Version : **2024.01**

Dernière mise-à-jour : 2024/11/22 08:50

RH13409 - Gestion des Conteneurs avec Podman

Contenu du Cours

Présentation de la Virtualisation par Isolation

Un isolateur est un logiciel qui permet d'isoler l'exécution des applications dans des **containers**, des **contextes** ou des **zones d'exécution**.

Historique

- **1979** - chroot - l'isolation par changement de racine,
- **2000** - BSD Jails - l'isolation en espace utilisateur,
- **2004** - Solaris Containers - l'isolation par **zones**,
- **2005** - OpenVZ - l'isolation par **partitionnement du noyau** sous Linux,
- **2008** - LXC - LinuX Containers - l'isolation en utilisant des **namespaces** et des **CGroups** avec **liblxc**,
- **2013** - Docker - l'isolation en utilisant des **namespaces** et des **CGroups** avec **libcontainer**,
- **2014** - LXD - LinuX Container Daemon - l'isolation en utilisant des **namespaces** et des **CGroups** avec **liblxc**,
- **2018** - Podman - l'isolation en utilisant des **namespaces** et des **CGroups** avec **libpod**.

Conteneurs vs Machines Virtuelles

Les conteneurs et les machines virtuelles (VM) sont des technologies de virtualisation, mais elles diffèrent dans leur fonctionnement, leur structure, et leurs cas d'utilisation.

- **Architecture et Isolation**
    - **Conteneurs** - Un conteneur virtualise uniquement le système d'exploitation. Les conteneurs partagent le noyau de l'OS hôte et utilisent des namespaces et des cgroups pour l'isolation et la gestion des ressources. Ils contiennent les bibliothèques et dépendances nécessaires

pour exécuter les applications, mais ne nécessitent pas un système d'exploitation complet.
    - **Machines Virtuelles** - Une VM virtualise le matériel physique. Chaque VM exécute un système d'exploitation complet (guest OS) au-dessus d'un hyperviseur (comme VMware, KVM, ou Hyper-V) qui fonctionne sur l'OS hôte. Les VMs sont donc complètement isolées les unes des autres, car elles n'interagissent pas directement avec l'OS hôte.
- **Poids et Performance**
    - **Conteneurs** - Les conteneurs sont légers et démarrent rapidement car ils n'incluent pas d'OS complet. Ils consomment moins de ressources car ils partagent le noyau de l'OS hôte, ce qui les rend plus performants pour les déploiements rapides.
    - **Machines Virtuelles** - Les VMs sont plus lourdes et demandent plus de ressources, car chaque VM nécessite son propre OS. Elles mettent plus de temps à démarrer et consomment plus de mémoire et de CPU.
- **Cas d'utilisation**
    - **Conteneurs** - Idéaux pour les microservices, les déploiements rapides et les applications nécessitant des environnements homogènes et portables. Ils permettent de standardiser les environnements de développement, test, et production.
    - **Machines Virtuelles** - Conviennent aux applications nécessitant un niveau élevé d'isolation, aux environnements multi-OS, ou aux applications qui ont besoin de toute une pile OS. Elles sont souvent utilisées dans les environnements multi-cloud, où des applications peuvent dépendre de fonctionnalités spécifiques d'un OS particulier.
- **Sécurité**
    - **Conteneurs** - Moins isolés que les VMs, car ils partagent le noyau de l'OS hôte. Bien que l'isolation soit renforcée par des namespaces et des cgroups, une compromission du noyau pourrait affecter tous les conteneurs.
    - **Machines Virtuelles** - Offrent une meilleure isolation, chaque VM étant indépendante et exécutant son propre OS. Même si l'une est compromise, les autres VM restent protégées.

En résumé, les conteneurs sont légers, rapides et optimisés pour le déploiement d'applications isolées mais interconnectées, tandis que les machines virtuelles offrent une isolation robuste et sont adaptées aux environnements multi-OS ou pour exécuter des applications exigeant un OS complet.

Graphiquement, les différences peuvent être consultées en consultant les deux images suivantes :

Conteneurs Rootless et Rootful

Sur l'hôte du conteneur, on peut exécuter les conteneurs en tant qu'utilisateur root ou en tant qu'un utilisateur ordinaire non privilégié. Les conteneurs exécutés par un utilisateur privilégié sont appelés conteneurs **Rootful**. Les conteneurs exécutés par des utilisateurs sont appelés conteneurs **Rootless**.

Un conteneur Rootless n'est pas autorisé à utiliser les ressources du système qui sont habituellement réservées aux utilisateurs privilégiés comme

l'accès à des répertoires restreints, ou de publier des services réseau sur des ports restreints (ports inférieurs à 1024). Cette fonctionnalité empêche un éventuel attaquant d'obtenir les privilèges de root sur l'hôte du conteneur.

On peut exécuter les conteneurs directement en tant que root si nécessaire, mais ce scénario affaiblit la sécurité du système si un bogue permet à un attaquant de compromettre le conteneur.

Architecture à base de Conteneurs

Les conteneurs sont un moyen efficace de réutiliser les applications hébergées et de les rendre portables. Les conteneurs peuvent être facilement déplacés d'un environnement à un autre, par exemple du développement à la production. On peut enregistrer plusieurs versions d'un conteneur et accéder rapidement à chacune d'entre elles en cas de besoin.

Les conteneurs sont généralement temporaires ou éphémères. Vous pouvez enregistrer de manière permanente dans un stockage persistant les données générées par un conteneur en cours d'exécution mais les conteneurs s'exécutent généralement en cas de besoin, puis s'arrêtent et sont supprimés. Un nouveau processus de conteneur est lancé la prochaine fois que ce conteneur est nécessaire.

On peut installer une application logicielle complexe avec plusieurs services dans un seul conteneur. Par exemple, un serveur web peut avoir besoin d'utiliser une base de données et un système de messagerie et un système de messagerie. Cependant, l'utilisation d'un conteneur pour plusieurs services est difficile à gérer. Une meilleure conception consiste à exécuter dans des conteneurs séparés chaque composant, le serveur web, la base de données et le système de messagerie. De cette manière, les mises à jour et la maintenance des composants individuels de l'application n'affectent pas les autres composants ou la pile d'applications.

Outils de Gestion des Conteneurs

RHEL fournit un ensemble d'outils de conteneurs que qui sont utilisés pour exécuter des conteneurs dans un serveur unique :

- **podman** pour gérer les Conteneurs et les Images,
- **skopeo** pour inspecter, copier, supprimer et signer les Images,
- **buildah** pour créer des Images.

Important : L'utilisation de la commande **buildah** ne fait pas partie de la certification

RH134. Cette commande est couverte dans la formation **DO180 - Red Hat OpenShift I: Containers & Kubernetes**.

Images et Registres des Conteneurs

Pour exécuter des conteneurs, il faut utiliser une image de conteneur. Une image de conteneur est un fichier statique qui contient des étapes codifiées et qui sert de modèle pour créer des conteneurs. Les images de conteneur empaquettent une application avec toutes ses dépendances, telles que les bibliothèques système, les moteurs d'exécution et les bibliothèques du langage de programmation, ainsi que d'autres paramètres de configuration.

Les images des conteneurs sont construites conformément à des spécifications, telles que la spécification du format d'image de l'**Open Container Initiative** (OCI). Ces spécifications définissent le format des images de conteneurs, ainsi que les métadonnées relatives aux systèmes d'exploitation hôtes des conteneurs et aux architectures matérielles que l'image prend en charge.

Un registre de conteneurs est un référentiel permettant de stocker et de récupérer des images de conteneurs. Un développeur pousse ou télécharge des images de conteneurs dans un registre de conteneurs. Ensuite le développeur extrait ou télécharge des images de conteneurs d'un registre vers un système local pour exécuter des conteneurs.

