe vers le sous-répertoire *diff* d'un autre sous-répertoire de *.../overlay2*.

Regardez maintenant ce qui se trouve dans l'image *alpine*, et comparez les entrées "MergeDir", "UpperDir" et "WorkDir". Que remarquez-vous ? Pourquoi n'y a-t-il pas d'entrée "LowerDir" ? Pouvez-vous expliquer ce qui se passe ?

c. Utilisez la commande :

```
docker image history fortune
```

Quelle information est accessible par cette commande ? Utilisez la même commande, pour l'image *alpine* cette fois. Pouvez-vous expliquer ce que vous observez ?

9.6 Utilisation d'un répertoire externe dans un conteneur

Dans cet exercice, vous utiliserez allez lier un conteneur à un répertoire externe et y donner accès.

a. Allez dans le répertoire de base de l'usager *app3s3i*, avec la commande *cd*
b. Créez un répertoire de test : *mkdir test*
c. Créez un fichier bidon dans le répertoire de tests : *touch test/bidon*

d. Démarrez un conteneur en lui donnant accès au répertoire que vous venez de créer :

```
docker run -it --rm --name app3s3i --mount

type=bind,source="$(pwd)"/test,target=/media/mestrucs fortune sh
```

(Attention, le paramètre *mount* est prédécé de deux traits d'union "-")

e. Dans le conteneur, observez ce qui se trouve dans le répertoire */media/mestrucs*
f. Créez un autre fichier bidon à partir du conteneur : *touch /media/mestrucs/bidon2*
g. Sortez du conteneur et observez le contenu du répertoire *test*. Il devrait maintenant contenir deux fichiers.

9.7 Création d'un volume et utilisation dans un conteneur

a. Créez un volume docker : *docker volume create mon-volume*
b. Vérifiez qu'il est bien présent : *docker volume ls*
c. Démarrez un conteneur en le liant à ce volume :

```
docker run -it --rm --name app3s3i --mount

source=mon-volume,target=/media/mestrucs fortune sh
```

d. Dans le conteneur, observez le contenu du volume :

```
ls /media/mestrucs
```

Il ne devrait rien y avoir, puisqu'il s'agit d'un tout nouveau volume

e. Créez un fichier bidon : *touch /media/mestrucs/bidon*, et vérifiez qu'il s'y trouve
f. Sortez du conteneur, et redémarrez un nouveau conteneur, en liant maintenant le volume sous un autre répertoire interne :

```
docker run -it --rm --name app3s3i --mount

source=mon-volume,target=/media/autreDir fortune sh
```

g. Dans le conteneur, observez le conteneu du volume : *ls /media/autreDir*. L'information attendue devrait s'y trouver
h. Quelle est la différence entre l'utilisation d'un volume et l'utilisation d'un répertoire du système hôte ? Quels sont les avantages et inconvénients de chacun ?

10 PRATIQUE EN LABORATOIRE 2

Dans cette activité, on collabore par équipe de deux.

But de l'activité

Le but de cette activité est de se familiariser avec l'orchestration des conteneurs. Nous explorerons tout d'abord la commande *docker compose*, en configurant et en utilisant des fichiers *.yaml* La commande *docker compose* permet de démarrer facilement un ensemble de conteneurs qui communiquent entre eux, tous les conteneurs étant déployés sur le même ordinateur physique.

Par la suite, nous utiliserons le système *kubernetes*, qui permet de faire de l'orchestration de conteneurs sur un ensemble de machines physiques distinctes. Pour les besoins du laboratoire, nous utiliserons une version locale de *kubernetes*, qui se nomme *minikube*.

Au laboratoire, à l'aide des commandes 'docker compose' et 'minikube' :

- Utiliser des fichiers *.yaml* existants et démarrer quelques systèmes de conteneurs qui collaborent.
- Modifier les fichiers *.yaml* pour modifier le comportement des systèmes étudiés.
- Utiliser des fichiers *.yaml* existants et utiliser minikube pour démarrer quelques systèmes.
- Modifier les fichiers *.yaml* et augmenter les fonctionnalités des systèmes démarrés.

Avant de débuter le laboratoire :

a. Aller chercher le fichier labo2.tgz sur le site de l'unité d'APP
b. Décompresser le fichier dans le répertoire */home/app3s3i* à l'aide de la commande *tar xvfz labo2.tgz*
c. Les activités du deuxième laboratoire se trouvent dans les répertoires *E1*, *E2*, etc.

