

2ème année cycle supérieure(2CS)

Option: Systèmes Informatiques (SQ)

Thème:

Mise en place et administration d'un cluster Kubernetes
pour la Gestion des TP au niveau de l'ESI

Equipe N° 10:

- BELKESSA Linda (CE)
- DJABELKHIR Sarah
- CHELLAT Hatem
- BENHAMADI Yasmine
- BOUZOUAD Meriem
- LAMDANI Wilem

Encadré par:

- Mr AMROUCHE Hakim
- Mr SEHAD Abdenour
- Mr HAMANI Nacer

Résumé

Dans Le livrable précédent, nous avons traité l'aspect théorique de notre solution exposant les différentes technologies que nous envisageons à utiliser, tout en analysant leurs avantages et les inconvénients de chacune, nous sommes arrivés à la conclusion d'utiliser docker combiné avec docker.

Durant cette phase nous nous intéressons au processus de mise en place et comment Dockeriser une application simple et une autre plus complexe. Docker permet de créer des déploiements à construire une seule fois et à exécuter partout, allant des ordinateurs portables locaux aux serveurs de production et tout ce qui se trouve entre les deux. Malheureusement, tirer parti de cette puissance nécessite de contourner certaines limitations à la fois dans Docker et potentiellement dans l'architecture de l'application choisie.

Mots Clés : containers, orchestration, Kubernetes, Docker

Table des matières

1	Introduction	3
2	Problématique générale	4
3	Mise en place des conteneurs avec Docker	5
3.1	Linux Terminal	5
3.1.1	Justification du choix de l'application :	5
3.1.2	Mise en place :	5
3.1.3	Configuration et test :	8
3.1.4	Exploitation des images Docker :	8
3.2	Packet tracer	9
3.2.1	Justification du choix de l'application :	9
3.2.2	Problèmes et défis rencontrés :	9
3.2.3	Étapes de la mise en place :	9
3.2.4	Exploitation des images Docker :	10
4	Résultats des tests	13
4.1	Linux Terminal	13
4.2	Packet Tracer	16
5	Conclusion	21
6	Annexe : Commandes Docker Utilisées et Description des Tests	22
6.1	Introduction :	22
6.2	Commandes de mise en place du DockerFile :	22

Liste des tableaux

3.1	Listes des commandes utilisées pour la mise en place de l'App Linux Terminal	7
4.1	Tests de fonctionnement Linux Terminal	15
4.2	Tests de fonctionnement Packet Tracer	20
6.1	Listes des commandes spécifiques utilisées dans un dockerfile	23

1. Introduction

Imaginons une situation où un enseignant qui a passé toute une nuit à préparer un guide d'installation et d'utilisation d'une application, teste le processus sur son ordinateur et fonctionne à merveille ! Le lendemain, les étudiants téléchargent les ressources et tentent d'installer les outils. Mais malheureusement, l'application ne fonctionne pas pour certains ! Une explication à ce problème est que l'environnement d'exécution de ces étudiants est inadéquat. Deux systèmes peuvent avoir des différences de version sur les dépendances ou encore des bibliothèques manquantes.

Avec Docker, nous sommes entrés dans l'ère des architectures à base de "conteneur". On parle aussi de "virtualisation de niveau système d'exploitation" par opposition aux technologies à base d'hyperviseurs (comme VMWare ou VirtualBox) qui cherchent à émuler un environnement matériel. Contrairement à une VM, un conteneur n'embarque pas un système d'exploitation complet. Il repose pour l'essentiel sur les fonctionnalités offertes par l'OS sur lequel il s'exécute. L'inconvénient de cette approche est qu'elle limite la portabilité du conteneur à des OS de la même famille (Linux dans le cas de Docker). Cette approche, en revanche, a l'avantage d'être beaucoup plus légère (les conteneurs sont nettement plus petits que les VM et plus rapides au démarrage) tout en offrant une isolation satisfaisante en termes de réseau, de mémoire ou de système de fichiers.

L'objet de cet ouvrage est d'offrir une approche à 360 degrés de l'écosystème Docker. Docker, plus qu'une technologie, est en effet aujourd'hui un écosystème de solutions fourmillantes : Docker Compose, Kubernetes (et ses nombreuses implémentations chez les leaders du cloud public), ou encore Docker Swarm. Autour du runtime (moteur d'exécution) qui exécute les conteneurs (le Docker Engine), des outils complémentaires visent à conditionner, assembler, orchestrer et distribuer des applications à base de conteneurs. Timidement, des initiatives de standardisation voient le jour, laissant espérer une meilleure interopérabilité que celle qui prévaut aujourd'hui dans le domaine de la virtualisation matérielle.

Les dockers viennent proposer une solution à notre problème principal (et au problème d'hétérogénéité) en général. Il s'agit d'une plateforme qui permet d'exécuter le code à l'intérieur d'un conteneur indépendamment de la machine sur laquelle on est ! Un conteneur ressemble à une machine virtuelle sauf qu'il n'inclut pas un système d'exploitation ce qui lui permet de s'exécuter en quelques secondes et d'être beaucoup plus léger. Mais, cette plateforme a des insuffisances. En effet, la création des conteneurs nécessite la saisie de plusieurs lignes de commandes.

Durant cette partie du projet, nous exploitons plusieurs aspects techniques notamment la mise en place détaillée des images dockers pour certaines applications utilisées très souvent dans l'apprentissage pratique de plusieurs modules, ensuite expliquer comment le client pourra les utiliser tout en détaillant comment nous avons procédé pour faciliter cette tâche et l'optimiser.

2. Problématique générale

Après avoir fait une analyse critique de l'existant qui nous a ramené à considérer la technologie de conteneurisation pour remédier aux problèmes exprimés précédemment, accompagnée d'une étude théorique approfondie, à travers laquelle nous avons exposé les différents aspects à contempler (principes, avantages et inconvénients, outils...) avant d'entamer cette phase.

Maintenant, il nous a été demandé de choisir au moins deux applications dans le cadre des TPs, mettre dans un conteneur les ressources matérielles, systèmes de fichiers, réseau, processeur, mémoire vive, etc, nécessaires à leur exécution. Les différentes étapes essentielles dans la concrétisation de cette phase devront être expliquées d'une manière détaillée avec la justification des différents choix effectués.