Il est possible d'utiliser un registre public contenant des images de tiers ou un registre privé contrôlé par une organisation. La source des images de conteneurs est importante. Comme pour tout autre logiciel, il faut savoir si on peut faire confiance au code de l'image de conteneur. Les politiques varient d'un registre à l'autre en ce qui concerne la fourniture, l'évaluation et le test des images de conteneurs qui leur sont soumises.

Red Hat distribue des images de conteneurs certifiées par le biais de deux registres de conteneurs principaux auxquels il est possible d'accéder à l'aide des identifiants de connexion Red Hat :

- **registry.redhat.io** pour les conteneurs basés sur les produits officiels de Red Hat,
- **registry.connect.redhat.com** pour les conteneurs basés sur des produits tiers.

Le **Red Hat Container Catalog** fournit une interface web pour rechercher des contenus certifiés dans ces registres.

Podman

Présentation

Podman, créé en 2018, est un outil open-source de gestion de conteneurs développé par Red Hat. Il offre des fonctionnalités similaires à Docker mais se distingue par sa conception “daemonless”, c'est-à-dire sans besoin de daemon en arrière-plan. Cette approche améliore la sécurité et la gestion des droits : les conteneurs peuvent être exécutés en mode rootless, évitant l'exécution en tant qu'utilisateur root, ce qui limite les risques de sécurité.

Podman est basé sur OCI (Open Container Initiative) pour la compatibilité avec les formats d'images et les standards de conteneurs, et s'appuie sur runC pour l'exécution des conteneurs. Il utilise aussi des outils comme conmon (un moniteur de conteneurs léger) pour superviser les conteneurs et libpod, une bibliothèque de gestion des conteneurs permettant d'assurer l’orchestration et la gestion des conteneurs et pods. Podman supporte également la compatibilité avec les outils et API de Docker, offrant une transition plus fluide aux utilisateurs de Docker.

Podman utilise à la fois les namespaces et les cgroups, qui sont des fonctionnalités centrales du noyau Linux pour isoler et limiter les ressources des conteneurs.

- **Namespaces** - Les namespaces sont utilisés pour isoler différents aspects de l'environnement d'un conteneur, comme le système de fichiers, le réseau, les processus, les utilisateurs, et les identifiants IPC. Podman, en tant qu’outil de conteneurisation, utilise les namespaces pour créer un environnement isolé pour chaque conteneur, de sorte que les processus d'un conteneur ne puissent pas interférer avec ceux des autres.
- **Cgroups** (Control Groups) - Les cgroups sont employés pour gérer et limiter l'utilisation des ressources (CPU, mémoire, I/O, etc.) des conteneurs. Avec Podman, chaque conteneur peut être configuré pour utiliser une quantité précise de ressources système. Cela permet une meilleure allocation et empêche qu’un conteneur monopolise les ressources du système hôte.

La combinaison des namespaces et des cgroups permet à Podman de fournir une isolation forte entre les conteneurs et de contrôler la consommation des ressources, tout en restant conforme aux standards OCI pour l'exécution des conteneurs.

La Commande Podman

Podman est contenu dans le méta-paquet **container-tools**. Podman fournit plusieurs sous-commandes pour interagir avec les conteneurs et les images.La liste suivante présente les sous-commandes utilisées dans cette section :

Commande	Description
podman build	Construire une image de conteneur avec un fichier de conteneur.
podman run	Exécuter une commande dans un nouveau conteneur.

Commande	Description
podman images	Liste des images stockées localement.
podman ps	Imprimer des informations sur les conteneurs.
podman inspect	Affiche la configuration d'un conteneur, d'une image, d'un volume, d'un réseau ou d'un pod.
podman pull	Télécharger une image à partir d'un registre.
podman cp	Copier des fichiers ou des dossiers entre un conteneur et le système de fichiers local.
podman exec	Exécuter une commande dans un conteneur en cours d'exécution.
podman rm	Supprimer un ou plusieurs conteneurs.
podman rmi	Supprimer une ou plusieurs images stockées localement.
podman search	Recherche d'une image dans un registre.

LAB #1 - Configuration des Registres

La configuration par défaut des registres de conteneurs se trouve dans le fichier **/etc/containers/registries.conf** :

```
[trainee@redhat9 ~]$ cat /etc/containers/registries.conf
# For more information on this configuration file, see containers-registries.conf(5).
#
# NOTE: RISK OF USING UNQUALIFIED IMAGE NAMES
# We recommend always using fully qualified image names including the registry
# server (full dns name), namespace, image name, and tag
# (e.g., registry.redhat.io/ubi8/ubi:latest). Pulling by digest (i.e.,
# quay.io/repository/name@digest) further eliminates the ambiguity of tags.
# When using short names, there is always an inherent risk that the image being
# pulled could be spoofed. For example, a user wants to pull an image named
# `foobar` from a registry and expects it to come from myregistry.com. If
# myregistry.com is not first in the search list, an attacker could place a
# different `foobar` image at a registry earlier in the search list. The user
# would accidentally pull and run the attacker's image and code rather than the
# intended content. We recommend only adding registries which are completely
# trusted (i.e., registries which don't allow unknown or anonymous users to
# create accounts with arbitrary names). This will prevent an image from being
```

```
# spoofed, squatted or otherwise made insecure.  If it is necessary to use one
# of these registries, it should be added at the end of the list.
#
# # An array of host[:port] registries to try when pulling an unqualified image, in order.

unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io", "docker.io"]

# [[registry]]
# # The "prefix" field is used to choose the relevant [[registry]] TOML table;
# # (only) the TOML table with the longest match for the input image name
# # (taking into account namespace/repo/tag/digest separators) is used.
# #
# # The prefix can also be of the form: *.example.com for wildcard subdomain
# # matching.
# #
# # If the prefix field is missing, it defaults to be the same as the "location" field.
# prefix = "example.com/foo"
#
# # If true, unencrypted HTTP as well as TLS connections with untrusted
# # certificates are allowed.
# insecure = false
#
# # If true, pulling images with matching names is forbidden.
# blocked = false
#
# # The physical location of the "prefix"-rooted namespace.
# #
# # By default, this is equal to "prefix" (in which case "prefix" can be omitted
# # and the [[registry]] TOML table can only specify "location").
# #
# # Example: Given
# #   prefix = "example.com/foo"
# #   location = "internal-registry-for-example.net/bar"
# # requests for the image example.com/foo/myimage:latest will actually work with the
```

```
# # internal-registry-for-example.net/bar/myimage:latest image.
#
# # The location can be empty iff prefix is in a
# # wildcarded format: "*.example.com". In this case, the input reference will
# # be used as-is without any rewrite.
# location = internal-registry-for-example.com/bar"
#
# # (Possibly-partial) mirrors for the "prefix"-rooted namespace.
# #
# # The mirrors are attempted in the specified order; the first one that can be
# # contacted and contains the image will be used (and if none of the mirrors contains the image,
# # the primary location specified by the "registry.location" field, or using the unmodified
# # user-specified reference, is tried last).
# #
# # Each TOML table in the "mirror" array can contain the following fields, with the same semantics
# # as if specified in the [[registry]] TOML table directly:
# # - location
# # - insecure
# [[registry.mirror]]
# location = "example-mirror-0.local/mirror-for-foo"
# [[registry.mirror]]
# location = "example-mirror-1.local/mirrors/foo"
# insecure = true
# # Given the above, a pull of example.com/foo/image:latest will try:
# # 1. example-mirror-0.local/mirror-for-foo/image:latest
# # 2. example-mirror-1.local/mirrors/foo/image:latest
# # 3. internal-registry-for-example.net/bar/image:latest
# # in order, and use the first one that exists.
short-name-mode = "enforcing"
```

Commencez par vous connecter au registre **registry.access.redhat.com** :

```
[trainee@redhat9 ~]$ podman login registry.access.redhat.com
Username: <your_login>
```

```
Password: <your_password>
Login Succeeded!
```

Pour vérifier si vous êtes connecté au registre **registry.access.redhat.com**, ajoutez l'option **–get-login** :

```
[trainee@redhat9 ~]$ podman login registry.access.redhat.com --get-login
<your_login>
```

