TPS Développement et Exploitation - DevOps DataOps

TP1 Découverte de l'infrastructure et des outils

Anthony MUDET - M1 SMART Computing

2. Usages principaux du serveur de déploiement

2.1 Connection SSH par mot de passe

Création de la clé ssh

- **sur machine** : `$ ssh-keygen -t ecdsa`

Connection à podman

- **sur machine** : `$ ssh E253432U@podman`
- **sur machine** : `$ ssh-copy-id podman`

Affichage de l'emplacement du répertoire

- **sur podman** : `$ pwd`
- **réponse** : `/comptes/E253432U/`

Affichagedu contenu

- **sur podman** : `$ ls` (aucune réponse)

Création du répertoire

- **sur podman** : `$ mkdir tp-decouverte-serveur` puis `$ cd tp-decouverte-serveur`

Création d'un fichier txt

- **sur podman** : `$ nano hello.txt`
- **dans hello.txt** : j'écris Hello world! puis `CRTL+O` pour écrire et `CRTL+X` pour quitter
- *Note* : pour que ce soit plus simple j'utiliserai `$ echo` pour écrire dans un nouveau fichier

Affichage de hello.txt

- **sur podman** : `$ cat hello.txt`
- **réponse** : `Hello world!`

2.2 Générer sa clé SSH personelle

Création de la clé SSH

- **sur podman** : `$ ssh-keygen -t ecdsa`

Vérification de la création

- **sur podman** : `$ ls ~/.ssh`
- **réponse** : `authorised_keys id_ecdsa id_ecdsa.pub`

2.4 Envoi de fichier vers le serveur

Création du dossier

- **sur machine** : `$ mkdir tp-decouverte-envoi` puis `$ cd tp-decouverte-envoi`

Mise en place des fichiers et dossiers

- **sur machine** :
  - `$ echo "Ceci est mon premier envoi" >> envoi1.txt`
  - `$ mkdir envoi2` puis `$ cd envoi2`
  - `$ echo "Ceci est le premier fichier de mon second envoi" >> envoi2_a.txt`
  - `$ echo "Ceci est le second fichier de mon second envoi" >> envoi2_b.txt`

Vérification des fichiers

- **sur podman** : `$ ls` (actuellement dans le répertoire tp-decouverte-serveur)
- **réponse** : `hello.txt`
- **sur machine** : `$ ls` (actuellement dans le répertoire tp-decouverte-envoi)
- **réponse** : `envoi1.txt envoi2`

Envoi de envoi1.txt

- **sur machine** : `$ scp envoi1.txt podman:~/tp-decouverte-serveur`
- **sur podman** :
  - `$ ls` **réponse** : `envoi1.txt hello.txt`
  - `$ cat envoi1.txt` **réponse** : `Ceci est mon premier envoi`

Envoi du dossier envoi2

- **sur machine** : `$ scp -r envoi2 podman:~/tp-decouverte-serveur`
- **sur podman** :
  - `$ ls` **réponse** : `envoi1.txt envoi2 hello.txt`
  - `$ cat envoi2/envoi2_a.txt` **réponse** : Ceci est le premier fichier de mon second envoi
  - `$ cat envoi2/envoi2_b.txt` **réponse** : Ceci est le second fichier de mon second envoi

3. Comprendre un peu l'environnement de déploiement

3.1. Carte d'identité de l'environnement

Connaître le système d'exploitation

- **sur podman** : `$ cat /etc/os-release`

- **réponse** :

```
PRETTY_NAME="Debian GNU/Linux 13 (trixie)"
NAME="Debian GNU/Linux"
VERSION_ID="13"
VERSION="13 (trixie)"
VERSION_CODENAME=trixie
DEBIAN_VERSION_FULL=13.1
ID=debian
HOME_URL="https://www.debian.org/"
SUPPORT_URL="https://www.debian.org/support"
BUG_REPORT_URL="https://bugs.debian.org/"
```

- ici on voit que podman tourne sur la version 13 (appelé trixie) de Debian
- **sur podman** : `$ hostnamectl`
- **réponse**

```
Static hostname: podman
    Icon name: computer-vm
        Chassis: vm 
    Machine ID: d0711669da3c4e54ad8b28726304b98a
        Boot ID: d45f1017c9ea4e8e96775894ef13289a
    AF_VSOCK CID: 1
Virtualization: kvm
Operating System: Debian GNU/Linux 13 (trixie)
        Kernel: Linux 6.12.43+deb13-amd64
    Architecture: x86-64
Hardware Vendor: QEMU
Hardware Model: Standard PC _i440FX + PIIX, 1996_
Firmware Version: rel-1.16.3-0-ga6ed6b701f0a-prebuilt.qemu.org
Firmware Date: Tue 2014-04-01
    Firmware Age: 11y 6month 1w 1d
```

- Avec cette commande on voit également que le système tourne sur la version 13 de Debian.

3.2. Variables d'environnement

- **sur podman** :
  - `$ printenv` (je n'affiche pas le résultat vu qu'il y a beaucoup de résultats)
  - `$ MAVARIABLE="mavaleur"`
  - `$ echo $MAVARIABLE` **réponse** : `mavaleur`

3.3 Droits d'accès

1. test des droits dans le répertoire courant

- **sur podman** :
  - `$ touch ~/fichier_temp`

- `$ ls ~` **réponse** : `fichier_temp tp-decouverte-serveur`
- `$ rm ~/fichier_temp` -> OK

2. test des droits dans un répertoire avec accès limité

- **sur podman** : `$ touch /etc/fichier_temp`
- **réponse** : `touch: impossible de faire un touch '/etc/fichier_temp': Permission non accordée`

3. test d'installation de git

- **sur podman** : `$ apt install git`
- **réponse** :

```
Erreur : Impossible d'ouvrir le fichier verrou /var/lib/dpkg/lock-frontend - open (13: Permission non accordée)
Erreur : Impossible d'obtenir le verrou de dpkg (/var/lib/dpkg/lock-frontend). Avez-vous les droits du superutilisateur ?
```

4. Quelques petits déploiements (sans conteneurs)

4.1. replace_string

Téléchargement et envoi

- **sur machine** :
  - `$ git clone https://gitlab.univ-nantes.fr/gl/developpement_exploitation/resources/small_programs.git`
  - `$ scp -r small_programs podman:~/`
- **sur podman** :
  - `$ cd ~/small_programs/1_replace_string`
  - `$ nano song.txt`
  - `$ python3 replace_string.py -s wall -r uwu song.txt`
- **réponse** :

```
wall -r uwu song.txt
Ten green uwus
Hanging on the uwu
Ten green uwus
Hanging on the uwu
And if one green uwu
Should accidentally fall
There’ll be nine green uwus
Hanging on the uwu
```

1. Il y a besoin de python pour qu'il fonctionne avec les librairies importées dans le code.
2. Non je n'ai pas vérifié

3. On peut utiliser la commande `$ python3 --version` ce qui retourne sur podman : `Python 3.13.5`

4.2. compress_file

- **sur podman** :
  - `$ cd ../2_compress_file`
  - `$ python3 compress_file.py -o compressed README.adoc`
- **réponse** :

```
Traceback (most recent call last):
File
"/comptes/E253432U/small_programs/2_compress_file/compress_file.py",
line 1, in <module>
    from compression import zstd
ModuleNotFoundError: No module named 'compression'
```

1. À l'exécution, le programme crash car il ne trouve pas le module 'compression'
2. En utilisant la commande `$ pip show compression` on a comme réponse `WARNING: Package(s) not found: compression` ce qui signifie que la dépendance n'est pas installée.
3. Il nous faudrait passer python3.14 pour utiliser le programme et nous n'avons pas accès aux permissions nécessaire pour mettre à jour.

4.3. print_colored_string

- **sur podman** :
  - `$ cd ../3_print_colored_string`
  - `$ python3 print_colored_string.py -b green -c red "It's a trap!"`
- **réponse** :

```
Traceback (most recent call last):
File
"/comptes/E253432U/small_programs/3_print_colored_string/print_colored
_string.py", line 1, in <module>
    from colored import Fore, Back, Style
ModuleNotFoundError: No module named 'colored'
```

1. À l'exécution, le programme crash car il ne trouve pas le module 'colored'
2. En utilisant la commande `$ pip show colored` on a comme réponse `WARNING: Package(s) not found: colored` ce qui signifie que la dépendance n'est pas installée.
3. Ici la bibliothèque existe bien pour notre version de l'interpréteur Python, mais il n'est pas installé et comme nous ne somme pas dans un environnement virtuel, il faudrait installer la dépendance via `$ apt install` sauf que l'on a vu plus tôt que l'on avait pas les droits nécessaires.