L'objectif principal de cette étape est de déployer les applications conteneurisées pour être exploitées en montrant comment un utilisateur peut s'en servir.

3. Mise en place des conteneurs avec Docker

3.1 Linux Terminal

3.1.1 Justification du choix de l'application :

Le choix de l'application du terminal linux repose sur la grande importance qu'elle offre aux étudiants de 1ere classe préparatoire au sein de l'école où les étudiants seront amenés à faire les tps en présentiel et qui seront notés par la suite et ça permet de couvrir toutes les commandes nécessaire pour effectuer les tps et donc peuvent exploiter le système sans avoir à installer des machines virtuelles et les configurer.

En général, les étudiants en première année ont tendance à utiliser windows 10, windows 11 et galèrent généralement avec l'installation et la configuration d'une machine virtuelle à l'aide d'un logiciel de virtualisation comme VirtualBox ou VMware, donc leur donner accès à un outil rapidement utilisable et exploitable sans avoir à perdre beaucoup de temps et passer directement à la pratique et reproduction des scénarios des TPs, du coté des enseignants, ceci leur permettra aussi de mettre à jour des installations de commandes ou des LAB préparé (par exemple).

3.1.2 Mise en place :

Tout d'abord pour créer le conteneur du terminal linux on aura besoin du fichier docker (dockerfile) dans lequel on définit l'image de base à l'aide de l'instruction FROM dans ce cas là l'image de base est ubuntu :18.04. On utilise la commande ENV DEBIAN_FRONTEND non interactive pour ne pas interagir avec le système lors de l'installation via apt-get c'est-à-dire qu'on choisit les réponses par défaut sans avoir à interrompre l'installation ou les mises à jour du système. c'est un mode parfait pour les installations automatiques.

La deuxième étape consiste à utiliser la commande RUN pour exécuter les commandes et installer les dépendances nécessaires à la création de notre conteneur. la création des images se fait à l'aide des couche avec chaque commande RUN il faut créer une autre couche donc l'idéal est de ne pas utiliser plusieurs commandes de type RUN dans le fichier docker Puis la dernière étape consiste à utiliser la commande CMD pour définir la commande par défaut pour l'exécution de notre conteneur .

Commande	Rôle	Exemple
ADD	Définit les fichiers à copier du système de fichiers hôte vers le conteneur	ADD ./local/config.file /etc/service/config.file
CMD	Il s'agit de la commande qui s'exécutera au démarrage du conteneur	CMD ["nginx", "-g", "daemon off;"]
ENTRYPOINT	Définit l'application par défaut utilisée chaque fois qu'un conteneur est créé à partir de l'image. S'il est utilisé conjointement avec CMD, vous pouvez supprimer l'application et simplement y définir les arguments	CMD Hello World! ENTRYPOINT echo
ENV	Définissez/modifiez les variables d'environnement dans les conteneurs créés à partir de l'image.	ENV VERSION 1.0
EXPOSE	Définir les ports de conteneur à exposer	EXPOSE 80
FROM	Sélectionnez l'image de base pour créer la nouvelle image par-dessus	FROM ubuntu :latest
LABEL maintainer	Champ facultatif pour vous permettre de vous identifier en tant que mainteneur de cette image. Ceci n'est qu'une étiquette	LABEL maintainer = so- meone@xyz.xyz

RUN	Spécifiez les commandes pour apporter des modifications à votre image, puis aux conteneurs démarrés à partir de cette image. Cela inclut la mise à jour des packages, l'installation de logiciels, l'ajout d'utilisateurs, la création d'une base de données initiale, la configuration de certificats, etc. Ce sont les commandes que vous exécuterez sur la ligne de commande pour installer et configurer votre application. C'est l'une des directives dockerfile les plus importantes.	<pre>RUN apt-get update && apt-get upgrade -y && apt-get install -y nginx && rm -rf /var/lib/apt/lists/*</pre>
USER	Définissez l'utilisateur par défaut, toutes les commandes seront exécutées dans n'importe quel conteneur créé à partir de votre image. Il peut s'agir d'un UID ou d'un nom d'utilisateur	<pre>USER docker</pre>
VOLUME	Créer un point de montage dans le conteneur le reliant aux systèmes de fichiers accessibles par l'hôte Docker. Les nouveaux volumes sont remplis avec le contenu préexistant de l'emplacement spécifié dans l'image. Il est particulièrement pertinent de mentionner que la définition de volumes dans un Dockerfile peut entraîner des problèmes.	<pre>VOLUME /var/log</pre>
WORKDIR	Définir le répertoire de travail par défaut définie au niveau de 'ENTRYPOINT' ou 'CMD'	<pre>WORKDIR /home</pre>

TABLE 3.1: Listes des commandes utilisées pour la mise en place de l'App Linux Terminal

3.1.3 Configuration et test :

Pour les commandes utilisées pour créer et exécuter le conteneur :

- `docker build -t bash/dockerized-bash .` : pour la création de l'image chaque commande dans le dockerfile crée une couche ce qui réduit le temps de mis à jour car si on ajoute une commande une ajoute une couche sans avoir à modifier le reste, le point indique que le dockerfile est présent dans le répertoire courant.
- `docker run -it --rm bash/dockerized-bash` : commande pour le démarrage du conteneur.

3.1.4 Exploitation des images Docker :

3.1.4.1 Première méthode : partage des fichiers de configurations

Les étudiants recevront des fichiers de configuration en particulier le Dockerfile.yaml par biaux d'un dossier drive partagé ou mail, ensuite utiliser les commandes nécessaires au build de l'image. Par contre ceci nécessite de guider les étudiants en leur envoyant les commandes à taper.

3.1.4.2 Deuxième méthode : partage des images directement grâce à Docker Hub

Pour se ramener à une solution permettant l'étudiant d'exploiter directement une image docker sans soucier de la partie build, l'enseignant pourra concevoir l'image elle-même et la mettre au niveau de docker hub (le site permettant de partager des projets et des images), par la suite un étudiant n'aura plus qu'à pull cette dernière et la lancer dans un conteneur.