Notez que vous devez vous connecter à chaque registre séparément :

```
[trainee@redhat9 ~]$ podman login registry.redhat.io --get-login
Error: not logged into registry.redhat.io

[trainee@redhat9 ~]$ podman login registry.redhat.io
Username: <your_login>
Password: <your_password>
Login Succeeded!

[trainee@redhat9 ~]$ podman login registry.redhat.io --get-login
<your_login>
```

Les directives dans le fichier **/etc/containers/registries.conf** sont surchargées par les directives dans le fichier **~$HOME/.config/containers/registries.conf**. Créez donc ce fichier :

```
[trainee@redhat9 ~]$ mkdir .config/containers

[trainee@redhat9 ~]$ vi .config/containers/registries.conf

[trainee@redhat9 ~]$ cat .config/containers/registries.conf
unqualified-search-registries = ["registry.access.redhat.com", "registry.redhat.io", "docker.io"]

[[registry]]
location = "registry.access.redhat.com"
insecure = true
```

```
blocked = false
```

Pour vérifier si ce fichier est pris en compte, consultez la sortie de la commande **podman info** :

```
[trainee@redhat9 ~]$ podman info
...
registries:
  search:
  - registry.access.redhat.com
  - registry.redhat.io
  - docker.io
store:
  configFile: /home/trainee/.config/containers/storage.conf
  containerStore:
    number: 0
    paused: 0
    running: 0
    stopped: 0
...
```

LAB #2 - Gestion des Images

2.1 - Télécharger une Image

Pour obtenir une image et la stocker localement, sans créer de conteneur, utilisez la commande **podman pull** :

```
[trainee@redhat9 ~]$ podman pull registry.access.redhat.com/ubi8/python-38
Trying to pull registry.access.redhat.com/ubi8/python-38:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob 8756f22094d0 done   |
```

```
...
Copying config 142e82b6e6 done   |
Writing manifest to image destination
Storing signatures
142e82b6e600e0a2208e32bcffab89cd6257316f93b22a1f12f172756ed7fe53
```

Pour examiner l'image, il faut utiliser la commande **skopeo**, or, cette commande n'est pas installée. Devenez donc **root** et installez le paquet **skopeo** :

```
[trainee@redhat9 ~]$ su -
Password:
[root@redhat9 ~]# dnf install skopeo -y
...
[root@redhat9 ~]# exit
logout
[trainee@redhat9 ~]$
```

Utilisez maintenant la commande **skopeo inspect** pour examiner l'image téléchargée :

```
[trainee@redhat9 ~]$ skopeo inspect docker://registry.access.redhat.com/ubi8/python-38
{
    "Name": "registry.access.redhat.com/ubi8/python-38",
    "Digest": "sha256:74e5b2d063d424cb06f8e41ef1983a94b1cb890e62ec656c52e81074be21c15e",
    "RepoTags": [
        "1",
        "1-100",
        "1-100-source",
...
        "latest",
        "sha256-0020f8ea6a94fd32518a31ec7301f07a08f0109ad1854c46feb19d82d0d640d2.sig",
...
        "sha256-ff413c7793bca66c872667eadef98d77df900b5f24b288f233003e1201956c22.sig"
    ],
    "Created": "2023-08-02T19:52:55.743348399Z",
    "DockerVersion": "",
```

```
    "Labels": {
        "architecture": "x86_64",
        "build-date": "2023-08-02T19:49:35",
        "com.redhat.component": "python-38-container",
        "com.redhat.license_terms": "https://www.redhat.com/en/about/red-hat-end-user-license-agreements#UBI",
        "description": "Python 3.8 available as container is a base platform for building and running various
Python 3.8 applications and frameworks. Python is an easy to learn, powerful programming language. It has
efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's
elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting
and rapid application development in many areas on most platforms.",
        "distribution-scope": "public",
        "io.buildah.version": "1.29.0",
        "io.buildpacks.stack.id": "com.redhat.stacks.ubi8-python-38",
        "io.k8s.description": "Python 3.8 available as container is a base platform for building and running
various Python 3.8 applications and frameworks. Python is an easy to learn, powerful programming language. It has
efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's
elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting
and rapid application development in many areas on most platforms.",
        "io.k8s.display-name": "Python 3.8",
        "io.openshift.expose-services": "8080:http",
        "io.openshift.s2i.scripts-url": "image:///usr/libexec/s2i",
        "io.openshift.tags": "builder,python,python38,python-38,rh-python38",
        "io.s2i.scripts-url": "image:///usr/libexec/s2i",
        "maintainer": "SoftwareCollections.org \u003csclorg@redhat.com\u003e",
        "name": "ubi8/python-38",
        "release": "131",
        "summary": "Platform for building and running Python 3.8 applications",
        "url": "https://access.redhat.com/containers/#/registry.access.redhat.com/ubi8/python-38/images/1-131",
        "usage": "s2i build https://github.com/sclorg/s2i-python-container.git --context-dir=3.8/test/setup-test-
app/ ubi8/python-38 python-sample-app",
        "vcs-ref": "92c79cfbeb4465ee73f816c7c6069b7402e4ec19",
        "vcs-type": "git",
        "vendor": "Red Hat, Inc.",
        "version": "1"
```

```
    },
    "Architecture": "amd64",
    "Os": "linux",
    "Layers": [
        "sha256:bea2a0b08f4fd7df72285c8ccf71ff0e9b76c025a0bc4dc67a4f40695feb0eca",
        "sha256:7822e944d15c45e998e88e0638073a1974246aea8fd268a925948eb2e070e048",
        "sha256:b82ddf37e40febb44c258077df217aef2b72f65c2c190ecd3a165ae894256e11",
        "sha256:8756f22094d074e5ea7b13b5a7cb8c5132b61a8b39d550f58e6a6053e4b3530d"
    ],
    "LayersData": [
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest": "sha256:bea2a0b08f4fd7df72285c8ccf71ff0e9b76c025a0bc4dc67a4f40695feb0eca",
            "Size": 79272789,
            "Annotations": null
        },
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest": "sha256:7822e944d15c45e998e88e0638073a1974246aea8fd268a925948eb2e070e048",
            "Size": 18418966,
            "Annotations": null
        },
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest": "sha256:b82ddf37e40febb44c258077df217aef2b72f65c2c190ecd3a165ae894256e11",
            "Size": 151252194,
            "Annotations": null
        },
        {
            "MIMEType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
            "Digest": "sha256:8756f22094d074e5ea7b13b5a7cb8c5132b61a8b39d550f58e6a6053e4b3530d",
            "Size": 78908672,
            "Annotations": null
        }
```

```
    ],
    "Env": [
        "container=oci",
        "STI_SCRIPTS_URL=image:///usr/libexec/s2i",
        "STI_SCRIPTS_PATH=/usr/libexec/s2i",
        "APP_ROOT=/opt/app-root",
        "HOME=/opt/app-root/src",
        "PLATFORM=el8",
        "NODEJS_VER=14",
        "PYTHON_VERSION=3.8",
        "PATH=/opt/app-root/src/.local/bin/:/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "PYTHONUNBUFFERED=1",
        "PYTHONIOENCODING=UTF-8",
        "LC_ALL=en_US.UTF-8",
        "LANG=en_US.UTF-8",
        "CNB_STACK_ID=com.redhat.stacks.ubi8-python-38",
        "CNB_USER_ID=1001",
        "CNB_GROUP_ID=0",
        "PIP_NO_CACHE_DIR=off",
        "SUMMARY=Platform for building and running Python 3.8 applications",
        "DESCRIPTION=Python 3.8 available as container is a base platform for building and running various Python
3.8 applications and frameworks. Python is an easy to learn, powerful programming language. It has efficient
high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant
syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and
rapid application development in many areas on most platforms.",
        "BASH_ENV=/opt/app-root/bin/activate",
        "ENV=/opt/app-root/bin/activate",
        "PROMPT_COMMAND=. /opt/app-root/bin/activate"
    ]
}
```

Pour lister les images disponibles localement, utilisez la commande **podman images** :

```
[trainee@redhat9 ~]$ podman images
REPOSITORY                                  TAG         IMAGE ID      CREATED        SIZE
registry.access.redhat.com/ubi8/python-38   latest      142e82b6e600  15 months ago  898 MB
```

Important - Notez que l'image est référencée par son IMAGE ID.

