Correction

Nantes Université

M1INFO – Développement, données et exploitation (XMS1IE020)
Contrôle Continu Écrit (CCE)

22/12/2023

– Documents non autorisés. –

- *Pour les cases,* ***utilisez un STYLO NOIR, pas de crayon de papier, ni de couleur.*** *Noircissez complètement (n'entourez pas et ne cochez pas) les cases. Si besoin, l'enseignant a un effaceur.*
- *Le barème est donné à titre indicatif, il sera susceptible de changer.*
- *La taille d'une boîte de réponse n'est pas forcément représentative de la longueur attendue de la réponse.*

Correction

Nom et prénom :

.....

N° étudiant : .....

Encodez ici votre N° étudiant **en noircissant exactement une case par colonne** (chaque colonne correspond à un caractère de votre N° étudiant) :

					A
					B
		0			C
		1			D
		2			E
		3			F
		4			G
		5			H
		6			J
		7			K
		8			L
		9			M
		A			N
	0	B	0	0	0	P
	1	C	1	1	1	Q
	2	D	2	2	2	R
	3	E	3	3	3	S
	4	F	4	4	4	T
0	5	G	5	5	5	U
1	6	H	6	6	6	W
2	7	J	7	7	7	X
8	8	K	8	8	8	Y
9	9	L	9	9	9	Z

Questions de cours

Question 1 *(1.5 points)* Quelle est la différence entre une image de conteneur et un conteneur ? Vous pouvez illustrer avec un exemple simple.

Corrigé : Une image de conteneur contient un logiciel dans un micro-environnement qui lui est adapté. Un conteneur est créé à partir d'une image de conteneur. On peut créer plusieurs conteneurs à partir d'une même image.

Par exemple : une image de conteneur Nginx permet de créer des conteneurs qui exécutent Nginx.

Question 2 *(1 point)* Soit un conteneur créé avec la commande

```
1 podman container run --rm localhost/monimage
```

Que devient le conteneur si son execution se termine abruptement (par exemple à cause d'un plantage) ?

Corrigé : Il est automatiquement supprimé (du fait de –rm).

Question 3 *(1.5 points)* Dans un contexte de conteneurisation, expliquez la différence entre une *publication de port* et un *réseau virtuel*. Vous pouvez illustrer avec des exemples simples.

Corrigé : Une publication de port permet de relier un conteneur avec son environnement hôte. Plus précisément, cela permet de rediriger vers un port d'un conteneur les connexions faites sur le port de l'environnement hôte. Par exemple, on peut rediriger le port 7777 d'un environnement hôte, vers le port 80 d'un conteneur Nginx.

Un réseau virtuel permet de relier des conteneurs entre eux, qui peuvent ensuite se parler par leurs ports réseaux. Par exemple, on peut mettre en réseau un conteneur avec une application, et un conteneur avec une base de données.

Correction

Question 4 *(1 point)* On se connecte en SSH sur un serveur dont l'adresse réseau est `monserveur`, puis on lance un conteneur web de la manière suivante :

```
1 podman container run -p 7777:80 docker.io/nginx
```

On tente ensuite de tester le déploiement en accédant à `monserveur:80`, mais ça ne fonctionne pas, avec un message "connexion impossible". Pourquoi ?

Corrigé : La publication de port fait qu'il faut se connecter à `monserveur:7777`, et non `monserveur:80`.

Problème : déploiement de *lldap*

Énoncé

Fonctionnement de *lldap*

On développe *lldap*, un logiciel libre qui fournit un service LDAP (*Lightweight Directory Access Protocol*) qui permet de gérer un annuaire de comptes utilisateurs centralisé. Chaque compte utilisateur possède des informations comme un nom, un identifiant, un mot de passe, un groupe, etc. Un serveur LDAP comme *lldap* peut être utilisé d'autres logiciels pour l'authentification de leurs utilisateurs. Ainsi, un utilisateur possédant un compte dans un annuaire LDAP donné peut s'authentifier dans tous les logiciels configurés pour utiliser cet annuaire.