Commandes pour mettre une image sur docker-hub (enseignant) :

- Vérifier les images : `docker images`
- S'authentifier sur docker hub (au niveau du terminal) : `docker login`
- Taguer l'image avec un nom :
`docker tag image_name:version repo_name/name_image:version`
si version est laissée vide alors elle va être latest par défaut
- Mettre l'image sur Docker Hub : `docker push name_repo/image_name`

Commandes pour télécharger l'image depuis docker-hub (étudiant) :

- `docker pull repo_name/image_name`
- `docker run -t -d container_name repo_name/image_name`

Et c'est bon ! l'étudiant pourra ainsi travailler dans le conteneur. Pour sauvegarder les changements et rendre l'image persistante, il suffit de faire un "docker commit", comme indiqué dans la partie test et résultats du rapport.

3.2 Packet tracer

3.2.1 Justification du choix de l'application :

Packet Tracer est un logiciel de CISCO, utilisé dans les TPs de 1CS et 2CS, permettant de construire un réseau physique virtuel et de simuler le comportement des différents protocoles sur ce réseau. L'étudiant construit sa topologie à l'aide d'équipements tels que les routeurs, les commutateurs ou des ordinateurs. Ces équipements doivent ensuite être reliés via des connexions (câbles divers, fibre optique). Une fois l'ensemble des équipements reliés, il est possible de configurer le réseau selon ce qui est demandé en TP.

Pendant l'année dernière et cette année, nous avons rencontré plusieurs problèmes, par exemple pour le projet réseau de l'année passée, les membres d'une même équipe peuvent avoir des versions différentes qui mènent à une perte de temps considérable. Aussi pendant le contrôle intermédiaire du module réseaux avancés de cette année, le même problème s'est reproduit pour quelques étudiants et par conséquent ils n'ont pas pu passer leur examen dans les meilleures conditions. De plus, les enseignants reçoivent les TPs par mail ce qui rend la procédure de correction plus difficile et ils peuvent même rater un livrable ou plus. D'où la nécessité de conteneuriser ce logiciel.

3.2.2 Problèmes et défis rencontrés :

Pendant la conteneurisation de Packet Tracer nous avons rencontrés plusieurs difficultés :

- **Les dépendances :** Le problème majeur que nous avons rencontré et qui nous a fait perdre énormément de temps est de trouver les packages nécessaires pour l'installation et démarrage corrects de Packet Tracer.
- **Les interactions lors de la création de l'image :** Pendant la création de l'image de packet tracer 8, le processus se bloque en attendant une entrée du clavier (la configuration du fuseau horaire et du clavier ...) qui ne peut pas être faite pendant la création de l'image, donc l'image ne peut pas être créée.

3.2.3 Étapes de la mise en place :

- **Decomposition :** Packet Tracer est un logiciel complexe qui a de nombreuses dépendances. D'abord, nous devons vérifier les dépendances nécessaires et les installer. Pour notre image (Version 8.1.1 de Cisco Packet Tracer) la commande suivante doit être exécutée pour les récupérer :

```
$ sudo dpkg-deb -I PacketTracer_8.1.1_amd64.deb
```

- **Sélection d'une image de base :** Il est inutile de réinventer la roue à chaque migration. Docker nous donne la possibilité de commencer à partir d'une image de base, dans notre cas on a choisis une image de base Ubuntu 20.04 qui est la version minimale.

- **Ajout du code :** Pour créer l'image, nous devons définir les étapes nécessaires à la construction de l'image dans un fichier Dockerfile. Nous commençons par définir l'image de base choisie (Ubuntu :20.04) avec la commande FROM. Ensuite, nous installons les dépendances avec les commandes que nous utiliserons lors de la création de notre image avec la commande RUN. Puis, nous créons l'utilisateur de notre conteneur (student). Par la suite, nous copions le fichier ".deb " de packet tracer dans l'image avec la commande COPY et nous installons Packet tracer. Finalement nous avons spécifié le nom d'utilisateur et le répertoire Home et la commande à exécuter lors du démarrage du conteneur.

Une fois l'image créée, nous pouvons l'ajouter au référentiel Docker Hub pour qu'elle puisse être utilisée par d'autres personnes.

- **Configuration et test :** Nous allons Configurer l'application conteneurisée : elle doit pouvoir se connecter automatiquement aux ressources externes telle que l'écran et les dossiers partagés, et pour cela nous utilisons le fichier docker-compose.yml pour partager la variable d'environnement DISPLAY de l'hôte avec le conteneur, partager le XServer de l'hôte avec le conteneur en créant un volume, Exécuter le conteneur avec le pilote de réseau hôte et pour les dossiers partagés nous créons un volume pour chacun d'entre eux.

Enfin, Le test de cette application se fait en exécutant un des TP du module réseaux avancés 2CS étudié dans 2CS (SIQ) pour garantir le bon fonctionnement des conteneurs pendant les séances TPs.

Pour qu'une application graphique fonctionne, nous devons disposer d'un XServer disponible dans chaque environnement de bureau Linux, mais dans un conteneur, nous n'avons aucun XServer, nous allons donc exécuter les commandes suivantes :

- Partager le XServer de l'hôte avec le conteneur en créant un volume :

```
--volume="\$HOME/.Xauthority:/root/.Xauthority:rw"
```

- Partager la variable d'environnement DISPLAY de l'hôte avec le conteneur :

```
--env="AFFICHER"
```

- Exécuter le conteneur avec le pilote de réseau hôte avec :

```
--net=hôte
```

3.2.4 Exploitation des images Docker :

3.2.4.1 Procédure d'installation :

Installation de Docker Compose : Docker Compose est un outil de ligne de commande permettant de gérer plusieurs conteneurs Docker. Il s'agit d'un outil permettant de créer des conteneurs isolés via le fichier YAML afin de modifier les services de l'application conteneurisée. L'installation de Docker Compose sur une machine Ubuntu se fait comme suit :

- Mettre à jour l'index des paquets : avant de procéder à l'installation du moteur Docker, il est recommandé de mettre à jour l'index des paquets de votre système d'exploitation pour le rendre plus sécurisé. Utilisez les deux commandes suivantes :

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

- Installer les paquets pré-requis : Une fois qu’on a mis à jour le système, on doit installer quelques paquets nécessaires avant d’installer Docker Compose (ou Docker tout simplement). Vous pouvez le faire avec une seule commande :

```
$ sudo apt-get install curl apt-transport-https ca-certificates  
software-properties-common
```

- Installer Docker Compose sur Ubuntu avec la commande :

```
$ sudo apt-get install docker-compose
```

- Vérifier l’installation avec la commande :

```
$ docker-compose --version
```

Le résultat de cette commande est :

```
docker-compose version [la version], build unknown
```

Une fois que Docker Compose est installé, il suffit d’exploiter l’image de l’application conteneurisée construite à partir des fichiers de configuration pour pouvoir l’utiliser, ce qu’on va développer dans la section suivante.