2.2 - Créer une Image à l'aide d'un Fichier Containerfile

Créez un contexte sous la forme d'un répertoire, placez-vous dans ce répertoire puis créez le fichier Containerfile :

```
[trainee@redhat9 ~]$ mkdir rh13409

[trainee@redhat9 ~]$ cd rh13409/

[trainee@redhat9 rh13409]$ vi Containerfile

[trainee@redhat9 rh13409]$ cat Containerfile
FROM registry.access.redhat.com/ubi8/ubi:latest
RUN dnf install -y python36
CMD ["/bin/bash", "-c", "sleep infinity"]
```

Dans le fichier **Containerfile**, les significations des commandes sont :

Commande	Description
FROM	Définit l'image à partir de laquelle sera construite la nouvelle image.
RUN	Lance un processus dans la construction de la nouvelle image.
CMD	Définit la commande qui sera exécutée dans le conteneur lors de sa création à partir de la nouvelle image.

Créez maintenant l'image **python36:1.0**, l'option **-t** indique un **tag** :

```
[trainee@redhat9 rh13409]$ podman build -t python36:1.0 .
STEP 1/3: FROM registry.access.redhat.com/ubi8/ubi:latest
Trying to pull registry.access.redhat.com/ubi8/ubi:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob 148a3ed2f70e done   |
Copying config 4f03f39cd4 done   |
Writing manifest to image destination
Storing signatures
STEP 2/3: RUN dnf install -y python36
Updating Subscription Management repositories.
Unable to read consumer identity
subscription-manager is operating in container mode.

This system is not registered with an entitlement server. You can use subscription-manager to register.

Red Hat Enterprise Linux 8 for x86_64 - AppStre  31 MB/s |  68 MB     00:02
Red Hat Enterprise Linux 8 for x86_64 - BaseOS   34 MB/s |  74 MB     00:02
Red Hat Universal Base Image 8 (RPMs) - BaseOS  2.9 MB/s | 722 kB     00:00
Red Hat Universal Base Image 8 (RPMs) - AppStre  11 MB/s | 3.2 MB     00:00
Red Hat Universal Base Image 8 (RPMs) - CodeRea 991 kB/s | 186 kB     00:00
Last metadata expiration check: 0:00:01 ago on Wed Oct 30 16:31:36 2024.
Dependencies resolved.
================================================================================
 Package               Arch    Version                                 Repository                          Size
================================================================================
Installing:
 python36              x86_64  3.6.8-39.module+el8.10.0+20784+edafcd43 rhel-8-for-x86_64-appstream-rpms   20 k
Installing dependencies:
 platform-python-pip   noarch  9.0.3-24.el8                            rhel-8-for-x86_64-baseos-rpms     1.6 M
 python3-pip           noarch  9.0.3-24.el8                            rhel-8-for-x86_64-appstream-rpms   20 k
 python3-setuptools    noarch  39.2.0-8.el8_10                         rhel-8-for-x86_64-baseos-rpms     163 k
Enabling module streams:
 python36                      3.6
```

```
Transaction Summary
================================================================================================================================
Install  4 Packages

Total download size: 1.8 M
Installed size: 7.1 M
Downloading Packages:
(1/4): python36-3.6.8-39.module+el8.10.0+20784+ 137 kB/s |  20 kB     00:00
(2/4): python3-setuptools-39.2.0-8.el8_10.noarc 2.9 MB/s | 163 kB     00:00
(3/4): platform-python-pip-9.0.3-24.el8.noarch. 7.2 MB/s | 1.6 MB     00:00
(4/4): python3-pip-9.0.3-24.el8.noarch.rpm       92 kB/s |  20 kB     00:00
--------------------------------------------------------------------------------
Total                                           7.8 MB/s | 1.8 MB     00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        :                                                        1/1
  Installing       : python3-setuptools-39.2.0-8.el8_10.noarch              1/4
  Installing       : platform-python-pip-9.0.3-24.el8.noarch                2/4
  Installing       : python3-pip-9.0.3-24.el8.noarch                        3/4
  Installing       : python36-3.6.8-39.module+el8.10.0+20784+edafcd43.x86   4/4
  Running scriptlet: python36-3.6.8-39.module+el8.10.0+20784+edafcd43.x86   4/4
  Verifying        : python36-3.6.8-39.module+el8.10.0+20784+edafcd43.x86   1/4
  Verifying        : python3-pip-9.0.3-24.el8.noarch                        2/4
  Verifying        : platform-python-pip-9.0.3-24.el8.noarch                3/4
  Verifying        : python3-setuptools-39.2.0-8.el8_10.noarch              4/4
Installed products updated.

Installed:
  platform-python-pip-9.0.3-24.el8.noarch
  python3-pip-9.0.3-24.el8.noarch
  python3-setuptools-39.2.0-8.el8_10.noarch
```

```
  python36-3.6.8-39.module+el8.10.0+20784+edafcd43.x86_64

Complete!
--> ffbfe7e2c52a
STEP 3/3: CMD ["/bin/bash", "-c", "sleep infinity"]
COMMIT python36:1.0
--> aeb6174afefe
Successfully tagged localhost/python36:1.0
aeb6174afefe34e16037a22efe8d6b9de6f7542dd15e24f33335fd5ba4689dd7
```

```
[trainee@redhat9 rh13409]$ podman images
REPOSITORY                                TAG         IMAGE ID      CREATED        SIZE
localhost/python36                        1.0         aeb6174afefe  9 minutes ago  529 MB
registry.access.redhat.com/ubi8/ubi       latest      4f03f39cd427  6 weeks ago    212 MB
registry.access.redhat.com/ubi8/python-38 latest      142e82b6e600  15 months ago  898 MB
```

En consultant l'image construite, on peut constater les résultat des commandes incluses dans le fichier **Containerfile**. Notez bien que l'on doit référencer l'image par son nom complet **registre/tag** :

```
[trainee@redhat9 ~]$ podman inspect localhost/python36:1.0
...
               "Cmd": [
                    "/bin/bash",
                    "-c",
                    "sleep infinity"
               ],
...
               {
                    "created": "2024-10-30T16:32:22.752492995Z",
                    "created_by": "/bin/sh -c dnf install -y python36",
                    "comment": "FROM registry.access.redhat.com/ubi8/ubi:latest"
               },
...
```

LAB #3 - Gestion des Conteneurs

Un conteneur peut être dans un de cinq états :

Etat	Description
Created	Un conteneur qui est créé mais qui n'est pas démarré.
Running	Un conteneur qui fonctionne avec ses processus.
Stopped	Un conteneur dont les processus sont arrêtés.
Paused	Un conteneur dont les processus sont en pause. Non pris en charge pour les conteneurs Rootless.
Deleted	Un conteneur dont les processus sont dans un état mort.