4.4. TicTacToe

- **sur podman**
  - `$ cd ../4_tictactoe`
  - `$ javac TicTacToe.java`
- **réponse** :

```
-bash: javac : commande introuvable
```

1. Le programme ne peut pas se compiler car la commande `$ javac` n'est pas présente
2. Non java n'est pas présent sur le système
3. Non on ne peut pas l'installer car on a pas la permission d'installer des choses sur le système.

TP2 Créer et manipuler des conteneurs

3. Tutoriel : un premier conteneur Ubuntu

3.1. Exécution du conteneur

- **sur podman** : `$ podman container run --name mon_premier_conteneur --tty --interactive docker.io/ubuntu:24.04`
- **sur le container** :
  - `$ echo "Bonjour depuis un conteneur !"`
  - `$ cat etc/os-release`
  - **réponse** :

```
PRETTY_NAME="Ubuntu 24.04.3 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.3 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-
and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
```

- **sur podman** :
  - `$ podman container list`
  - **réponse** :

```
  CONTAINER ID  IMAGE                                COMMAND
CREATED        STATUS        PORTS       NAMES
  57dcefd0cd59  docker.io/library/ubuntu:24.04  /bin/bash
5 minutes ago  Up 5 minutes
mon_premier_conteneur
```

3.2. Installation d'un paquet comme administrateur du conteneur

- **sur le container** :
  - `$ apt update && apt install cowsay`
  - `$ /usr/games/cowsay "Bonjour depuis un conteneur !"`
  - **réponse** :

```
 _______________________________
< Bonjour depuis un conteneur ! >
 -------------------------------
        \   ^__^
        \   (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

3.3. Copier un fichier dans le conteneur

- **sur podman** : `$ nano r2d2.txt` -> puis copie-colle le contenu dans le fichier
- **sur podman** : `$ mkdir ~/tp-conteneurs && mv r2d2.txt ~/tp-conteneurs`
- **sur podman** : `$ podman container cp ~/tp-conteneurs/r2d2.cow mon_premier_conteneur:/root/`
- **sur le container** : `$ cd root && ls` ; **réponse** : `r2d2.txt`
- **sur le container** : `$ /usr/games/cowsay -f /root/r2d2.cow "Beep. boop."`
  - **réponse** :

```
 _____________
< Beep. boop. >
 -------------
  \
   \       ___
    \     /() \
     \    _|_____|_
          | | === | |
          |_|  O  |_|
          ||  O  ||
          ||__*__||
          |~ \___/ ~|
          /=\ /=\ /=\
```

```
          [_] [_] [_]
```

3.4. Stopper, lister et supprimer le conteneur

- **sur le container** : `$ exit`
- **sur podman** :
  - `$ podman container list --all`
    - **réponse** :

```
CONTAINER ID  IMAGE                                COMMAND
CREATED         STATUS                     PORTS
NAMES
57dcefd0cd59  docker.io/library/ubuntu:24.04  /bin/bash
26 minutes ago  Exited (0) 12 seconds ago
mon_premier_conteneur
```

  - `$ podman container rm mon_premier_conteneur`
  - `$ podman container list --all`
    - **réponse** :

```
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS
PORTS       NAMES
```

4. Exercice : Asciidoctor depuis un conteneur Alpine

4.2. Questions

1. Préparez et exécutez une commande pour créer le conteneur Alpine souhaité (cf l'énoncé plus haut), avec les options nécessaires pour lui donner le nom alpine_asciidoctor et permettre les interactions clavier avec le conteneur. On utilisera l'image de conteneur accessible par l'étiquette 3.22.

- **sur podman** : `$ podman container run --name alpine_asciidoctor --tty --interactive docker.io/alpine:3.22`

2. Une fois dans le conteneur, exécutez les commandes nécessaires pour installer asciidoctor. Vérifiez que l'outil est bien installé en l'exécutant avec l'option --version.

- **sur le container** : `$ apk add asciidoctor && asciidoctor --version`
  - **réponse** :

```
Asciidoctor 2.0.23 [https://asciidoctor.org]
Runtime Environment (ruby 3.4.4 (2025-05-14 revision
a38531fd3f) +PRISM [x86_64-linux-musl]) (lc:UTF-8 fs:UTF-8
```

```
in:UTF-8 ex:UTF-8)
```

3. Sur votre poste de travail, préparez un fichier financiers.adoc contenant la recette de financiers montrée en exemple plus haut.

- **sur podman** : `$ nano financiers.adoc` -> puis copie colle le contenu

4. À l'aide d'un second terminal, copiez financiers.adoc au sein de votre conteneur.

- **sur podman** : `$ podman container cp financiers.adoc alpine_asciidoctor:/`

5. Dans votre conteneur, exécutez asciidoctor de manière à convertir le fichier financiers.adoc en HTML. Utilisez ls pour vérifier que vous avez bien produit un fichier financiers.html

- **sur le container** : `$ asciidoctor financiers.adoc && ls` ; **résultat** : `financiers.adoc financiers.html`

6. À l'aide d'un second terminal, copiez le fichier financiers.html depuis conteneur, vers l'environnement de déploiement.

- **sur podman** : `$ podman container cp alpine_asciidoctor:/financiers.html ~/tp-conteneurs/`

7. Vérifiez avec ls que vous avez bien récupéré le fichier qui était dans le conteneur.

- **sur podman** : `$ ls ~/tp-conteneurs` ; **réponse** : `financiers.html r2d2.cow`

8. Quittez le conteneur, puis supprimez le.

- **sur le container** : `$ exit`
- **sur podman** : `$ podman container rm alpine_asciidoctor`

9. Selon vous, qu'est devenu le fichier financiers.adoc que vous aviez copié au sein du conteneur ?

    - L'image du conteneur est supprimé ainsi que tout ce que l'on a mit dedans, le principe de l'isolement.

5. Tutoriel : utiliser un conteneur pour déployer replace_string

5.1. Stratégie 1 : depuis une image de base Ubuntu

- **sur podman** : `$ podman container run -ti --name replace_string_container docker.io/ubuntu:24.04`
- **sur le container** : `$ apt update && apt install -y python3`
- **sur podman** : `$ podman container cp small_programs/1_replace_string replace_string_container:/`
- **sur le container** : `$ python3 replace_string.py -s uwus -r birds song.txt` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `$ podman container rm replace_string_container`

5.2. Stratégie 2 : depuis une image de base Python

- **sur podman** :
  - `$ podman container run -ti --name replace_string_container_v2 docker.io/python:3-slim bash`
  - `$ podman container cp small_programs/1_replace_string replace_string_container:/`
- **sur le container** : `$ python3 1_replace_string/replace_string.py -s birds -r uwus 1_replace_string/song.txt` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `$ podman container rm replace_string_container`

6. Exercice : utiliser des conteneurs pour déployer différents programmes

6.1. compress_file

- **sur podman** : `$ podman container run -ti --name compress_file docker.io/python:latest bash`
- **sur le container** : `$ python --version` ; **réponse** : `Python 3.14.0`
- **sur podman** :
  - `$ podman container cp small_programs/2_rcompress_file compress_file:/`
  - `$ podman container cp tp-decouverte-server/hello.txt compress_file:/`
- **sur le container** : `$ python3 2_compress_file/compress_file.py -o TEST_COMPRESSION hello.txt` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `podman container rm compress_file`

6.2. print_colored_string

- **sur podman** :
  - `$ podman container run -ti --name print_colored_string docker.io/python:latest bash`
  - `$ podman container cp small_programs/3_print_colored_string print_colored_string:/`
- **sur le container** :
  - `$ python3 -m venv .venv`
  - `$ pip install colored`
  - `$ python3 3_print_colored_string/print_colored_string.py -b green -c red "It's a trap!"` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `podman container rm print_colored_string`

6.4. TicTacToe

- **sur podman** :
  - `$ podman container run -ti --name tictacttoe docker.io/eclipse-temurin bash`
  - `$ podman container cp small_programs/4_tictactoe tictactoe:/`
- **sur le container** :
  - `$ cd 4_tictactoe/`
  - `$ javac TictTacToe.java`

    - `$ java TicTacToe` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `$ podman container rm tictactoe`

6.5 economic_dispatch

- **sur podman** :
    - `$ podman container run -ti --name dispatch docker.io/julia:1.11-trixie bash`
    - `$ podman container cp small_programs/5_economic_dispatch/ dispatch:/`
    - `$ nano input_file.jl` -> puis copie-colle le contenu de l'exemple dans le fichier
    - `$ podman container cp input_file.jl dispatch:/`
- **sur le container** :
    - `$ julia -e 'import Pkg; Pkg.add("JuMP")'`
    - `$ julia -e 'import Pkg; Pkg.add("HiGHS")'`
    - `julia 5_economic_dispatch/economic_dispatch.julia input_file.jl` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `$ podman container rm dispatch`

TP3 Montage de répertoires/fichiers

3. Tutoriel : premiers montages dans un conteneur Ubuntu

3.1. Un premier montage de répertoire

- **sur podman** :
    - `$ mkdir ~/tp-montage/tutoriel_1`
    - `$ echo "Ce fichier a été créé en dehors du conteneur" > ~/tp-montage/tutoriel_1/mon_fichier`
    - `$ cd ~`
    - ```
      $ podman container run --name conteneur_avec_montage \
            --tty \
            --interactive \
            --mount type=bind,source="./tp-montage/tutoriel_1",target="/repertoire_montage" \
            docker.io/ubuntu:24.04
      ```
- **sur le container** :
    - `$ ls /`
        - **réponse** :
        - ```
          bin   dev  home  lib64  mnt  proc                root  sbin  sys  usr
          boot  etc  lib   media  opt  repertoire_montage  run   srv   tmp  var
          ```

- `$ cd /repertoire_montage`
- `$ ls` **réponse** `mon_fichier`
- `$ cat mon_fichier` **réponse** : `Ce fichier a été créé en dehors du conteneur`

3.2. Deux répertoires très très liés

- **sur podman** : `$ nano ./mon_fichier` -> ajout de `ceci est un test`
- **sur le container** :
  - `$ cat mon_fichier`
    - **réponse** :
    - ```
      Ce fichier a été créé en dehors du conteneur
      Ceci est un test
      ```
  - `$ echo "Ceci est un second fichier" > ./mon_second_fichier`
- **sur podman** :
  - `$ ls` **réponse** : `mon_fichier mon_second_fichier`
  - `$ cat mon_second_fichier` **réponse** : Ceci est un second fichier

3.3. Un montage de fichier

- **sur podman** :
  - `$ echo "Ce fichier va être monté dans un conteneur" > ~/tp-montage/tutoriel_1/mon_troisième_fichier`
  - ```
    $ podman container run --name conteneur_avec_montage_2 \
              --tty \
              --interactive \
              --mount type=bind,source="./tp-montage/tutoriel_1/mon_troisième_fichier",target="/root/fichier_dans_conteneur" \
              docker.io/ubuntu:24.04
    ```
- **sur le container** :
  - `$ cd /root`
  - `$ ls` **réponse** : `fichier_dans_mon_conteneur`
  - `$ cat fichier_dans_conteneur` **réponse** : `Ce fichier va être monté dans un conteneur`