3.2.4.2 Comment un étudiant peut l’exploiter :

Une image Docker constitue un modèle immuable qui peut être employé de manière répétée pour créer des conteneurs Docker. L’image contient toutes les informations et dépendances requises pour exécuter un conteneur, y compris toutes les bibliothèques de programme basiques et interfaces utilisateur.

Le processus d’exploitation d’une application conteneurisée est le suivant :

- Pendant une séance de TP, un dossier est partagé avec les étudiants, contenant 3 fichiers principaux nécessaires au fonctionnement des conteneurs :
 - **Fichier Dockerfile** : un simple fichier texte qui contient les commandes qu’un utilisateur peut appeler pour assembler une image.
 - **Fichier docker-compose.yml** : un fichier définissant les différents services qui composent une application afin qu’ils puissent être exécutés ensemble dans un environnement isolé.
 - **Fichier d’installation de l’application.**

En plus d’un dossier qui contiendra les fichiers des TPs concernés (initialement vide). Ce dernier doit être déclaré comme partagé entre la machine hôte et le conteneur lorsqu’il sera exécuté dans le fichier **docker-compose.yml**, en l’insérant dans la section des volumes.

Ensuite, il suffit que l’étudiant crée l’image à partir des fichiers de configuration avec la commande suivante :

```
$ sudo docker compose build
```

Après avoir créé l'image Docker, il faut créer et démarrer le conteneur avec la commande :

```
$ sudo docker compose up
```

L'étudiant pourra à cette étape tirer avantage de l'application conteneurisée en ouvrant le fichier du TP et faire son travail tout simplement.

4. Résultats des tests

4.1 Linux Terminal

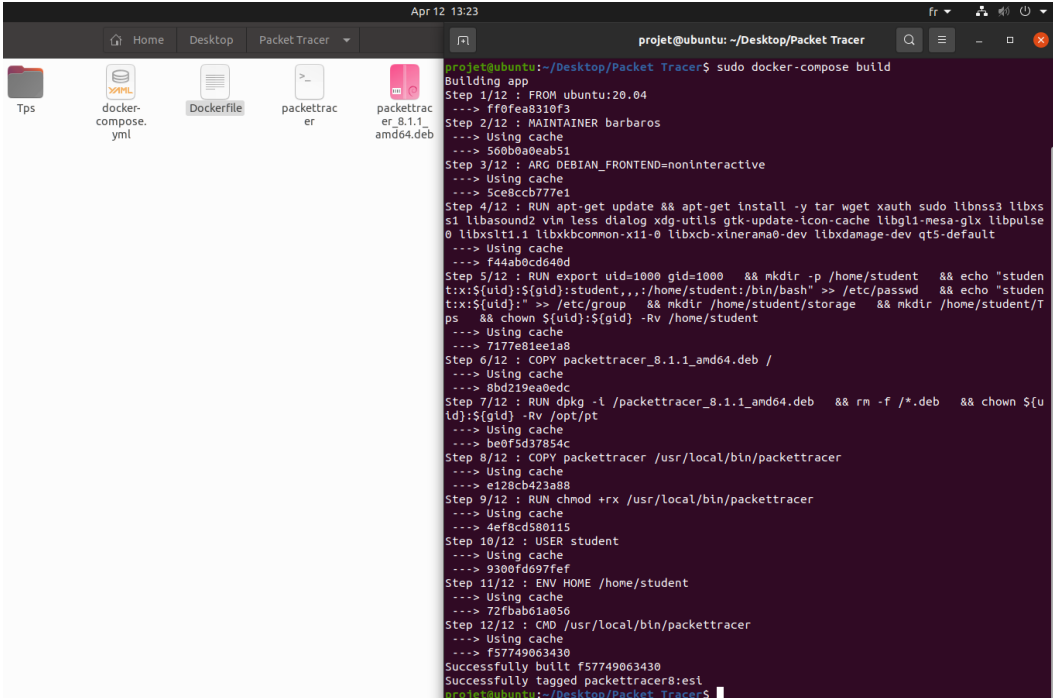
Test	Résultats obtenus
Créer l'image	<pre>root@ubuntu:/home/meriem/bash# docker build -t dockernized-bash . Sending build context to Docker daemon 2.048kB Step 1/5 : FROM ubuntu:18.04 18.04: Pulling from library/ubuntu 08a6abff8943: Pull complete Digest: sha256:982d72c16416b09ffd2f71aa381f761422085eda1379dc66b668653 Status: Downloaded newer image for ubuntu:18.04 --> f5cbcd4244ba Step 2/5 : MAINTAINER BARBAROS Team <support@barbaros.dz> --> Running in 6952c6b8b7ba Removing intermediate container 6952c6b8b7ba --> 14633db92e30 Step 3/5 : ENV DEBIAN_FRONTEND noninteractive --> Running in c537ea431ddd Removing intermediate container c537ea431ddd --> a20d99c1100a Step 4/5 : RUN apt-get -qq update && apt-get -qq install vim wget telnet make nano sudo && apt-get clean && rm</pre> <p>Explication : On commence par la commande : <code>docker build</code> qui crée l'image décrite par le dockerfile. On utilise l'option : <code>-t</code> pour spécifier le nom de l'image, dans notre cas c'est : <code>dockernized-bash</code>. Et on spécifie le chemin vers le dockerfile qui se trouve dans le répertoire courant représenté par le "dot" . Le résultat de cette commande et l'exécution des commandes qui se trouve dans le dockerfile et la création de l'image.</p>