3.1 - Création d'un Conteneur

Créez un conteneur dénommé **python36** à partir de la nouvelle image. Notez que l'image est référencée par son **IMAGE ID** :

```
[trainee@redhat9 ~]$ podman create --name python36 aeb6174afefe
f42e19a0627eb457570ca3626c8bb1fff77963542b7dc59ae5d07e86bf1a3fca
```

Pour visualiser la liste des conteneurs en cours d'exécution, utilisez la commande **podman ps** :

```
[trainee@redhat9 ~]$ podman ps
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

Pour consulter la liste de tous les conteneurs, ajoutez l'option **-a** à la commande précédente :

```
[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                   COMMAND               CREATED         STATUS      PORTS       NAMES
f42e19a0627e  localhost/python36:1.0  /bin/bash -c slee...  32 seconds ago  Created                 python36
```

3.2 - Démarrage d'un Conteneur

Pour démarrer un conteneur déjà créé, utilisez la commande **podman start** :

```
[trainee@redhat9 ~]$ podman start python36
python36

[trainee@redhat9 ~]$ podman ps
CONTAINER ID  IMAGE                     COMMAND               CREATED             STATUS        PORTS       NAMES
f42e19a0627e  localhost/python36:1.0  /bin/bash -c slee...  About a minute ago  Up 5 seconds
python36
```

Pour démarrer un conteneur à partir d'une image distante, il convient d'utiliser la commande **podman run** :

```
[trainee@redhat9 ~]$ podman run -d --name python38 registry.access.redhat.com/ubi8/python-38 sleep infinity
85e26c02bfad3b47270b785b74ce225799dea5aff16ebf4f002a51688da2b3a7

[trainee@redhat9 ~]$ podman ps
CONTAINER ID  IMAGE                                           COMMAND               CREATED         STATUS
PORTS       NAMES
f42e19a0627e  localhost/python36:1.0                          /bin/bash -c slee...  4 minutes ago   Up 3
minutes               python36
85e26c02bfad  registry.access.redhat.com/ubi8/python-38:latest  sleep infinity        38 seconds ago  Up 38
seconds             python38
```

3.3 - Exécution d'une Commande dans un Conteneur

Pour exécuter une commande à l'intérieur d'un conteneur en cours d'exécution, utilisez la commande **podman exec** :

```
[trainee@redhat9 ~]$ podman exec python38 ps -ax
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep infinity
```

```
      2 ?        R      0:00 ps -ax
```

Il est aussi possible d'utilser cette commande en stipulant un shell spécifique :

```
[trainee@redhat9 ~]$ podman exec python38 sh -c 'ps -ax > /tmp/process-data.log'

[trainee@redhat9 ~]$ podman exec python38 cat /tmp/process-data.log
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:00 /usr/bin/coreutils --coreutils-prog-shebang=sleep /usr/bin/sleep infinity
      3 ?        S      0:00 sh -c ps -ax > /tmp/process-data.log
      4 ?        R      0:00 ps -ax
```

Créez maintenant le fichier **/tmp/hello.sh** contenant la chaîne **hello world** et regardez ses caractéristiques :

```
[trainee@redhat9 ~]$ echo "echo 'hello world'" > /tmp/hello.sh

[trainee@redhat9 ~]$ stat /tmp/hello.sh
  File: /tmp/hello.sh
  Size: 19              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 33670667    Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/ trainee)   Gid: ( 1000/ trainee)
Context: unconfined_u:object_r:user_tmp_t:s0
Access: 2024-10-31 09:49:46.727000000 +0100
Modify: 2024-10-31 09:49:46.727000000 +0100
Change: 2024-10-31 09:49:46.727000000 +0100
 Birth: 2024-10-31 09:49:46.727000000 +0100

Bien évidement ce fichier n'existe pas encore dans le conteneur **python38** :

<code>
[trainee@redhat9 ~]$ podman exec python38 stat /tmp/hello.sh
stat: cannot statx '/tmp/hello.sh': No such file or directory
```

Copiez donc le fichier dans le conteneur et regardez ses caractéristiques :

```
[trainee@redhat9 ~]$ podman cp /tmp/hello.sh python38:/tmp/hello.sh

[trainee@redhat9 ~]$ podman exec python38 stat /tmp/hello.sh
  File: /tmp/hello.sh
  Size: 19              Blocks: 8          IO Block: 4096   regular file
Device: 40h/64d Inode: 115769725   Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/ default)   Gid: (    0/    root)
Access: 2024-10-31 08:49:47.000000000 +0000
Modify: 2024-10-31 08:49:47.000000000 +0000
Change: 2024-10-31 08:52:14.862000000 +0000
 Birth: 2024-10-31 08:52:14.862000000 +0000
```

Exécutez le script dans le conteneur avec le shell **bash** :

```
[trainee@redhat9 ~]$ podman exec python38 bash /tmp/hello.sh
hello world
```

3.4 - Suppression des Images et des Conteneurs

En essayant de supprimer l'image **registry.access.redhat.com/ubi8/python-38**, vous constaterez que ceci n'est pas possible à cause de la présence du conteneur **python38** qui dépend de l'image :

```
[trainee@redhat9 ~]$ podman rmi registry.access.redhat.com/ubi8/python-38
Error: image used by 85e26c02bfad3b47270b785b74ce225799dea5aff16ebf4f002a51688da2b3a7: image is in use by a
container: consider listing external containers and force-removing image
```

Arrêtez donc le conteneur **python38** :

```
[trainee@redhat9 ~]$ podman stop python38
WARN[0010] StopSignal SIGTERM failed to stop container python38 in 10 seconds, resorting to SIGKILL
python38
```

```
[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                           COMMAND               CREATED         STATUS
PORTS       NAMES
f42e19a0627e  localhost/python36:1.0                          /bin/bash -c slee...  18 minutes ago  Up 17
minutes                            python36
85e26c02bfad  registry.access.redhat.com/ubi8/python-38:latest  sleep infinity        14 minutes ago  Exited
(137) 17 seconds ago               python38
```

Supprimez le conteneur **python38** avec la commande **podman rm** :

```
[trainee@redhat9 ~]$ podman rm python38
python38
```

Il est maintenant possible de supprimer l'image **registry.access.redhat.com/ubi8/python-38** :

```
[trainee@redhat9 ~]$ podman rmi registry.access.redhat.com/ubi8/python-38
Untagged: registry.access.redhat.com/ubi8/python-38:latest
Deleted: 142e82b6e600e0a2208e32bcffab89cd6257316f93b22a1f12f172756ed7fe53
```

Créez un conteneur **db01** à partir de l'image **registry.redhat.io/rhel8/mariadb-105** :

```
[trainee@redhat9 ~]$ podman run -d registry.redhat.io/rhel8/mariadb-105 --name db01
Trying to pull registry.redhat.io/rhel8/mariadb-105:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob 0258a9a5cd06 done   |
Copying blob 148a3ed2f70e skipped: already exists
Copying blob 4de1bba6ee61 done   |
Copying config b0e9bcdcc3 done   |
Writing manifest to image destination
Storing signatures
aea7c40abc0570fb616be14de5642683be5ebbadc95359f4b2d3ec32f1b12ddd

[trainee@redhat9 ~]$ podman ps -a
```

```
CONTAINER ID  IMAGE                                           COMMAND               CREATED         STATUS
PORTS       NAMES
f42e19a0627e  localhost/python36:1.0                          /bin/bash -c slee...  40 minutes ago  Up 38 minutes
python36
aea7c40abc05  registry.redhat.io/rhel8/mariadb-105:latest  --name db01           45 seconds ago  Exited (2) 45
seconds ago              great_nash
```

Examinez l'image avec la commande **skopeo** :

```
[trainee@redhat9 ~]$ skopeo inspect docker://registry.redhat.io/rhel8/mariadb-105
{
    "Name": "registry.redhat.io/rhel8/mariadb-105",
    "Digest": "sha256:eed1478afb45d095a6d667dc8903d9a656c894397805d1a3faf8b3301a4bec23",
...
        "summary": "MariaDB 10.5 SQL database server",
        "url":
"https://access.redhat.com/containers/#/registry.access.redhat.com/rhel8/mariadb-105/images/1-204",
        "usage": "podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -p 3306:3306
rhel8/mariadb-105",
        "vcs-ref": "6d784bf12dd71b38121fd97070c13447470e237e",
        "vcs-type": "git",
        "vendor": "Red Hat, Inc.",
        "version": "1"
...
}
```

Vous noterez que pour lancer un conteneur, il faut spécifier les variables d'environnement indiquées dans la sortie de la commande **skopeo** :