4. Ménage

- **sur podman** :
  - `$ podman container rm conteneur_avec_montage_2`
  - `$ podman container rm conteneur_avec_montage`

5. Exercice : faciliter l'usage du conteneur Asciidoc avec un montage

5.2. Questions

podman container run --name alpine_asciidoctor --tty --interactive docker.io/alpine:3.22`

- **sur podman** :
  - `$ cd ~/tp-montage`
  - `$ mkdir tp_3_asciidoc`
  - ```
    $ podman container run --name alpine_asciidoctor \
              --tty \
              --interactive \
              --mount type=bind,source="~/tp-montage/tp_3_asciidoc",target="/main" \
              docker.io/alpine:3.22
    ```
- **sur le container** : `$ apk add asciidoctor && asciidoctor --version` -> ok
- **sur podman** :
  - `$ cd ~`
  - `$ cp ~/financiers.adoc tp-montage/tp_3_asciidoc`
- **sur le container** :
  - `$ cd main`
  - `$ ls` **réponse** : `financiers.adoc`
  - `$ asciidoctor financiers.adoc`
  - `$ ls` **réponse** : `financiers.adoc financiers.html`
- **sur podman** : `$ ls tp-montage/tp_3_asciidoc` **réponse** : `financiers.adoc financiers.html`
- **sur le container** : `$ exit`
- **sur podman** : `$ podman container rm alpine_asciidoctor`

6. Exercice : faciliter le déploiement de `economic_dispatch`

6.2. Questions

- **sur podman** :
  - `$ mkdir ~/tp-montage/tp_3_ed`
  - `cp ~/small_programs/5_economic_dispatch/economic_dispatch.julia ~/tp-montage/tp_3_ed/`
  - `$ cp ~/input_file.jl ~/tp-montage/tp_3_ed`
  - ```
    $ podman container run --name economic_dispatch \
              --tty \
              --interactive \
              --mount type=bind,source="~/tp-montage/tp_3_ed",target="/main" \
              docker.io/julia:1.11-trixie \
              bash
    ```

- **sur le container** :
  - `$ julia -e 'import Pkg; Pkg.add("JuMP")'`
  - `$ julia -e 'import Pkg; Pkg.add("HiGHS")'`
  - `julia 5_economic_dispatch/economic_dispatch.julia input_file.jl` -> tout fonctionne ; donc `$ exit`
- **sur podman** : `$ podman container rm economic_dispatch`

7. Exercice : déployer le logiciel docker-ffmpeg-converter

- **sur la machine** : Téléchargement de la vidéo puis `$ scp Téléchargements/Schlossbergbahn.webm podman:~/`
- **sur podman** :
  - `$ mkdir tp-montage/ffmpeg`
  - `$ mkdir tp-montage/ffmpeg/source`
  - `$ mkdir tp-montage/ffmpeg/destination`
  - ```
    $ podman container run --name ffmpeg \
                --tty \
                --interactive \
                --mount type=bind,source="~/tp-montage/ffmpeg/source",target="/source" \
                --mount type=bind,source="~/tp-montage/ffmpeg/destination",target="/destination" \
                --env SOURCE_DIRECTORY_PATH=/source \
                --env DESTINATION_DIRECTORY_PATH=/destination \
                --env GLOB_PATTERNS=*.webm \
                --env FFMPEG_ARGS=-loglevel error -y -fflags +genpts -i %s -r 24 %s.mp4 \
                ghcr.io/kennethwussmann/docker-ffmpeg-converter:1.2.0 bash
    ```
  - `$ mv ~/Schlossbergbahn ~/tp-montage/ffmpeg/source`
  - `$ ls ~/tp-montage/ffmpeg/destination` **réponse** : `Schlossbergbahn.mp4`
  - `$ scp podman:~/tp-montage/ffmpeg/destinationSchlossbergbahn.mp4 ~/`

TP4 Publication de ports réseaux

1. Tutoriel : déployer un serveur web nginx avec publication de port

1.1. Exécution avec publication de port

> MUDET Anthony Début : 8760 | Fin : 8769

- **sur podman** :
  - ```
    podman container run --rm \
                --detach \
                --name serveur_web_1 \
    ```

```
                    --publish 8760:80 \
                    docker.io/nginx:1.25
```

- `$ podman container ps`
  - **réponse** :
  - ```
    CONTAINER ID  IMAGE                           COMMAND
    CREATED         STATUS         PORTS                   NAMES
    d77eab3f389f  docker.io/library/nginx:1.25  nginx -g
    daemon o...  20 seconds ago  Up 21 seconds  0.0.0.0:8760-
    >80/tcp  server_web_1
    ```
- `$ podman container port serveur_web_1` **réponse** : 80/tcp -> 0.0.0.0:8760
- `$ podman container logs serveur_web_1`
- On va à l'url `http://podman:8760/` -> tout fonctionne
- `$ podman container stop serveur_web_1`

1.2. Exécution avec montage de répertoire

- **sur podman** :
  - `$ mkdir tp3 && cd tp3`
  - `$ nano index.html` et on ajoute le contenu :
  - ```
    <!DOCTYPE html>
    <html>