<p>Lister les images créées</p>	 <pre> root@ubuntu:/# docker images REPOSITORY TAG IMAGE ID CREATED SIZE dockernized-bash latest dd5b43793b12 About a minute ago 214MB ubuntu 18.04 f5cbcd4244ba 5 days ago 63.2MB ubuntu 20.04 ff0fea8310f3 3 weeks ago 72.8MB ubuntu latest ff0fea8310f3 3 weeks ago 72.8MB hello-world latest feb5d9fea6a5 6 months ago 13.3kB root@ubuntu:/# </pre> <p>Explication : Pour vérifier que l'image a été créée, on utilise la commande : <code>docker images</code> qui liste les images sur le système. Lorsque l'image est bien créée elle apparaît dans la liste des images comme montré dans la capture.</p>
<p>Démarrer le conteneur</p>	 <pre> root@ubuntu:/home/meriem/bash# docker run -it dockernized-bash root@d2b4647058b6:/# ls bin dev home lib64 mnt proc run srv tmp var boot etc lib media opt root sbin sys usr root@d2b4647058b6:/# apt-get update Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB] Get:2 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [2694 kB] Get:3 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB] Get:4 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB] Get:5 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB] Get:6 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages [21.1 kB] </pre> <p>Explication Pour démarrer le conteneur on utilise la commande : <code>docker run</code> On spécifie le nom d'image : <code>dockernized-bash</code> Avec les options : <code>-t</code> : pour allouer un pseudo-terminal. <code>-i</code> : pour démarrer le conteneur en mode interactif qui permet d'accéder au terminal du conteneur en cours d'exécution. Le résultat de cette commande qu'on on peut voir sur la capture ce qu'on ouvre un terminal sur le système de conteneur, et qu'on est sur le répertoire root <code>"/</code>, maintenant on peut exécuter 'n'importe quelle commande. on a exécuter <code>apt-get update</code> pour obtenir les informations sur les packages à installer ou mettre à jour. Et qui est recommandée pour pouvoir installer des des packages par la suite.</p>

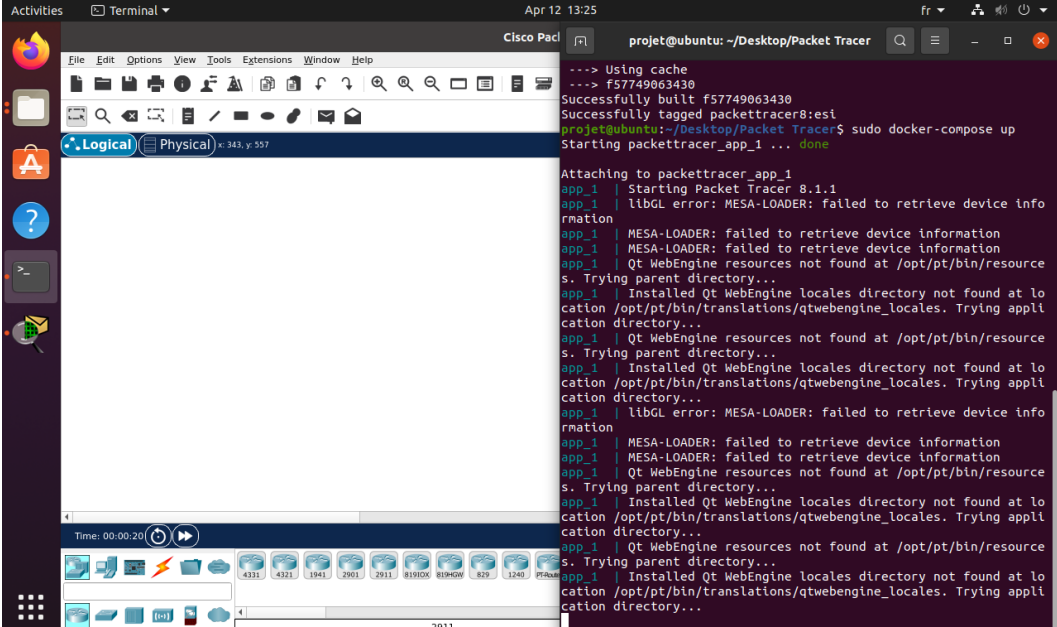
Tester la commande ping à l'intérieur du conteneur	<pre>root@d2b4647058b6:/# apt-get install iputils-ping Reading package lists... Done Building dependency tree Reading state information... Done iputils-ping is already the newest version (3:20161105-1ubuntu3). 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded. root@d2b4647058b6:/# ping google.com PING google.com (172.217.18.46) 56(84) bytes of data. 64 bytes from ham02s12-in-f46.1e100.net (172.217.18.46): icmp_seq=1 ttl=127 time=48.4 ms 64 bytes from ham02s12-in-f46.1e100.net (172.217.18.46): icmp_seq=2 ttl=127 time=46.8 ms 64 bytes from ham02s12-in-f46.1e100.net (172.217.18.46): icmp_seq=3 ttl=127 time=172 ms</pre> <p>Explication : Sur le conteneur on installer la commande : ping , qu'on l'utilise pour vérifier la connexion internet on pingant : <i>google.com</i></p>												
Executer un script à l'intérieur du conteneur	<pre>root@d2b4647058b6:/# echo echo script executed from docker container >script root@d2b4647058b6:/# ls bin boot dev etc home lib lib64 media mnt opt proc root run sbin script srv sys tmp usr var root@d2b4647058b6:/# chmod +x script root@d2b4647058b6:/# ./script script executed from docker container</pre> <p>Explication Sur le conteneur on crée un script appelé “<i>script</i>” qui exécute une commande echo.</p> <p>On peut configurer les permission d'accès, et rendre le fichier exécutable, avec la commande : chmod</p> <p>Le script s'exécute avec succès.</p>												
Afficher la liste des conteneurs créés	<pre>root@ubuntu:/# docker ps -a</pre> <table><thead><tr><th>CONTAINER ID</th><th>IMAGE</th><th>COMMAND</th><th>CREATED</th><th>STATUS</th><th>PORTS</th></tr></thead><tbody><tr><td>d2b4647058b6</td><td>dockernized-bash</td><td>"bash"</td><td>8 minutes ago</td><td>Up 8 minutes</td><td></td></tr></tbody></table> <p>Explication Pour lister les conteneurs créer on utilise la commande : docker ps avec l'option :</p> <p>-a : pour afficher tous les conteneurs (même qui ne sont pas en cours d'exécution)</p>	CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	d2b4647058b6	dockernized-bash	"bash"	8 minutes ago	Up 8 minutes	
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS								
d2b4647058b6	dockernized-bash	"bash"	8 minutes ago	Up 8 minutes									