```
[trainee@redhat9 ~]$ podman run -d -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -p 3306:3306
rhel8/mariadb-105
afc0cc8427677eed8cf17ef1ee9669b937ca25ccd322c066ee9c46e070baf3d3

[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                           COMMAND               CREATED         STATUS
```

```
PORTS                    NAMES
f42e19a0627e  localhost/python36:1.0                        /bin/bash -c slee...  46 minutes ago  Up 44 minutes
python36
aea7c40abc05  registry.redhat.io/rhel8/mariadb-105:latest  --name db01           6 minutes ago   Exited (2) 6
minutes ago                           great_nash
afc0cc842767  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            12 seconds ago  Up 11 seconds
0.0.0.0:3306->3306/tcp  boring_ganguly
```

Notez que le conteneur crée avec les variables d'environnement n'a pas été créé en spécifiant l'option **–name**. De ce fait, le conteneur a été automatiquement créé avec un nom généré aléatoirement.

Supprimez maintenant le conteneur **db01**. Notez que vous que l'on peut référencer le conteneur par une partie de son **CONTAINER ID**, à condition que cette partie soit unique :

```
[trainee@redhat9 ~]$ podman rm aea
aea
```

Supprimez le deuxième conteneur en stipulant son nom :

```
[trainee@redhat9 ~]$ podman rm boring_ganguly
Error: cannot remove container afc0cc8427677eed8cf17ef1ee9669b937ca25ccd322c066ee9c46e070baf3d3 as it is running
- running or paused containers cannot be removed without force: container state improper

[trainee@redhat9 ~]$ podman rm -f boring_ganguly
boring_ganguly
```

Re-créez un conteneur MariaDB ayant le nom **db1** :

```
[trainee@redhat9 ~]$ podman run -d --name db1 -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -p
3306:3306 rhel8/mariadb-105
90c582b10512c3049ded2937d72c6c4ccfe98a6e40311f44000a7c0425aa219e

[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                       COMMAND               CREATED         STATUS
```

```
PORTS                    NAMES
f42e19a0627e  localhost/python36:1.0                        /bin/bash -c slee...  49 minutes ago  Up 48 minutes
python36
90c582b10512  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            8 seconds ago   Up 7 seconds
0.0.0.0:3306->3306/tcp  db1
```

LAB #4 - Gestion du Stockage et du Réseau

4.1 - Gestion du Stockage Persistant

Consultez les deux fichiers **/proc/self/uid_map** et **cat /proc/self/gid_map** :

```
[trainee@redhat9 ~]$ podman unshare cat /proc/self/uid_map
         0       1000          1
         1     100000      65536

[trainee@redhat9 ~]$ podman unshare cat /proc/self/gid_map
         0       1000          1
         1     100000      65536
```

> **Important** - La sortie précédente montre que dans le conteneur, l'utilisateur root (UID et GID de 0) correspond à votre utilisateur (UID et GID de 1000) sur la machine hôte. Dans le conteneur, l'UID et le GID de 1 correspondent à l'UID et au GID de 100000 sur la machine hôte. Chaque UID et GID après 1 s'incrémente de 1. Par exemple, l'UID et le GID de 30 dans un conteneur correspondent à l'UID et au GID de 100029 sur la machine hôte.

Obtenez l'UID et le GID de l'utilisateur **mysql** dans le conteneur :

```
[trainee@redhat9 ~]$ podman exec -it db1 grep mysql /etc/passwd
```

```
mysql:x:27:27:MySQL Server:/var/lib/mysql:/sbin/nologin
```

Il convient maintenant de monter le répertoire **/home/trainee/db_data** sur le répertoire **/var/lib/mysql** dans le conteneur db1 pour fournir un stockage persistant.

Créez donc le répertoire **/home/trainee/db_data** et utilisez la commande **podman unshare** pour définir l'UID et le GID de 27 en tant que propriétaire du répertoire :

```
[trainee@redhat9 ~]$ mkdir /home/trainee/db_data

[trainee@redhat9 ~]$ podman unshare chown 27:27 /home/trainee/db_data

[trainee@redhat9 ~]$ ls -l /home/trainee
total 8
drwxr-xr-x. 2 trainee trainee  22 Oct 21 12:01 bin
drwxr-xr-x. 2  100026  100026   6 Oct 31 10:37 db_data
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Desktop
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Documents
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Downloads
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Music
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Pictures
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Public
drwxr-xr-x. 2 trainee trainee  27 Oct 30 17:29 rh13409
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Templates
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Videos
```

Vérifiez que l'état de SELinux est **enforcing** dans la machine hôte **redhat9** :

```
[trainee@redhat9 ~]$ su -
Password: fenestros

[root@redhat9 ~]# getenforce
Permissive
```

```
[root@redhat9 ~]# setenforce enforcing

[root@redhat9 ~]# exit
logout
```

Arrêtez le conteneur **bd1** :

```
[trainee@redhat9 ~]$ podman stop db1
db1
```

Créez un conteneur dénommé **bd01** :

```
[trainee@redhat9 ~]$ podman run -d --name db01 -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e MYSQL_DATABASE=db -p
3306:3306 -v /home/trainee/db_data:/var/lib/mysql rhel8/mariadb-105
959e8039ba2bc11a30bbe0ffb9185c4779aa6c8865a9288767675c5163e119bd
```

Consultez l'état du conteneur. Le conteneur **db01** sera en état d'arrêt :

```
[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                       COMMAND               CREATED             STATUS
PORTS                   NAMES
f42e19a0627e  localhost/python36:1.0                      /bin/bash -c slee...  About an hour ago   Up About an
hour                                  python36
90c582b10512  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            27 minutes ago      Exited (0) 7
minutes ago       0.0.0.0:3306->3306/tcp  db1
959e8039ba2b  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            About a minute ago  Exited (1)
About a minute ago  0.0.0.0:3306->3306/tcp  db01
```

Afin de comprendre la nature du problème, consultez les journaux du conteneur avec la commande **podman container logs** :

```
[trainee@redhat9 ~]$ podman container logs db01
...
2024-10-31  9:53:10 0 [ERROR] mysqld: Got error 'Can't open file' when trying to use aria control file
'/var/lib/mysql/data/aria_log_control'
```

```
2024-10-31  9:53:10 0 [ERROR] Plugin 'Aria' registration as a STORAGE ENGINE failed.
2024-10-31  9:53:10 0 [ERROR] InnoDB: Operating system error number 13 in a file operation.
2024-10-31  9:53:10 0 [ERROR] InnoDB: The error means mysqld does not have the access rights to the directory.
...
```

En notant l'erreur **The error means mysqld does not have the access rights to the directory.**, devenez root dans la machine hôte **redhat9** :

```
[trainee@redhat9 ~]$ su -
Password: fenestros
```

Générez un fichier d'alertes de SELinux :

```
[root@redhat9 ~]# sealert -a /var/log/audit/audit.log > /root/mylogfile.txt

[root@redhat9 ~]# cat /root/mylogfile.txt
...
--------------------------------------------------------------------------------

SELinux is preventing /usr/libexec/mariadbd from 'read, write' accesses on the file aria_log_control.