    <head>
        <meta charset="utf-8">
        <title>Premier déploiement</title>
    </head>

    <body>
        <h1>Bienvenue !</h1>
        <p>Ce déploiement a été réalisé par Anthony
    MUDET</p>
        <p>Si vous voyez cette page, c'est que tout a bien
    fonctionné.</p>
    </body>

    </html>
    ```
  - `$ pwd index.html` **réponse** : `/comptes/E253432U/tp4`
  - ```
    podman container run --rm \
                    --detach \
                    --name serveur_web_2 \
    ```
    ```

```
                    --publish 8760:80 \
                    --mount
type=bind,source="/comptes/E253432U/tp4",target="/usr/share/nginx
/html" \
                    docker.io/nginx:1.25
```

- On va à l'url `http://podman:8760/` -> tout fonctionne

1.3. Exécution de commandes dans un conteneur déjà en exécution

- **sur podman** : `$ podman container exec -ti serveur_web_2 bash`
- **sur le container** :
  - `$ ls /usr/share/nginx/html` **réponse** : `index.html`
  - `$ cat /usr/share/nginx/html/index.html`
    - **réponse** :
    - ```
      <!DOCTYPE html>
      <html>

      <head>
          <meta charset="utf-8">
          <title>Premier déploiement</title>
      </head>

      <body>
          <h1>Bienvenue !</h1>
          Ce déploiement a été réalisé par Anthony
      MUDET
          Si vous voyez cette page, c'est que tout a bien
      fonctionné.
      </body>

      </html>
      ```
  - `$ exit`

2. Exercice : déployer un service Gokapi avec publication de port

2.1. Préparation de l'environnement

- **sur podman** :
  - `$ mkdir -p ~/gokapi_data`
  - `$ mkdir -p ~/gokapi_config`

2.2. Déploiement du conteneur

- **sur podman** :
      ```

- `$ podman run -d \ --name gokapi_instance \ -p 8760:53842 \ -v ~/gokapi_data:/app/data:Z \ -v ~/gokapi_config:/app/config:Z \ docker.io/f0rc3/gokapi:v2.1.0`

2.3. Configuration via le Navigateur

On va à l'adresse `http://localhost:8760/setup/` et on rempli les champs avec username : `admin` et password : `my_password`

On va maintenant à l'adresse `http://localhost:8760/login` et on entre les credentials puis on upload l'image

2.4. Test de persistance

- **sur podman** :
  - `$ podman container stop gokapi_instance`
  - `$ podman container rm gokapi_instance`
  - ```
    $ podman run -d \
        --name gokapi_instance \
        -p 8760:53842 \
        -v ~/gokapi_data:/app/data:Z \
        -v ~/gokapi_config:/app/config:Z \
        docker.io/f0rc3/gokapi:v2.1.0
    ```

2.5. Vérification finale

- On va à l'adresse `http://localhost:8760/login` et on entre les credentials puis on vérifie que l'image est toujours là.

2.6. Nettoyage

Tout fonctionne, on peut donc nettoyer :

- **sur podman** :
  - `$ podman container stop gokapi_instance`
  - `$ podman container rm gokapi_instance`
  - `$ rm -rf ~/gokapi_data`
  - `$ rm -rf ~/gokapi_config`

3. Exercice : déployer un service Kanboard avec publication de port

3.1. Préparation

- **sur podman** :
  - `$ mkdir -p ~/kanboard_data`
  - `$ chmod 777 ~/kanboard_data`
    ```

3.2. Déploiement du conteneur

- **sur podman** :
  - ```
    $ podman run -d \
        --name kanboard_instance \
        -p 8761:80 \
        -v ~/kanboard_data:/var/www/app/data:Z \
        docker.io/kanboard/kanboard:v1.2.32
    ```

3.4. Utilisation et test

- On va à l'adresse `http://localhost:8761/` et on se connecte avec username : `admin` et password : `admin`
- On créer un nouveau projet avec le nom `TP4` et on sauvegarde
- On va sur le board du projet et on crée une nouvelle tâche `Faire le TP4 de DevOps`
- On glisse et dépose la tâche dans la colonne `Ready`

3.5 Test de persistance

- **sur podman** :
  - `$ podman container stop kanboard_instance`
  - `$ podman container rm kanboard_instance`
  - ```
    $ podman run -d \
        --name kanboard_instance \
        -p 8761:80 \
        -v ~/kanboard_data:/var/www/app/data:Z \
        docker.io/kanboard/kanboard:v1.2.32
    ```

3.6. Vérification finale

- On va à l'adresse `http://localhost:8761/` et on se connecte avec username : `admin` et password : `admin`
- On vérifie que le projet `TP4` est toujours là avec la tâche `Faire le TP4 de DevOps`

3.7. Nettoyage

- **sur podman** :
  - `$ podman container stop kanboard_instance`
  - `$ podman container rm kanboard_instance`
  - `$ rm -rf ~/kanboard_data`

4. Exercices supplémentaires

4.0. Tunnel SSH

- **sur la machine** : `$ ssh -L 8760:127.0.0.1:8760 -L 8761:127.0.0.1:8761 -L 8762:127.0.0.1:8762 E253432U@podman.ensinfo.sciences.univ-nantes.prive`

4.1. Déploiement du serveur IRC (InspIRCd)

- **sur podman** :
  - `mkdir ~/inspircd_data ~/inspircd_config`
  - ```
    $ podman run -d \
      --name inspircd_server \
      -p 8762:6667 \
      -v ~/inspircd_config:/inspircd/conf:Z \
      -v ~/inspircd_data:/inspircd/data:Z \
      docker.io/inspircd/inspircd-docker:latest
    ```

B - Test avec le client kirc

- **sur la machine** :
  - `$ wget https://github.com/mcpcpc/kirc/releases/download/0.3.2/kirc`
  - `$ chmod +x kirc`

C - Connexion au serveur IRC

- **sur la machine** : `$ ./kirc -s localhost -p 8762 -c`

4.2. Déploiement de Automatic-Image-Converter

- **sur podman** :
  - `$ mkdir -p ~/aic_input ~/aic_output`
  - ```
    $ podman run -d \
        --name aic_instance \
        -p 8763:8080 \
        -v ~/aic_input:/input:Z \
        -v ~/aic_output:/output:Z \
        docker.io/kennethwussmann/automatic-image-converter:latest
    ```

D - Test de conversion d'image

- **sur la machine** : `$ scp -P 22 test.jpg E253432U@podman.ensinfo.sciences.univ-nantes.prive:~/aic_input/`

- **sur podman** : `$ ls -l ~/aic_output/` **réponse** : `test.png` -> tout fonctionne

E - Nettoyage

- **sur podman** :
  - `$ podman container stop inspircd_server aic_instance`
  - `$ podman container rm inspircd_server aic_instance`
  - `$ rm -rf ~/inspircd_data ~/inspircd_config ~/aic_input ~/aic_output`

TP5 Mise en réseau de plusieurs conteneurs

1. Tutoriel : premiers réseaux virtuels

1.1. Deux conteneurs sur un même réseau virtuel

1.1.1. Mise en place

- **sur podman (terminal 1)** :
  - `$ podman network create r1`
  - `$ podman container run --name conteneurA --rm -ti --network r1 docker.io/alpine:3.22 bash`
- **sur le containerA (terminal 1)** : `$ apk update && apk add netcat-openbsd`
- **sur podman (terminal 2)** : `$ podman container run --name conteneurB --rm -ti --network r1 docker.io/alpine:3.22 bash`
- **sur le containerB (terminal 2)** : `$ apk update && apk add netcat-openbsd`

1.1.2. Test de communication

- **sur le containerA (terminal 1)** : `$ nc -l -p 8760`
- **sur le containerB (terminal 2)** : `$ nc conteneurA 8760` puis tape `Salut depuis le conteneur B !` et appuie sur Entrée
- **sur le containerA (terminal 1)** : on voit apparaître `Salut depuis le conteneur B !`