- *lldap* est un logiciel développé principalement avec le langage de programmation Rust, qui est un langage qui nécessite une compilation avant de pouvoir être exécuté.
- À l'aide du port réseau 3890, *lldap* fournit un service de type LDAP. Ce service est utilisé par les logiciels qui souhaitent utiliser l'annuaire LDAP pour authentifier leurs utilisateurs.
- Les données de *lldap* (à savoir, les comptes utilisateurs) sont stockées au sein d'une base de données de type PostgreSQL. On spécifie à *lldap* les coordonnées de la base de données sous la forme d'un argument passé à la commande (voir ci-après).
- On exécute *lldap* avec la commande suivante :

  ```
  $ lldap run <options>
  ```

  où `<options>` contient les options de lancement pour *lldap*. On peut notamment spécifier les options suivantes :

  `--database-url` est une option obligatoire qui spécifie la connexion à la base de données à l'aide du format suivant : `postgres://<utilisateur>:<mot_de_passe>@<adresse_réseau>/<nom_de_la_base>`

  `--server-key-seed` est une option obligatoire qui spécifie la *graine aléatoire* que doit utiliser le logiciel pour générer ses clés cryptographiques. Cette valeur doit être une chaîne de caractères créée à l'aide d'un générateur d'aléatoire, et doit pouvoir la paramétrer pour chaque déploiement réalisé.

Développement de *lldap*

Le développement de *lldap* se fait dans un répertoire qui contient tout le code source du logiciel. La gestion des dépendances et de la compilation du code Rust est réalisée à l'aide du logiciel *Cargo*. Ainsi, pour télécharger toutes les dépendances et compiler une version déployable de *lldap*, on se place dans le répertoire de développement, et on exécute l'unique commande suivante :

```
$ cargo build --release -p lldap
```

Une fois cette commande réussie, un exécutable est disponible à l'emplacement `./target/release/lldap` du répertoire de développement. Ainsi, si on se place dans le répertoire de développement, cet exécutable peut être directement exécuté à l'aide de la commande :

```
$ ./target/release/lldap run <options>
```

Plan de déploiement

On souhaite faire un déploiement de notre logiciel *lldap* à l'aide de conteneurs exécutés via Podman, sur un serveur de déploiement similaire au serveur de déploiement utilisé durant les travaux pratiques. Pour cela, on prévoit de créer *deux conteneurs* sur le serveur de déploiement :

- Un **conteneur PostgreSQL** pour la base de données.
- Un **conteneur *lldap***, à l'aide d'une image confectionnée par nos soins.

On souhaite en outre que notre image de conteneur *lldap* puisse être utilisée dans différents déploiements (par exemple, pour des déploiements avec des bases de donnée différentes, ou bien avec des graines aléatoires différentes).

Pour accomplir ces tâches, on s'intéresse tout particulièrement à deux images de conteneur existantes : l'image de conteneur officielle PostgreSQL, et l'image de conteneur officielle Rust. Ces deux images sont décrites ci-après.

Image de conteneur PostgreSQL

- L'image est disponible sur le registre du Docker Hub (qui a pour adresse `docker.io`) sous le nom `postgres`.
- Une étiquette valide pour l'image est l'étiquette `16`.
- Le point d'entrée de l'image est l'exécution du logiciel PostgreSQL.
- Le logiciel écoute par défaut sur le port 5432 du conteneur.
- Le logiciel stocke ses données dans le répertoire `/var/lib/postgresql/data` du conteneur.
- Lors du premier lancement d'un conteneur, un compte PostgreSQL est créé, ainsi qu'une base de données vide à laquelle ce compte peut accéder. Des variables d'environnement sont disponibles pour configurer cela :
  - `POSTGRES_USER` permet de spécifier le nom du compte PostgreSQL,
  - `POSTGRES_PASSWORD` permet de spécifier le mot de passe du compte `POSTGRES_USER`,
  - `POSTGRES_DB` permet de spécifier le nom de la base de données créée pour le compte `POSTGRES_USER`,

Image de conteneur Rust

- L'image est disponible sur le registre du Docker Hub (qui a pour adresse `docker.io`) sous le nom `rust`.
- Une étiquette valide pour l'image est l'étiquette `1.74.1-slim`.
- L'image a été construite avec comme base une image de conteneur Debian.
- L'image possède l'outil de compilation *Cargo*, exécutable avec la commande `cargo`.