10.1 Utilisation de fichiers *.yaml* existants à l'aide de la commande *docker compose*

a. Aller dans le répertoire */home/app3s3i/Labo2/E1*
b. Ouvrir le fichier *compose.yaml* qui s'y trouve. *Sublime text* pourrait être utile... Pouvez-vous déterminer (à haut niveau) ce que ce système va faire ? Deux conteneurs vont être démarrés, l'un avec *grafana*, l'autre avec *prometheus*. Observez bien les paramètres qui sont utilisés dans le fichier *compose.yaml*.
c. Utiliser *docker compose up -d* pour démarrer le tout.

d. Utilisez *docker ps* pour vous assurer que les conteneurs sont bien démarrés. Observez les ports utilisés.
e. Démarrez Firefox et connectez-vous sur *localhost*, sur l'un (et l'autre) des deux ports appropriés . Lorsque vous vous connecterez sur *grafana*, on vous demandera un mot de passe. Pouvez-vous le trouver ? Vous avez toute l'information nécessaire pour le faire...
f. Arrêtez ce système de conteneurs en utilisant la commande *docker compose down*.
g. Modifiez le fichier *compose.yaml* pour modifier les ports accessibles par votre fureteur. Pouvez-vous aussi modifier l'usager et le mot de passe du conteneur *grafana* ?
h. Redémarrez le système. Vérifiez que vous pouvez maintenant accéder aux nouveaux ports
i. Pouvez-vous accéder directement aux deux conteneurs du système ? La commande *docker exec* devrait vous le permettre. Attention : le **shell** *bash* n'est pas toujours disponible. Cependant, le *shell sh* est habituellement présent.
j. Observez quelles tâches s'exécutent dans chacun des conteneurs du système. Quelle commande pouvez-vous utiliser ?
k. Utilisez la commande *ifconfig* dans chacun des conteneurs. Que vous dit cette commande ?

10.2 Configuration et utilisation de *minikube*

Pour débuter cette partie, vous devez installer *minikube*. Les instructions qui suivent sont très inspirées de ce qu'on retrouve **ici**

a. Pour commencer, installez minikube et kubectl : Débutez en vous assurant que vous êtes l'usager *app3s3i*, dans le répertoire */home/app3s3i*. Puis, utilisez les commandes suivantes, qui sont disponibles dans le répertoire E2 du deuxième laboratoire. Vous pouvez aussi utiliser le fichier *commandes_pour_kuberenetes.txt* qui se trouve sur le site de l'APP.
   - Mise à jour de la liste apt-get : *sudo apt-get update -y*
   - Mise à jour des applications apt-get : *sudo apt-get upgrade -y*
   - Installation de curl : *sudo apt-get install curl*
   - Installation de la librairie d'accès https : *sudo apt-get install apt-transport-https*
   - Copie de minikube à partir du site officiel :
     wget https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
   - Copie de l'exécutable au bon endroit :
     sudo cp minikube-linux-amd64 /usr/local/bin/minikube

- Assure que l'exécutable a les droits d'exécution :
  sudo chmod 755 /usr/local/bin/minikube
- Permet à l'usager app3s3i d'exécuter docker :
  sudo usermod -aG docker app3s3i && newgrp docker
- Vérification de minikube : *minikube version*
- Installation de kubectl (CLI) : *curl -LO https://storage.googleapis.com/kubernetes-release/release/ùcurl -s https ://storage .googleapis.com/kubernetes-release/release/stable.txt/ùbin/*
- Assure que kubectl est exécutable : *chmod +x ./kubectl*
- Copie de kubectl au bon endroit : *sudo mv ./kubectl /usr/local/bin/kubectl*
- Vérification de kubectl : *kubectl version -o json*
- Démarrage de minikube (long la première fois) : *minikube start*
- Vérification de la configuration de kubectl : *kubectl config view*
- Information sur l'infrastructure déployés par kubectl : *kubectl cluster-info*
- Voir les noeuds utilisés (vous verrez minikube) : *kubectl get nodes*
- Entrée dans minikube par *ssh* : *minikube ssh* (pour sortir : *exit*)
- État de minikube : *minikube status*
- Arrêt de minikube : *minikube stop* (repartir avec *minikube start*)
- Liste des *add-ons* de minikube : *minikube addons list*
- Accès au *dashboard* par le fureteur : *minikube dashboard*

b. Faire la partie 4 (Deploy applications) de l'exercice qui se trouve sur la page web accessible par l'hyperlien

11 VALIDATION AU LABORATOIRE

Le but de cette activité est de vous permettre de valider la solution proposée à la problématique de cette unité.

- Vous devriez avoir réussi à faire fonctionner les 6 étapes de la problématique.
- Vous serez questionnés sur ce que vous avez réalisé, en vous demandant d'en expliquer le fonctionnement
- Vous aurez à réserver une plage horaire pour rencontrer le tuteur au cours de la période réservée.

Avec le formateur, vous devrez :

- Présenter votre solution à l'une des six (6) étapes de la problématique (déterminée par le tuteur)
- Expliquer comment elle fonctionne
- Répondre aux questions touchant les technologies sous-jacentes