1.2. Trois conteneurs sur deux réseaux virtuels

1.2.1. Configuration des réseaux

- **sur podman (terminal 3)** : `$ podman network create r2`
- **sur podman (terminal 2)** : `$ podman container run --name conteneurB --rm -ti --network r1 --network r2 docker.io/alpine:3.22 bash`
- **sur le containerB (terminal 2)** : `$ apk update && apk add netcat-openbsd`
- **sur podman (terminal 3)** : `$ podman container run --name conteneurC --rm -ti --network r2 docker.io/alpine:3.22 bash`
- **sur le containerC (terminal 3)** : `$ apk update && apk add netcat-openbsd`

1.2.2. Test de cloisonnement

1. Le pivot (B peut parler à tout le monde)

- **sur le containerA (terminal 1)** : `$ nc -l -p 8760`
- **sur le containerC (terminal 3)** : `$ nc -l -p 8760`
- **sur le containerB (terminal 2)** : `$ nc conteneurA 8760` puis tape `Salut depuis le conteneur B vers A !` et appuie sur Entrée
- **sur le containerA (terminal 1)** : on voit apparaître `Salut depuis le conteneur B vers A !`
- **sur le containerB (terminal 2)** : `$ nc conteneurC 8760` puis tape `Salut depuis le conteneur B vers C !` et appuie sur Entrée
- **sur le containerC (terminal 3)** : on voit apparaître `Salut depuis le conteneur B vers C !`

2. L'isolation (A ne peut pas parler à C)

- **sur le containerC (terminal 3)** : `$ nc -l -p 8760`
- **sur le containerA (terminal 1)** : `$ nc conteneurC 8760` puis tape `Salut depuis le conteneur A vers C !` et appuie sur Entrée
- **sur le containerC (terminal 3)** : rien

1.3. Nettoyage

- **sur podman (terminal 1)** : `$ podman network rm r1`
- **sur podman (terminal 3)** : `$ podman network rm r2`

2. Tutoriel : déployer un service Mealie à l'aide d'un réseau virtuel

2.1. Réseau virtuel

- **sur podman** :
  - `$ podman network create mealie_network`
  - `$ mkdir -p ~/mealie_deployment/db-data`
  - `$ cd ~/mealie_deployment`

2.2. Déploiement de la base de données PostgreSQL

- **sur podman** :

```
        $ podman container run --rm \
            --detach \
            --name mealie_db \
            --network mealie_network \
            --mount type=bind,source="$(pwd)/db-
  data",target="/var/lib/postgresql/data" \
            --env POSTGRES_USER=mealie \
            --env POSTGRES_PASSWORD=mealie_password \
            --env POSTGRES_DB=mealie_data \
            docker.io/postgres:16
```

2.3. Déploiement de Mealie

- **sur podman** :
  - ```
    $ podman container run --rm \
        --detach \
        --name mealie \
        --network mealie_network \
        --publish 8763:9000 \
        --env DB_ENGINE=postgres \
        --env POSTGRES_SERVER=mealie_db \
        --env POSTGRES_DB=mealie_data \
        --env POSTGRES_USER=mealie \
        --env POSTGRES_PASSWORD=mealie_password \
        --env POSTGRES_PORT=5432 \
        ghcr.io/mealie-recipes/mealie:v1.0.0-RC1.1
    ```
  - `$ podman container ps` pour vérifier que les deux conteneurs sont bien en cours d'exécution.

2.4. Test et validation

- On va à l'adresse `http://localhost:8763/` et on crée un compte admin avec username : `admin` et password : `mealie_admin_password`
- Clique sur le bouton "Créer" (Create) -> "Importer depuis l'URL" (Import from URL).
- Colle l'URL demandée : https://deliacious.com/2020/03/babka-cannelle-vegan.html
- Clique sur "Importer la recette" (Import Recipe).
- Vérifie que la recette apparaît dans ta liste de recettes.

2.5. Nettoyage

- **sur podman** :
  - `$ podman container stop mealie mealie_db`
  - `$ podman network rm mealie_network`
  - `rm -rf ~/mealie_deployment`

3. Exercice : déployer un service Kanboard+MariaDB avec un réseau virtuel

3.1. Préparation

- **sur podman** :
  - `$ podman network create kanboard_net`
  - `$ mkdir -p ~/kanboard_full/db_data`
  - `$ mkdir -p ~/kanboard_full/kb_data`

3.2. Déploiement de la base de données MariaDB

- **sur podman** :
    ```

```
$ podman run -d \
    --name kanboard_db \
    --network kanboard_net \
    --mount
type=bind,source=~/kanboard_full/db_data,target=/var/lib/mysql,Z
\
    --env MARIADB_USER=kanboard \
    --env MARIADB_PASSWORD=secret_password \
    --env MARIADB_DATABASE=kanboard_db \
    --env MARIADB_ROOT_PASSWORD=root_secret \
    docker.io/library/mariadb:11.1
```

3.3. Déploiement de Kanboard

- **sur podman** :

```
$ podman run -d \
    --name kanboard_instance \
    -p 8764:80 \
    --network kanboard_net \
    --mount
type=bind,source=~/kanboard_full/kb_data,target=/var/www/app/data
,Z \
    --env
DATABASE_URL="mysql://kanboard:secret_password@kanboard_db:3306/k
anboard_db" \
    docker.io/kanboard/kanboard:v1.2.32
```

3.4. Test et validation

- On va à l'adresse `http://localhost:8764/` et on se connecte avec username : `admin` et password : `admin`
- Clique sur l'icône de profil (en haut à droite) -> "My profile" (ou Settings/Préférences).
- Dans le menu de gauche, clique sur "Information".
- Cherche la ligne Database driver.
- Vérifie que le driver affiché est bien "mysql".

3.5. Nettoyage

- **sur podman** :
  - `$ podman container stop kanboard_instance kanboard_db`
  - `$ podman container rm kanboard_instance kanboard_db`
  - `$ podman network rm kanboard_net`
  - `rm -rf ~/kanboard_full`

4. Exercice : déployer lldap, Kanboard et Mealie à l'aide de réseaux virtuels

4.1. Préparation

- **sur podman** :
  - `$ podman network create sso_network`
  - `$ mkdir -p ~/final_tp/lldap_data`
  - `$ mkdir -p ~/final_tp/mkdir -p ~/final_tp/kanboard_db`
  - `$ mkdir -p ~/final_tp/mealie_data ~/final_tp/mealie_db`

4.2. Déploiement de lldap

- **sur podman** :

```
$ podman run -d \
    --name lldap_instance \
    --network sso_network \
    -p 8765:17170 \
    -v ~/final_tp/lldap_data:/data:Z \
    -e LLDAP_JWT_SECRET=secret_jwt_super_long_et_complexe \
    -e LLDAP_KEY_SEED=secret_seed_super_long_et_complexe \
    -e LLDAP_LDAP_BASE_DN=dc=monannuaire \
    -e LLDAP_LDAP_USER_PASS=motdepasse \
    docker.io/lldap/lldap:2025-10-14
```

  - On ouvre un navigateur à l'adresse `http://localhost:8765/` et on crée un utilisateur admin avec username : `admin` et password : `admin_password`
  - Clique sur "Create User".
  - ID: etudiant / Email: etudiant@test.com / Password: superpassword.
  - Valide et ajoute l'utilisateur au groupe "lldap_users"

4.3. Déploiement de Kanboard + MariaDB

1. Déploiement de la base de données MariaDB

- **sur podman** :

```
$ podman run -d \
    --name kanboard_db \
    --network sso_network \
    --mount
type=bind,source=~/final_tp/kanboard_db,target=/var/lib/mysql,Z \
    --env MARIADB_USER=kanboard \
    --env MARIADB_PASSWORD=secret_password \
    --env MARIADB_DATABASE=kanboard_db \
    --env MARIADB_ROOT_PASSWORD=root_secret \
    docker.io/library/mariadb:11.1
```

2. Déploiement de Kanboard

- **sur podman** :