TABLE 4.1: Tests de fonctionnement Linux Terminal

4.2 Packet Tracer

Test	Résultats obtenus
<p>création de l'image</p>	 <p>Explication : On commence par la commande : <code>docker-compose build</code> qui crée l'image en utilisant le fichier <code>docker-compose.yml</code>. Le résultat de cette commande est l'exécution des commandes du <code>dockerfile</code> qui se trouve dans le chemin indiqué par le champ build de <code>docker-compose</code>, et la création de l'image.</p>

Création du conteneur



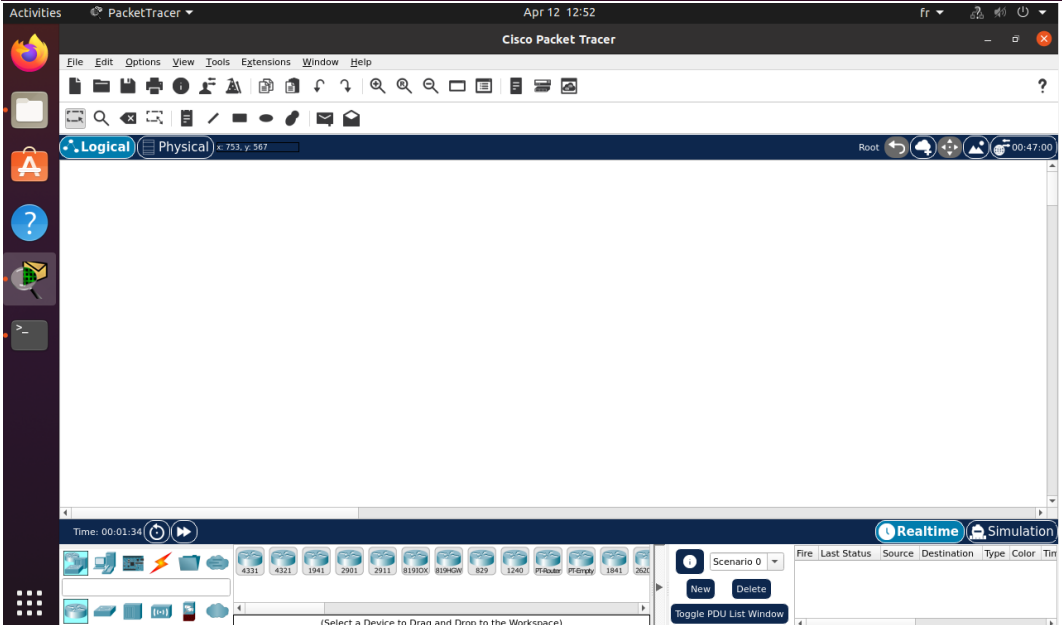
```
---> Using cache
---> f57749063430
Successfully built f57749063430
Successfully tagged packettracer8:est
proj@ubuntu:~/Desktop/Packet Tracer$ sudo docker-compose up
Starting packettracer_app_1 ... done

Attaching to packettracer_app_1
app_1 | Starting Packet Tracer 8.1.1
app_1 | libGL error: MESA-LOADER: failed to retrieve device info
rmation
app_1 | MESA-LOADER: failed to retrieve device information
app_1 | MESA-LOADER: failed to retrieve device information
app_1 | Qt WebEngine resources not found at /opt/pt/bin/resource
s. Trying parent directory...
app_1 | Installed Qt WebEngine locales directory not found at lo
cation /opt/pt/bin/translations/qtwebengine_locales. Trying appli
cation directory...
app_1 | Qt WebEngine resources not found at /opt/pt/bin/resource
s. Trying parent directory...
app_1 | Installed Qt WebEngine locales directory not found at lo
cation /opt/pt/bin/translations/qtwebengine_locales. Trying appli
cation directory...
app_1 | libGL error: MESA-LOADER: failed to retrieve device info
rmation
app_1 | MESA-LOADER: failed to retrieve device information
app_1 | MESA-LOADER: failed to retrieve device information
app_1 | Qt WebEngine resources not found at /opt/pt/bin/resource
s. Trying parent directory...
app_1 | Installed Qt WebEngine locales directory not found at lo
cation /opt/pt/bin/translations/qtwebengine_locales. Trying appli
cation directory...
app_1 | Qt WebEngine resources not found at /opt/pt/bin/resource
s. Trying parent directory...
app_1 | Installed Qt WebEngine locales directory not found at lo
cation /opt/pt/bin/translations/qtwebengine_locales. Trying appli
cation directory...
```

Explication : On utilise la commande : `docker-compose up` pour créer le conteneur et démarrer les services définie dans le fichier **docker-compose.yml**, dans le mode attaché ou attaché mode (ajouter l'option `-d` pour le mode détache ou detached mode)