*****  Plugin catchall (100. confidence) suggests   **************************

If you believe that mariadbd should be allowed read write access on the aria_log_control file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# ausearch -c 'mysqld' --raw | audit2allow -M my-mysqld
# semodule -X 300 -i my-mysqld.pp


Additional Information:
Source Context                system_u:system_r:container_t:s0:c602,c794
Target Context                system_u:object_r:user_home_t:s0
```

```
Target Objects                aria_log_control [ file ]
Source                        mysqld
Source Path                   /usr/libexec/mariadbd
Port                          <Unknown>
Host                          <Unknown>
Source RPM Packages
Target RPM Packages
SELinux Policy RPM            selinux-policy-targeted-38.1.35-2.el9_4.2.noarch
Local Policy RPM              selinux-policy-targeted-38.1.35-2.el9_4.2.noarch
Selinux Enabled               True
Policy Type                   targeted
Enforcing Mode                Enforcing
Host Name                     redhat9.ittraining.loc
Platform                      Linux redhat9.ittraining.loc
                              5.14.0-427.37.1.el9_4.x86_64 #1 SMP
                              PREEMPT_DYNAMIC Fri Sep 13 12:41:50 EDT 2024
                              x86_64 x86_64
Alert Count                   1
First Seen                    2024-10-31 10:53:10 CET
Last Seen                     2024-10-31 10:53:10 CET
Local ID                      c9066e3a-05ae-464e-bbd4-fd94585c6b64

Raw Audit Messages
type=AVC msg=audit(1730368390.63:60521): avc:  denied  { read write } for  pid=228495 comm="mysqld"
name="aria_log_control" dev="dm-0" ino=373133 scontext=system_u:system_r:container_t:s0:c602,c794 tcontext=sys
tem_u:object_r:user_home_t:s0 tclass=file permissive=0


type=SYSCALL msg=audit(1730368390.63:60521): arch=x86_64 syscall=openat success=no exit=EACCES a0=ffffff9c
a1=7ffd32756e90 a2=80002 a3=0 items=0 ppid=228473 pid=228495 auid=1000 uid=100026 gid=100026 euid=10002
6 suid=100026 fsuid=100026 egid=100026 sgid=100026 fsgid=100026 tty=(none) ses=6974 comm=mysqld
exe=/usr/libexec/mariadbd subj=system_u:system_r:container_t:s0:c602,c794 key=(null)ARCH=x86_64 SYSCALL=openat
AUI
D=trainee UID=unknown(100026) GID=unknown(100026) EUID=unknown(100026) SUID=unknown(100026) FSUID=unknown(100026)
```

```
EGID=unknown(100026) SGID=unknown(100026) FSGID=unknown(100026)

Hash: mysqld,container_t,user_home_t,file,read,write

[root@redhat9 ~]# exit
logout
```

En consultant l'erreur ci-dessus, il semblerait que pour accéder au répertoire **/home/trainee/db_data**, le SC du répertoire diot conteneur le type **container_file_t**. Consultez le SC actuel du répertoire :

```
[root@redhat9 ~]# ls -lZ /home/trainee | grep db_data
drwxr-xr-x. 3  100026  100026 unconfined_u:object_r:user_home_t:s0   36 Oct 31 10:48 db_data
```

Pour forcer l'utilisation du SC correct, il faut ajouter la lettre **Z** à la fin de la valeur de l'option **-v** de la commande **podman run**. Par exemple, **-v chemin_hôte:chemin_conteneur:Z**. Relancez donc la commande en spécifiant Z **et** l'option **–replace** qui va remplacer le conteneur actuel :

```
[trainee@redhat9 ~]$ podman run -d --replace --name db01 -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e
MYSQL_DATABASE=db -p 3306:3306 -v /home/trainee/db_data:/var/lib/mysql:Z rhel8/mariadb-105
47a5626d51dd02a8a7c769bfd37c4ba763320f731a2ccc71797567817205a20b

[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                          COMMAND               CREATED             STATUS
PORTS                   NAMES
f42e19a0627e  localhost/python36:1.0                         /bin/bash -c slee...  2 hours ago         Up 2 hours
python36
90c582b10512  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            51 minutes ago      Exited (0)
31 minutes ago  0.0.0.0:3306->3306/tcp  db1
47a5626d51dd  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            About a minute ago  Up About a
minute          0.0.0.0:3306->3306/tcp  db01
```

Finalement vérifiez le SC du répertoire **/home/trainee/db_data** :

```
[trainee@redhat9 ~]$ ls -lZ /home/trainee | grep db_data
```

```
drwxr-xr-x. 3  100026  100026 system_u:object_r:container_file_t:s0:c274,c277  36 Oct 31 11:16 db_data
```

4.2 - Gestion du Réseau

Pour fournir un accès réseau aux conteneurs, les clients doivent se connecter aux ports de l'hôte du conteneur qui transmettent le trafic réseau aux ports du conteneur. Par exemple, il convient de mapper le port 13306 de l'hôte du conteneur au port 3306 du conteneur pour communiquer avec le conteneur MariaDB.

```
[trainee@redhat9 ~]$ podman run -d --replace --name db01 -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e
MYSQL_DATABASE=db -p 3306:3306 -v /home/trainee/db_data:/var/lib/mysql:Z -p 13306:3306 rhel8/mariadb-105
97f190a660ceba8f62b688781a5df1daef06e3ded241d799b15a8df744049116
```

Notez ensuite la prise en charge de cette commande :

```
[trainee@redhat9 ~]$ podman ps -a | grep db01
97f190a660ce  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld           55 seconds ago  Up 56 seconds
0.0.0.0:3306->3306/tcp, 0.0.0.0:13306->3306/tcp  db01
```

Pour voir le mappage des ports, utilisez la commande **podman port -a** :

```
[trainee@redhat9 ~]$ podman port -a
97f190a660ce    3306/tcp -> 0.0.0.0:3306
97f190a660ce    3306/tcp -> 0.0.0.0:13306

[trainee@redhat9 ~]$ podman port db01
3306/tcp -> 0.0.0.0:3306
3306/tcp -> 0.0.0.0:13306
```

Depuis la version 4.0, Podman supporte deux réseaux pour les conteneurs, **Netavark** et **CNI**. À partir de RHEL 9, les systèmes utilisent **Netavark** par défaut. Pour vérifier quel backend réseau est utilisé, exécutez la commande podman info suivante :

```
[trainee@redhat9 ~]$ podman --version
```

```
podman version 4.9.4-rhel

[trainee@redhat9 ~]$ podman info --format {{.Host.NetworkBackend}}
netavark
```

Pour modifier le backend utilisé, il convient de modifier la directive **network_backend** dans le fichier **/usr/share/containers/containers.conf** :

```
[trainee@redhat9 ~]$ cat /usr/share/containers/containers.conf | grep network_backend
#network_backend = ""
```

Lors de l'installation de podman un réseau par défaut, utilisant le DRIVER **bridge** et appelé **podman**, est créé :

```
[trainee@redhat9 ~]$ podman network ls
NETWORK ID    NAME        DRIVER
2f259bab93aa  podman      bridge
```

> **Important** - Le DRIVER **bridge** est limité aux conteneurs d'un hôte unique exécutant podman. Les conteneurs ne peuvent communiquer qu'entre eux et ils ne sont pas accessibles depuis l'extérieur. Pour que les conteneurs sur le réseau puissent communiquer ou être accessibles du monde extérieur, il faut configurer le mappage de port.

Regardez les caractéristiques de ce réseau :

```
[trainee@redhat9 ~]$ podman network inspect podman
[
     {
          "name": "podman",
          "id": "2f259bab93aaaaa2542ba43ef33eb990d0999ee1b9924b557b7be53c0b7a1bb9",
          "driver": "bridge",
          "network_interface": "podman0",
```

```
          "created": "2024-10-31T11:37:47.368883073+01:00",
          "subnets": [
               {
                    "subnet": "10.88.0.0/16",
                    "gateway": "10.88.0.1"
               }
          ],
          "ipv6_enabled": false,
          "internal": false,
          "dns_enabled": false,
          "ipam_options": {
               "driver": "host-local"
          }
     }
]
```

Créez maintenant un deuxième réseau appelé **db_net** :

```
[trainee@redhat9 ~]$ podman network create --gateway 10.87.0.1 --subnet 10.87.0.0/16 db_net
db_net

[trainee@redhat9 ~]$ podman network ls
NETWORK ID    NAME        DRIVER
556c9797ea6d  db_net      bridge
2f259bab93aa  podman      bridge