```
        $ podman run -d \
            --name kanboard_instance \
            --network sso_network \
            -p 8766:80 \
            -v ~/final_tp/kanboard_data:/var/www/app/data:Z \
            -e
DATABASE_URL="mysql://kanboard:kb_secret@kanboard_db/kanboard" \
            -e LDAP_AUTH=true \
            -e LDAP_BIND_TYPE=user \
            -e LDAP_SERVER="ldap://lldap_instance:3890" \
            -e LDAP_USERNAME="uid=%s,ou=people,dc=monannuaire" \
            -e LDAP_USER_BASE_DN="ou=people,dc=monannuaire" \
            -e LDAP_USER_FILTER="uid=%s" \
            docker.io/kanboard/kanboard:v1.2.32
```

3. Test et validation : Va sur http://localhost:8766. Essaie de te connecter avec l'utilisateur créé dans l'étape 2 (etudiant / superpassword)

4.4. Déploiement de Mealie

1. Déploiement de la base de données PostgreSQL

- **sur podman** :

```
        $ podman run -d \
            --name mealie_db \
            --network sso_network \
            -v ~/final_tp/mealie_db:/var/lib/postgresql/data:Z \
            -e POSTGRES_USER=mealie \
            -e POSTGRES_PASSWORD=mealie_secret \
            -e POSTGRES_DB=mealie \
            docker.io/postgres:16
```

2. Déploiement de Mealie

- **sur podman** :

```
        $ podman run -d \
            --name mealie_instance \
            --network sso_network \
            -p 8767:9000 \
            -v ~/final_tp/mealie_data:/app/data:Z \
            -e DB_ENGINE=postgres \
            -e POSTGRES_SERVER=mealie_db \
```

```
            -e POSTGRES_USER=mealie \
            -e POSTGRES_PASSWORD=mealie_secret \
            -e POSTGRES_DB=mealie \
            -e POSTGRES_PORT=5432 \
            -e LDAP_AUTH_ENABLED=true \
            -e LDAP_SERVER_URL="ldap://lldap_instance:3890" \
            -e LDAP_BASE_DN="ou=people,dc=monannuaire" \
            -e LDAP_QUERY_BIND="cn=admin,ou=people,dc=monannuaire"
 \
            -e LDAP_QUERY_PASSWORD="motdepasse" \
            -e LDAP_ID_ATTRIBUTE=uid \
            -e LDAP_NAME_ATTRIBUTE=uid \
            -e LDAP_MAIL_ATTRIBUTE=mail \
            ghcr.io/mealie-recipes/mealie:v1.0.0-RC1.1
```

3. Test et validation : Va sur http://localhost:8767. Essaie de te connecter avec etudiant / superpassword.

4.5. Résumé :

1. LDAP est accessible sur localhost:8765 et contient ton user.
2. Kanboard est accessible sur localhost:8766 et te logue via LDAP.
3. Mealie est accessible sur localhost:8767 et te logue via LDAP.

4.6. Nettoyage

- **sur podman** :
  - `$ podman rm -f lldap_instance kanboard_instance kanboard_db mealie_instance mealie_db`
  - `$ podman network rm sso_network`
  - `$ podman unshare rm -rf ~/final_tp`

TP6 Créer manuellement des images de conteneurs

1. Préparatifs

1.2. Authentification podman

- **sur podman** : `$ podman login docker-registry.univ-nantes.fr` : Username : `E253432U` + password -> Login succeeded

2. Tutoriel : une première image Ubuntu avec cowsay

2.2. Création de l'image (v1)

1. On lance le container modèle :

- **sur podman** : `$ podman container run -ti --name cowsay_template docker.io/ubuntu:24.04`

2. On installe cowsay dans le container modèle :

- **sur le container** :
  - `$ apt update && apt install -y cowsay`
  - `$ exit`

3. Créer l'image :

- **sur podman** : `$ podman container commit cowsay_template tutoriel-cowsay:v1`

4. Vérification de la creation :

- **sur podman** :
  - `$ podman image ls | grep tutoriel-cowsay`
    - **réponse** :
    - ```
      REPOSITORY                                  TAG
      IMAGE ID      CREATED             SIZE
      localhost/tutoriel-cowsay                   v1
      fc7a1fa489d2  About a minute ago  189 MB
      ```

5. Test de l'image :

- **sur podman** : `$ podman container run -ti tutoriel-cowsay:v1 bash`
- **sur le container** : `$ cowsay "Bonjour le monde !"` -> tout fonctionne ; donc `$ exit`

2.3. Publication de l'image

1. Publication de l'image sur le registre distant :

   - **sur podman** : `$ podman push tutoriel-cowsay:v1 docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/tutoriel-cowsay:v1`

2. Vérification de la publication :

   - **sur podman** : `$ podman container run -ti docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/tutoriel-cowsay:v1` -> tout fonctionne ; donc `$ exit`

2.4. Ajout d'un point d'entrée (v2) :

1. Créer la version v2 avec CMD : On réutilise le conteneur template cowsay_template (qui existe toujours éteint) pour créer une v2 qui lance cowsay automatiquement.

- `sur podman` : `$ podman container commit cowsay_template --change 'CMD /usr/games/cowsay $MESSAGE' tutoriel-cowsay:v2`

2. Test de la version v2 :
      ```

- **sur podman** : `$ podman container run -ti -e MESSAGE="Bonjour le monde en v2 !" tutoriel-cowsay:v2` -> tout fonctionne

2.5. Republication de l'image v2

1. Publication de l'image v2 sur le registre distant :
    - **sur podman** : `$ podman push tutoriel-cowsay:v2 docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/tutoriel-cowsay:v2` -> On a bien v1 et v2

3. Exercice : image Alpine avec Asciidoctor

3.2. Questions :

1. est-il nécessaire de pouvoir configurer l'exécution du logiciel conteneurisé (à savoir Asciidoctor) ? si oui, quel mode de configuration est possible et pourrait être mis en place ? Oui, il faut indiquer quel fichier convertir. On utilisera une variable d'environnement (ex: `DOC_FILE`)

2. quel point d'entrée définir pour l'image ? comment articuler ce point d'entrée avec le mode de configuration choisi au point précédent ? La commande sera asciidoctor /data/$DOC_FILE

3. est-ce qu'un montage sera nécessaire lors de la phase d'usage de l'image ? lequel et pourquoi ? comment rendre cohérent ce point de montage avec le mode de configuration et avec le point d'entrée définis précédemment ? Un montage de volume est indispensable pour que le conteneur accède au fichier source .adoc et puisse écrire le fichier .html résultant sur la machine hôte.

3.3. Réalisation

1. Création du conteneur template et installation

    - **sur podman** : `$ podman run -ti --name asciidoctor_template docker.io/alpine:3.22`
    - **sur le container** :
        - `$ apk update`
        - `$ apk add asciidoctor`
        - `$ mkdir /data`
        - `$ exit`

2. Transformation en image :

    - **sur podman** : `$ podman container commit asciidoctor_template --change 'CMD asciidoctor /data/$DOC_FILE' exercice-asciidoctor:1.0`

3. Test de l'image :

- **sur podman** :
    - `$ echo "= Mon Titre" > test.adoc`
    - `echo "Ceci est un test." >> test.adoc`
    - `$ cp ~/financiers.adoc ~/tp-images/asciidoctor_test/`

```
$ podman container run --rm \
    -v $(pwd):/data:Z \
    -e DOC_FILE=test.adoc \
    localhost/exercice-asciidoctor:1.0
```

- `$ ls -l test.html` -> existe

4. Publication sur gitlab

- **sur podman** : `$ podman image push localhost/exercice-asciidoctor:1.0 docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/exercice-asciidoctor:1.0`

4. Tutoriel : une image pour déployer un site web avec nginx

4.2. Création de l'image

1. Création du conteneur template et installation

- **sur podman** : `$ podman container run -ti --name nginx_template docker.io/nginx:1.25 bash`

2. Nettoyage du dossier html :

- **sur le container** :
  - `$ rm /usr/share/nginx/html/*`

3. Copie du site web dans le conteneur :

- **sur podman** : `$ podman container cp test.html nginx_template:/usr/share/nginx/html/index.html`

4. Verification :

- **sur le container** : `$ ls /usr/share/nginx/html/` **réponse** : `index.html`

5. Création de l'image :

- **sur podman** : `$ podman container commit nginx_template --change 'CMD nginx -g "daemon off;"' tutoriel-nginx:1`

6. Test de l'image :

- **sur podman** :