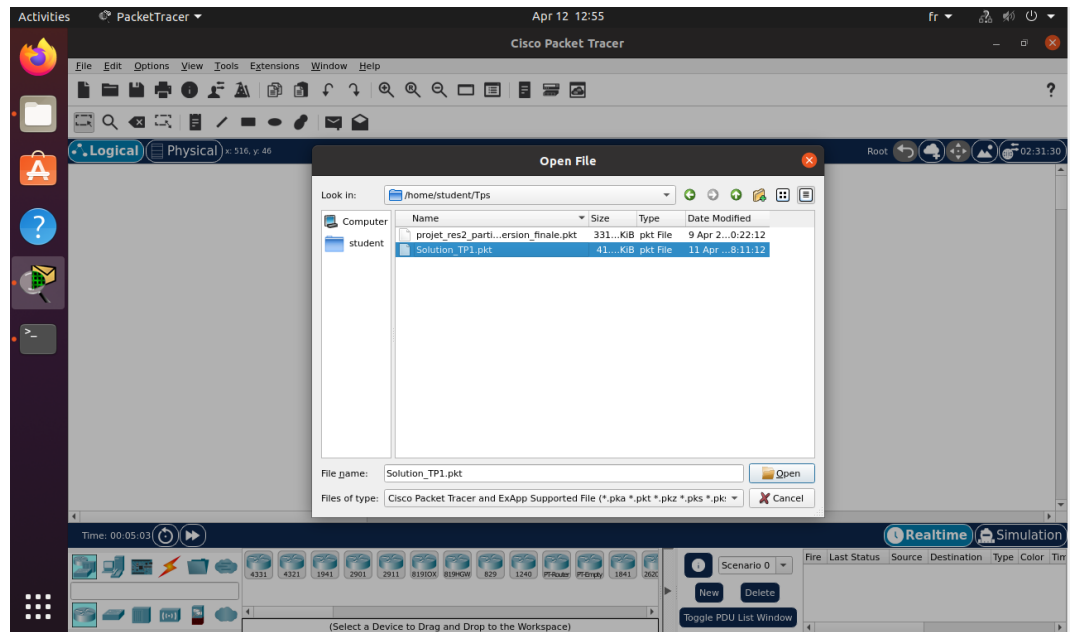
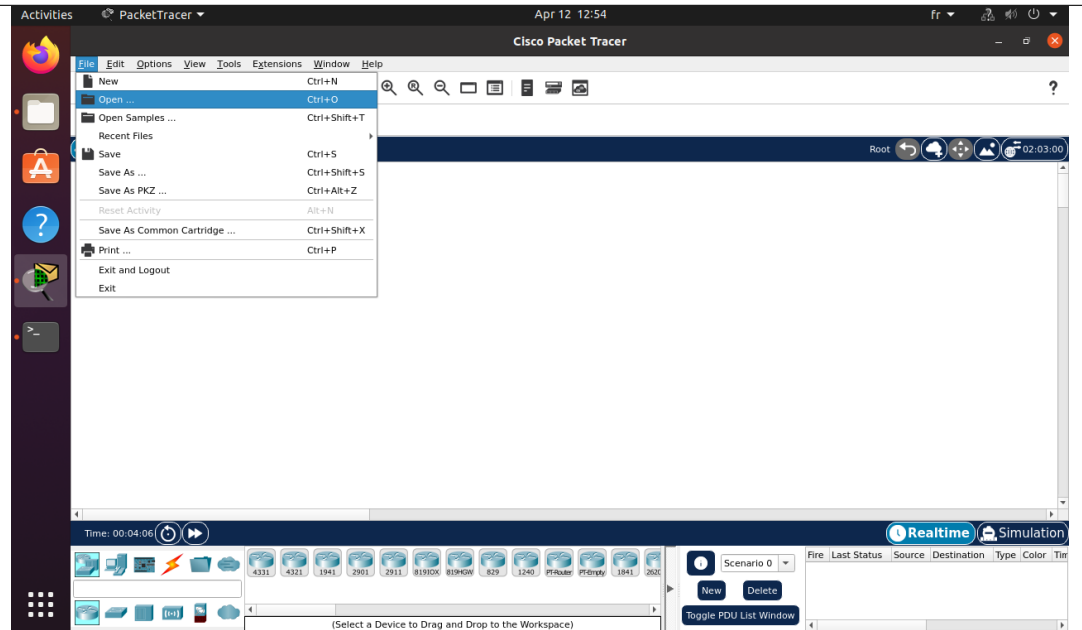
Démarrer
packet tracer

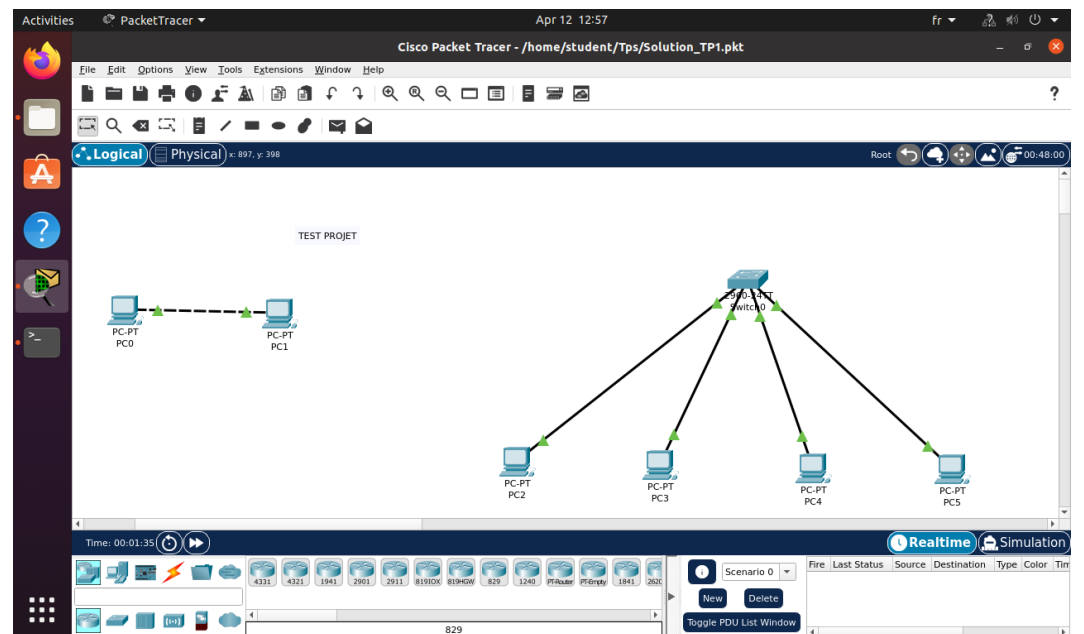
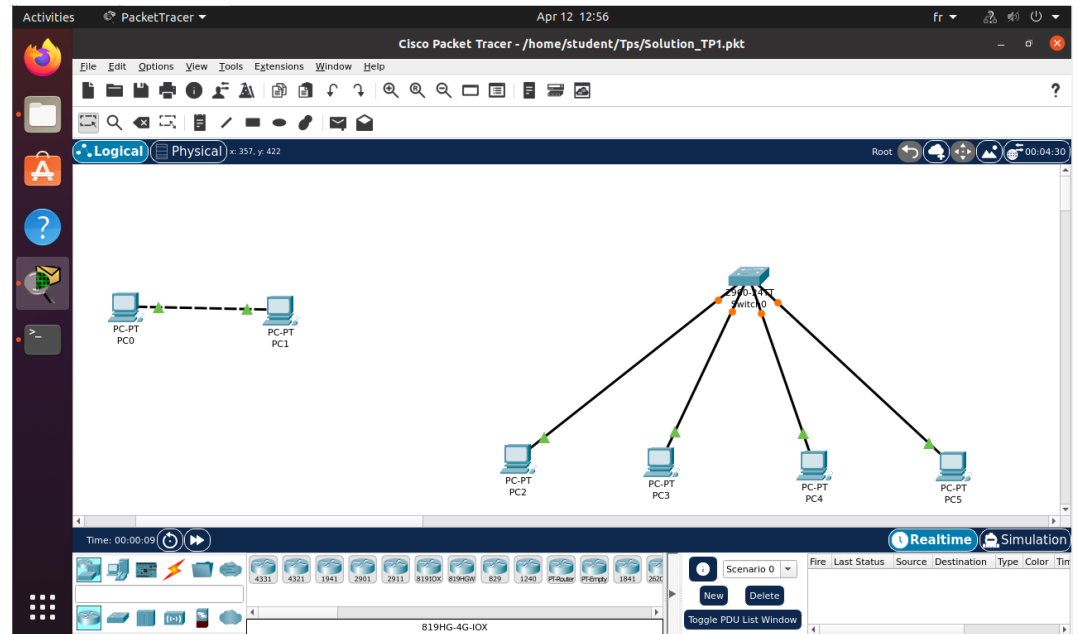
```
projet@ubuntu:~/Desktop$ sudo docker start packettracer_app_1
packettracer_app_1
projet@ubuntu:~/Desktop$
```