Aide technique

Rappel de la syntaxe de `podman network create` :

```
$ podman network create <nom du réseau>
```

Rappel de la syntaxe de la commande `podman container run` :

```
$ podman container run
    --name <nom du conteneur> \
    <autres options> \
    <référence d'image>
```

où `<autres options>` peut notamment inclure :

- `--env <nom variable>=<valeur variable>` (version courte : `-e`)
- `--rm`
- `--detach` (version courte : `-d`)
- `--volume <chemin environnement hôte>:<chemin à l'intérieur du conteneur>` (version courte : `-v`)
- `--publish <port environnement hôte>:<port conteneur>` (version courte : `-p`)
- `--network <nom du réseau>`

Rappel de la syntaxe de la commande `podman image build` :

```
$ podman image build
     --file <chemin Containerfile> \
     --tag <nom de l'image>:<étiquette> \
     <chemin contexte>
```

Questions

Question 5 *(1.5 points)* Listez les liens au monde extérieur requis par le conteneur PostgreSQL à déployer. Pour chaque lien, nommez le mécanisme utilisé. *Il n'est pas demandé ici de préciser les commandes à exécuter.*

Corrigé : Le conteneur doit posséder un montage de répertoire pour persister les fichiers de la base de données, et doit être relié à un réseau virtuel sur lequel se trouvera le conteneur lldap.

Question 6 *(1 point)* Est-ce que des actions sont nécessaires avant la création et l'exécution de nos deux conteneurs ? Si vous estimez que oui, préciser la ou les commandes à exécuter.

Corrigé : podman network create lldap_network

(et éventuellement création de répertoires avant montage)

Question 7
Donnez la commande Podman permettant de créer et exécuter un conteneur PostgreSQL pour le déploiement demandé. Vous prendrez en compte votre réponse à la question précédente, ainsi que les contraintes additionnelles suivantes :

- Le nom du conteneur PostgreSQL doit être `lldap_db`.
- Le conteneur doit être en mode détaché, et doit se supprimer automatiquement si arrêté.
- Vous pouvez choisir librement les valeurs de configuration.

Corrigé :

```
podman container run --name lldap_db  \
                        --rm  \
                        --detach  \
                        --volume /chemin/vers/data1:/var/lib/postgresql/data \
                        --network lldap_network  \
                        --env POSTGRES_USER=lldap_user \
                        --env POSTGRES_PASSWORD=lldap_password \
                        --env POSTGRES_DB=lldap_database \
                        docker.io/postgres:16
```

à prendre en compte : nom, rm, detach, montage, network, envs, image, étiquette

Correction

Question 8 *(1.5 points)* Quel(s) paramètre(s) faudra-il donner à *lldap* quand il sera exécuté dans un conteneur ? Expliquez comment permettre la définition de ce(s) paramètre(s) lors de la création d'un conteneur *lldap*.

Corrigé : Il faudra lui donner la graine d'aléatoire. On peut définir ce paramètre sous la forme d'une variable d'environnement utilisée dans le point d'entrée (CMD), et transmise à la création du conteneur avec –env.

Question 9
Proposez le contenu d'un *Containerfile* qui permette de construire une image de conteneur pour *lldap* avec les contraintes suivantes :

- La compilation de *lldap* doit se faire durant la construction de l'image.
- L'exécution d'un conteneur à partir de l'image doit entraîner le démarrage de *lldap*.

BONUS : *0.5 point supplémentaire si la solution proposée est de type multi-stage, en utilisant* `docker.io/debian:12` *comme environnement d'exécution.*

Corrigé :
(version sans multi-stage)

```
1  FROM docker.io/rust:1.74.1-slim
2
3  WORKDIR /lldap_src
4
5  COPY . .
6
7  RUN cargo build --release -p lldap
8
9  CMD ./target/release/lldap run \
10     --database-url postgres://lldap_user:lldap_password@lldap_db/lldap_database \
11     --server-key-seed $CUSTOM_SEED
```

Question 10
Donnez la commande Podman permettant de construire une image à partir de votre *Containerfile*, avec les contraintes suivantes :

- On suppose que la commande est exécutée depuis le répertoire de développement.
- L'image de conteneur doit être nommée `lldap` et avoir pour étiquette `v1`.
- On ne souhaite pas publier l'image sur un registre de conteneur distant.

Corrigé : podman image build –tag lldap .

Correction

Question 11
Donnez la commande Podman permettant de créer et exécuter un conteneur à partir de votre image, avec les contraintes suivantes :

- On suppose que votre image a déjà été construite selon les consignes de la question précédente.
- Le conteneur doit être nommé `lldap_instance`.
- On doit pouvoir accéder au service en se connectant au port 4564 du serveur de déploiement.
- On suppose qu'une graine aléatoire a déjà été générée, et possède comme valeur `"123456"`.

Corrigé :

```
podman container run --name lldap_instance \
                     --rm \
                     --detach \
                     --env CUSTOM_SEED=123456 \
                     --publish 4564:3890 \
                     localhost/lldap
```

Correction