```
$ podman container run --rm \
    --detach \
    --name nouveau_serveur_web \
    --publish 8768:80 \
    localhost/tutoriel-nginx:1
```

- On va à l'url `http://podman:8760/` -> tout fonctionne

4.3. Publication de l'image

- **sur podman** : `$ podman image push tutoriel-nginx:1 docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/tutoriel-nginx:1`

5. Exercice : une image de conteneur pour economic_dispatch

5.2. Questions

1. Faites un travail préliminaire de conception de votre future image de conteneur.

- Configuration : L'outil a besoin d'un fichier d'entrée (scénario JSON).
- Point d'entrée : L'exécution de Julia sur le script principal (main.jl).
- Montage : Un volume /data pour fournir le fichier JSON depuis l'hôte vers le conteneur.
- Commande finale : julia --project=/app /app/src/main.jl /data/scenario.json

5.3. Réalisation

- **sur podman** : `$ podman run -ti --name dispatch_template docker.io/julia:1.11-trixie bash`
- **dans le container** :
  - `$ apt update && apt install -y git`
  - `$ git clone https://gitlab.univ-nantes.fr/bousse-e/economic_dispatch.git /app`
  - `$ cd /app`
  - `$ julia --project=. -e 'import Pkg; Pkg.instantiate();'`
  - `$ exit`

5.4. Création de l'image

- **sur podman** : `$ podman container commit dispatch_template --change 'WORKDIR /app' --change 'CMD ["julia", "--project=/app", "src/main.jl", "/data/input.json"]' exercice-economic-dispatch:a`

5.5. Test de l'image

- **sur podman** :
  - ```
    $ podman container run --rm \
        exercice-economic-dispatch:a \
        julia --project=/app src/main.jl data/example.json
    ```
  - **réponse** : plusieurs lignes de logs -> tout fonctionne

5.6. Publication de l'image
    ```

- **sur podman** : `$ podman image push exercice-economic-dispatch:a docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/exercice-economic-dispatch:a`

6. Exercice : créer une image pour Web Calculator (Python)

6.2. Questions

1. Faites un travail préliminaire de conception de votre future image de conteneur.

- Image de base : docker.io/python:3.11 (contient déjà Python et pip, pas besoin de venv).
- Installation : git pour récupérer le code, puis pip install -r requirements.txt pour les dépendances.
- Point d'entrée (CMD) : Lancement du serveur web Uvicorn. Important : Il faut spécifier l'hôte 0.0.0.0 pour que le conteneur accepte les connexions venant de l'extérieur (le mapping de port Podman).

6.3. Réalisation

1. Préparation du template

- **sur podman** : `$ podman run -ti --name webcalc_template docker.io/python:3.11 bash`
- **dans le container** :
  - `$ apt update && apt install -y git`
  - `$ git clone https://gitlab.univ-nantes.fr/bousse-e/webcalculator.git /app`
  - `$ cd /app`
  - `$ pip install -r requirements.txt`
  - `$ ls -R`
  - `$ exit`

2. Transformation en image

- **sur podman** :

```
       podman container commit webcalc_template \
         --change 'WORKDIR /app' \
         --change 'CMD ["uvicorn", "src.main:app", "--host",
"0.0.0.0", "--port", "8000"]' \
         webcalculator:dev
```

3. Test de l'image

- **sur podman** :

```
       $ podman container run --rm \
           --detach \
           --name mon_calculateur \
```

```
            -p 8769:8000 \
            localhost/webcalculator:dev
```

- On va à l'url `http://podman:8769/` -> tout fonctionne

6.4. Publication de l'image

- **sur podman** : `$ podman image push localhost/webcalculator:dev docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/webcalculator:dev`

TP7 Scripter la création d'une image de conteneur

3. Tutoriel : image cowsay via un Containerfile

3.2. Premier Containerfile (Version v3 - Installation simple)

1. Création du Containerfile

```
# On écrit le contenu dans le fichier Containerfile
cat > Containerfile <<EOF
FROM docker.io/ubuntu:24.04
RUN apt update
RUN apt install -y cowsay
EOF
```

2. Construction de l'image

- **sur podman** : `$ podman build --tag tutoriel-cowsay:v3 .`

3. Test de l'image

- **sur podman** : `$ podman container run --rm -ti localhost/tutoriel-cowsay:v3`
- **sur le container** : `$ cowsay "Bonjour le monde en v3 !"` -> tout fonctionne ; donc `$ exit`

3.3. Ajout d'un point d'entrée avec CMD (Version v4)

1. Modification du Containerfile

```
# On écrase le fichier précédent avec la nouvelle version
cat > Containerfile <<EOF
FROM docker.io/ubuntu:24.04
RUN apt update
RUN apt install -y cowsay
CMD /usr/games/cowsay \$MESSAGE
EOF
```

2. Construction de l'image v4

- **sur podman** : `$ podman build --tag tutoriel-cowsay:v4 .`

3. Test de l'image v4

- **sur podman** : `$ podman container run --rm -ti -e MESSAGE="Bonjour le monde en v4 !" localhost/tutoriel-cowsay:v4` -> tout fonctionne

4. Exercice : image Asciidoctor via Containerfile

4.1. Préparation

- **sur podman** : `$ mkdir -p ~/asciidoctor-containerfile && cd ~/asciidoctor-containerfile`

4.2. Création du Containerfile

```
cat > Containerfile <<EOF
FROM docker.io/alpine:3.22

# Installation des dépendances
RUN apk update && apk add asciidoctor

# Définition du répertoire de travail (optionnel mais propre)
WORKDIR /data

# Commande par défaut utilisant la variable d'environnement
CMD asciidoctor /data/\$DOC_FILE
EOF
```

4.3. Construction de l'image

- **sur podman** : `$ podman image build --tag exercice-asciidoctor:2.0 .`

4.4. Test de l'image

- **sur podman** :
  - ```
    $ echo "= Mon Titre" > test.adoc
    echo "Ceci est un test." >> test.adoc
    $ cp ~/financiers.adoc ~/asciidoctor-containerfile/
    $ podman container run --rm \
        -v $(pwd):/data:Z \
        -e DOC_FILE=test.adoc \
        localhost/exercice-asciidoctor:2.0
    ```
  - `$ ls -l test.html` -> existe
    ```

4.5. Publication de l'image

- **sur podman** : `$ podman image push localhost/exercice-asciidoctor:2.0 docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/exercice-asciidoctor:2.0`

5. Tutoriel : image site web nginx via Containerfile

5.1. Préparation de l'espace de travail

- **sur podman** :
  - `$ mkdir -p ~/nginx-containerfile`
  - `$ cd ~/nginx-containerfile`
  - `$ cp ~/asciidoctor-containerfile/test_v2.html ./index.html`

5.2. Création du Containerfile

```
cat > Containerfile <<EOF
FROM docker.io/nginx:1.25

# Suppression de la page d'accueil par défaut
RUN rm /usr/share/nginx/html/*

# Copie de notre fichier index.html dans le dossier web de Nginx
COPY ./index.html /usr/share/nginx/html/index.html
EOF
```

Alternative : WORKDIR

```
FROM docker.io/nginx:1.25
WORKDIR /usr/share/nginx/html
RUN rm -rf ./*
COPY ./index.html .
```

5.3. Construction de l'image

- **sur podman** : `$ podman build --tag tutoriel-nginx:2 .`

5.4. Exécution du conteneur

- **sur podman** :
  - `$ podman rm -f nouveau_serveur_web` (si déjà existant)
  -
```
$ podman container run --rm \
    --detach \
    --name nouveau_serveur_web \
```

```
            --publish 8770:80 \
            localhost/tutoriel-nginx:2
```

- On va à l'url `http://podman:8770/` -> tout fonctionne

6. Exercice : créer une image pour Web Calculator (Python)

6.2. Préparatifs

- **sur podman** :
  - `$ mkdir -p ~/webcalc-containerfile`
  - `$ cd ~/webcalc-containerfile`
  - `$ git clone https://gitlab.univ-nantes.fr/bousse-e/webcalculator.git ./`
  - `$ cd webcalculator`

6.3. Création du Containerfile