Explication : Pour démarrer le conteneur on peut utiliser la commande : `docker start`, en spécifiant le nom de conteneur : `packettracer_app_1`. Le résultat de cette commande est le démarrage de l'application Packet Tracer comme on peut voir dans la capture.

Ouvrir un TP
dans
l'application
Packet Tracer





Explication : Sur l'application conteneurisé Packet Tracer on ouvre le fichier de TP (fichier `.pkt`) qui se trouve dans le volume : `/home/student/Tps`. Le fichier de TP s'ouvre avec succès, et on peut voir que les liaisons entre les éléments de la topologie fonctionnent correctement.

TABLE 4.2: Tests de fonctionnement Packet Tracer

5. Conclusion

Les expérimentations sont divisées en deux parties, la première a concerné les commandes shell de linux qui représente une application légère et permet après de tester les limites d'une conteneurisation combinée avec orchestration à grande échelle. Ces tests nous ont permis d'avoir une estimation de certains paramètres comme la stabilité, sécurité et scalabilité potentielle. Nous avons aussi présenté les problèmes et défis liés à l'utilisation des images docker ce qui nous pousse par la suite à considérer une solution basée sur l'orchestration des conteneurs. Les expérimentations ont permis d'évaluer la performance des images selon certaines configurations et démontré qu'avec le bon choix de configuration, ces dernières peuvent au moins délivrer la même performance qu'un ordinateur de bureau.

La deuxième partie de ce travail a concerné les résultats obtenus. Nous avons motivé le choix de la technologie qui est Docker, puisqu'elle est la plus populaire et la plus rapide du marché en ce moment et qu'elle offre une facilité de déploiement remarquable, en se basant sur le travail élaboré dans le rapport précédent.

6. Annexe : Commandes Docker Utilisées et Description des Tests

6.1 Introduction :

Les tests sont une partie essentielle du développement de logiciels modernes. Les tests peuvent signifier beaucoup de choses pour différentes équipes de développement. Il existe des tests unitaires, des tests d'intégration et des tests de bout en bout. Dans notre cas, nous examinons comment exécuter des tests unitaires dans Docker.

Dans ce qui suit nous reprenons les commandes utilisées pour la mise en place des images tout en détaillant les scénarios et plans de test .

6.2 Commandes de mise en place du DockerFile :

Syntaxe générale d'une commande : " command argument argument 1..."

Exemples :

- Print "Get Certified. Get Ahead"
- Run echo "Get Certified. Get Ahead"

Avant de créer notre premier Dockerfile, il est important de comprendre ce qui compose le fichier. Dockerfile se compose de commandes spécifiques qui vous guident sur la façon de créer une image Docker spécifique.

Les commandes spécifiques que vous pouvez utiliser dans un dockerfile sont :

FROM, PULL, RUN et CMD.

Commande	Effet
FROM os:version	Créer une couche à partir de l'OS hôte
PULL	Ajouter des fichiers à partir du repo Docker
RUN	Build le conteneur
CMD	Spécifier quelles commandes exécuter dans le conteneur
ENTRYPOINT application "arg, arg1"	Permet de spécifier une commande avec ses paramètres

<code>ADD /[source]/[destination]</code>	Permet de copier des données vers l'image Docker
<code>ENV key value</code>	Fournit les valeurs par défaut pour les variables dans le conteneur
<code>MAINTAINER [name]</code>	Déclarer l'auteur des images docker
<code>docker build [OPTIONS] PATH URL -</code>	Déclarer le chemin où stocker le DockerFile
<code>docker build [-t] tag [location of your dockerfile]</code>	Construire une image basique à partir d'un DockerFile, l'option -t permet de tagguer une image avec un nom
<code>docker images</code>	Vérifier la création des images docker
<code>docker run --name nom_conteneur nom_image</code>	Permet de créer un conteneur Docker à partir d'une image Docker

TABLE 6.1: Listes des commandes spécifiques utilisées dans un dockerfile