```
cat > Containerfile <<EOF
# Image de base officielle Python
FROM docker.io/python:3.11

# Définition du répertoire de travail dans le conteneur
WORKDIR /app

# Copie des fichiers du projet (hôte) vers le conteneur
COPY . /app

# Installation des dépendances Python
RUN pip install --no-cache-dir -r requirements.txt

# Commande de démarrage (Serveur Web)
# --host 0.0.0.0 est CRUCIAL pour que le conteneur accepte les connexions
externes
CMD ["uvicorn", "src.main:app", "--host", "0.0.0.0", "--port", "8000"]
EOF
```

6.4. Construction de l'image

- **sur podman** : `$ podman build --tag webcalculator:prod .`

6.5. Test de l'image

- **sur podman** :

```
        $ podman container run --rm \
            --detach \
            --name mon_calculateur_prod \
```

```
            -p 8771:8000 \
            localhost/webcalculator:prod
```

- On va à l'url `http://podman:8771/` -> tout fonctionne

6.6. Publication de l'image

- **sur podman** : `$ podman image push localhost/webcalculator:prod docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/webcalculator:prod`

7. Exercice : créer une image webdav-embedded-server (Java)

7.2. Préparatifs

- **sur podman** :
  - `$ mkdir -p ~/webdav-containerfile`
  - `$ cd ~/webdav-containerfile`
  - `$ git clone https://gitlab.univ-nantes.fr/bousse-e/webdav-embedded-server.git`
  - `$ cd webdav-embedded-server`

7.3. Création de l'image

Points clés de la conception :

- Image de base : gradle:8-jdk17 (contient tout le nécessaire pour compiler).
- Proxy : Indispensable pour la commande gradle fatJar (sinon Gradle ne pourra pas télécharger les dépendances depuis Internet).
- Entrée : On utilise des variables d'environnement (ENV) pour rendre l'utilisateur et le mot de passe configurables au démarrage.

- Construction de l'image via un Containerfile :

```
# Image de base contenant Gradle et Java
FROM docker.io/gradle:8-jdk17

# Répertoire de travail
WORKDIR /app

# Copie des sources
COPY . .

# Compilation avec configuration du PROXY de l'université
# C'est ici qu'on injecte les drapeaux -Dhttps.proxyHost...
RUN gradle -Dhttps.proxyHost=proxy.ensinfo.sciences.univ-nantes.prive -
Dhttps.proxyPort=3128 fatJar

# Création du dossier de stockage des données
```

```
RUN mkdir /data

# Variables d'environnement par défaut (surchargeables au run)
ENV WEBDAV_USER=admin
ENV WEBDAV_PASS=admin

# Port exposé par le conteneur
EXPOSE 8080

# Lancement du serveur
# On pointe vers le FatJar généré dans build/libs/
CMD java -jar build/libs/webdav-embedded-server-0.2.1-SNAPSHOT-fatjar.jar -
-credentials "\$WEBDAV_USER:\$WEBDAV_PASS" --port 8080 /data
```

- Construction de l'image :
- **sur podman** : `$ podman image build --tag webdavserver:latest .`
- Déploiement et Test :
- **sur podman** :
  - `mkdir -p ~/webdav-data`
  - ```
    $ podman container run --rm \
        --detach \
        --name mon_webdav \
        -p 8770:8080 \
        -v ~/webdav_data:/data:Z \
        -e WEBDAV_USER=etudiant \
        -e WEBDAV_PASS=supersecret \
        localhost/webdavserver:latest
    ```
  - On va à l'url `http://podman:8772/` -> tout fonctionne (login : etudiant / superpassword)
- Publication de l'image
- **sur podman** : `$ podman image push localhost/webdavserver:latest docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-images/webdavserver:latest`

7.4. Amélioration de l'image

Pourquoi l'image actuelle est-elle "mauvaise" ? Si vous faites podman image list, vous verrez que l'image webdavserver fait probablement plus de 600 Mo (voire 800 Mo).

- Raison : Elle contient le code source, Gradle, le cache de compilation, et tout le JDK (compilateur Java).
- Besoin réel : Pour exécuter le programme, on a juste besoin d'un JRE (Java Runtime Environment) et du fichier .jar final. Solution : Le Multi-stage Build On utilise une image pour compiler (Stage 1), et on
    ```

copie uniquement le résultat dans une image finale très légère (Stage 2).

- Nouveau Containerfile avec Multi-stage Build :

```
# --- Étape 1 : Builder (Grosse image avec outils) ---
FROM docker.io/gradle:8-jdk17 AS builder
WORKDIR /app
COPY . .
RUN gradle -Dhttps.proxyHost=proxy.ensinfo.sciences.univ-nantes.prive -
Dhttps.proxyPort=3128 fatJar

# --- Étape 2 : Runner (Petite image juste pour exécuter) ---
FROM docker.io/eclipse-temurin:17-jre-alpine
WORKDIR /app

# On copie UNIQUEMENT le jar depuis l'étape 1
COPY --from=builder /app/build/libs/webdav-embedded-server-0.2.1-SNAPSHOT-
fatjar.jar /app/server.jar

RUN mkdir /data
ENV WEBDAV_USER=admin
ENV WEBDAV_PASS=admin

CMD ["java", "-jar", "/app/server.jar", "--credentials",
"\$WEBDAV_USER:\$WEBDAV_PASS", "--port", "8080", "/data"]
```

8. Exercice : créer une image WordcloudGenerator (langage R)

8.2. Préparatifs

- **sur podman** :
  - `$ mkdir -p ~/wordcloud-containerfile`
  - `$ cd ~/wordcloud-containerfile`
  - `$ git clone https://gitlab.univ-nantes.fr/bousse-e/wordcloud-generator.git`
  - `$ cd wordcloud-generator`

8.3. Conception et Création du Containerfile

1. Image de base : docker.io/r-base:latest. C'est une image basée sur Debian.
2. Installation de libxml2 : Puisque c'est une base Debian, on utilise apt. Il faut installer libxml2-dev (la version de développement) pour pouvoir compiler les paquets R qui en dépendent.
3. Paquets R : On utilise install.packages via Rscript.
4. Point d'entrée : On lance R pour exécuter l'application Shiny sur le port 8000 et l'hôte 0.0.0.0.

- Containerfile :

```
FROM docker.io/r-base:latest

# Installation des dépendances système (bibliothèque C requise pour le
package XML)
# On nettoie le cache apt à la fin pour réduire la taille de l'image
RUN apt-get update && apt-get install -y \
    libxml2-dev \
    && rm -rf /var/lib/apt/lists/*

# Répertoire de travail
WORKDIR /app

# Copie du code source
COPY . /app

# Installation des paquets R nécessaires
# Attention : Cette étape est longue car elle compile les sources
RUN Rscript -e 'install.packages(c("shiny", "tm", "wordcloud", "memoise",
"XML"), repos="http://cran.rstudio.com/")'

# Exposition du port
EXPOSE 8000

# Commande de démarrage
# On lance l'application Shiny sur le port 8000 et on écoute sur toutes les
interfaces (0.0.0.0)
CMD ["R", "-e", "shiny::runApp('/app', port=8000, host='0.0.0.0')"]
```

8.4. Construction de l'image

- **sur podman** : `$ podman build --tag wordcloudgenerator:latest .`

8.5. Test de l'image

- **sur podman** :
  - ```
    $ podman container run --rm \
        --detach \
        --name mon_wordcloud \
        -p 8773:8000 \
        localhost/wordcloudgenerator:latest
    ```
  - On va à l'url `http://podman:8773/` -> tout fonctionne

8.6. Publication de l'image

- **sur podman** : `$ podman image push localhost/wordcloudgenerator:latest docker-registry.univ-nantes.fr/e253432u/developpement_exploitation-tp-`
    ```

```
images/wordcloudgenerator:latest
```

8.7. Nettoyage

- **sur podman** :
  - `$ podman stop mon_wordcloud`
  - `$ podman rmi wordcloudgenerator:latest`

- Développement et Exploitation - DevOps DataOps
- Master 1 Informatique
- Nantes Université
- 2025 - 2026
- Anthony MUDET - M1 Informatique parcours SMART Computing