[trainee@redhat9 ~]$ podman network inspect db_net
[
     {
          "name": "db_net",
          "id": "556c9797ea6df5a823901755db1e9526c7de975a0881def515e61f75d7db38bc",
          "driver": "bridge",
          "network_interface": "podman1",
          "created": "2024-10-31T11:41:38.479538212+01:00",
```

```
          "subnets": [
               {
                    "subnet": "10.87.0.0/16",
                    "gateway": "10.87.0.1"
               }
          ],
          "ipv6_enabled": false,
          "internal": false,
          "dns_enabled": true,
          "ipam_options": {
               "driver": "host-local"
          }
     }
]
```

Placez le conteneur **db01** dans le réseau **db_net** :

```
[trainee@redhat9 ~]$ podman run -d --replace --name db01 -e MYSQL_USER=user -e MYSQL_PASSWORD=pass -e
MYSQL_DATABASE=db -p 3306:3306 -v /home/trainee/db_data:/var/lib/mysql:Z -p 13306:3306 --network db_net
rhel8/mariadb-105
36c9f95ae52f76f96c45bb7fd6600cb014adc751059a531bb7bf38f927a11541
```

Créez maintenant le conteneur **client** en le plaçant dans le réseau **db_net** :

```
[trainee@redhat9 ~]$ podman run -d --name client --network db_net registry.access.redhat.com/ubi8/ubi:latest
sleep infinity
00b2b82c2adf022badf91f88a9a7f59e71cac705bf4885fa08553681ad1f594c

[trainee@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                           COMMAND               CREATED            STATUS
PORTS                                       NAMES
f42e19a0627e  localhost/python36:1.0                          /bin/bash -c slee...  2 hours ago        Up 2 hours
python36
90c582b10512  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld            About an hour ago  Exited (0)
```

```
About an hour ago  0.0.0.0:3306->3306/tcp                              db1
36c9f95ae52f  registry.redhat.io/rhel8/mariadb-105:latest  run-mysqld           7 minutes ago      Up 7 minutes
0.0.0.0:3306->3306/tcp, 0.0.0.0:13306->3306/tcp  db01
00b2b82c2adf  registry.access.redhat.com/ubi8/ubi:latest   sleep infinity       13 seconds ago     Up 13 seconds
client
```

Installez dans le conteneur **client**, les paquets **iputils** et **iproute** :

```
[trainee@redhat9 ~]$ podman exec -it client dnf install -y iputils iproute
```

Testez la connectivité du réseau **db_net** :

```
[trainee@redhat9 ~]$ podman exec -it client ip a | grep 10.8
    inet 10.87.0.3/16 brd 10.87.255.255 scope global eth0

[trainee@redhat9 ~]$ podman exec -it client ping -c3 db01
PING db01.dns.podman (10.87.0.2) 56(84) bytes of data.
64 bytes from 10.87.0.2 (10.87.0.2): icmp_seq=1 ttl=64 time=0.048 ms
64 bytes from 10.87.0.2 (10.87.0.2): icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.87.0.2 (10.87.0.2): icmp_seq=3 ttl=64 time=0.053 ms

--- db01.dns.podman ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2046ms
rtt min/avg/max/mdev = 0.048/0.055/0.064/0.006 ms
```

Testez la résolution des noms **db_net** :

```
[trainee@redhat9 ~]$ podman exec -it client ping -c3 www.free.fr
PING www.free.fr (212.27.48.10) 56(84) bytes of data.
64 bytes from www.free.fr (212.27.48.10): icmp_seq=1 ttl=254 time=88.3 ms
64 bytes from www.free.fr (212.27.48.10): icmp_seq=2 ttl=254 time=88.1 ms
64 bytes from www.free.fr (212.27.48.10): icmp_seq=3 ttl=254 time=88.3 ms

--- www.free.fr ping statistics ---
```

```
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 88.116/88.250/88.327/0.356 ms
```

Créez le troisième réseau **backend** :

```
[trainee@redhat9 ~]$ podman network create backend
backend

[trainee@redhat9 ~]$ podman network ls
NETWORK ID    NAME        DRIVER
9c7fcc7b2b5d  backend     bridge
556c9797ea6d  db_net      bridge
2f259bab93aa  podman      bridge

[trainee@redhat9 ~]$ podman network inspect backend
[
     {
          "name": "backend",
          "id": "9c7fcc7b2b5dea98f34acecbd58ef076675edd475f059f36bf6c84f476de653a",
          "driver": "bridge",
          "network_interface": "podman2",
          "created": "2024-10-31T12:10:36.370384672+01:00",
          "subnets": [
               {
                    "subnet": "10.89.0.0/24",
                    "gateway": "10.89.0.1"
               }
          ],
          "ipv6_enabled": false,
          "internal": false,
          "dns_enabled": true,
          "ipam_options": {
               "driver": "host-local"
          }
```

```
     }
]
```

Placez les deux conteneurs existants dans ce nouveau réseau :

```
[trainee@redhat9 ~]$ podman network connect backend db01

[trainee@redhat9 ~]$ podman network connect backend client
```

Constatez que les deux conteneurs se trouvent dans les deux réseaux, **backend** et **db_net** :

```
[trainee@redhat9 ~]$ podman inspect db01
[
...
               "Networks": {
                    "backend": {
                         "EndpointID": "",
                         "Gateway": "10.89.0.1",
                         "IPAddress": "10.89.0.2",
                         "IPPrefixLen": 24,
                         "IPv6Gateway": "",
                         "GlobalIPv6Address": "",
                         "GlobalIPv6PrefixLen": 0,
                         "MacAddress": "ca:67:46:a3:a9:01",
                         "NetworkID": "backend",
                         "DriverOpts": null,
                         "IPAMConfig": null,
                         "Links": null,
                         "Aliases": [
                              "36c9f95ae52f"
                         ]
                    },
                    "db_net": {
                         "EndpointID": "",
```

```
                         "Gateway": "10.87.0.1",
                         "IPAddress": "10.87.0.2",
                         "IPPrefixLen": 16,
                         "IPv6Gateway": "",
                         "GlobalIPv6Address": "",
                         "GlobalIPv6PrefixLen": 0,
                         "MacAddress": "3e:ee:2c:34:2e:ed",
                         "NetworkID": "db_net",
                         "DriverOpts": null,
                         "IPAMConfig": null,
                         "Links": null,
                         "Aliases": [
                              "36c9f95ae52f"
                         ]
                    }
               }
          },
...
]
```

```
[trainee@redhat9 ~]$ podman inspect client
[
...
               "Networks": {
                    "backend": {
                         "EndpointID": "",
                         "Gateway": "10.89.0.1",
                         "IPAddress": "10.89.0.3",
                         "IPPrefixLen": 24,
                         "IPv6Gateway": "",
                         "GlobalIPv6Address": "",
                         "GlobalIPv6PrefixLen": 0,
                         "MacAddress": "c6:eb:7a:f3:b0:5c",
                         "NetworkID": "backend",
```

```
                         "DriverOpts": null,
                         "IPAMConfig": null,
                         "Links": null,
                         "Aliases": [
                              "00b2b82c2adf"
                         ]
                    },
                    "db_net": {
                         "EndpointID": "",
                         "Gateway": "10.87.0.1",
                         "IPAddress": "10.87.0.3",
                         "IPPrefixLen": 16,
                         "IPv6Gateway": "",
                         "GlobalIPv6Address": "",
                         "GlobalIPv6PrefixLen": 0,
                         "MacAddress": "06:e3:cb:75:63:6b",
                         "NetworkID": "db_net",
                         "DriverOpts": null,
                         "IPAMConfig": null,
                         "Links": null,
                         "Aliases": [
                              "00b2b82c2adf"
                         ]
                    }
               }
          },
...
]
```

LAB #5 - Gestion des Conteneurs en tant que Services Système

Sans faire appel à un Orchestrateur de Conteneurs tel Kubernetes, il est possible de gérer des petites infrastructures en utilisant simplement **-**

systemd.

5.1 - Création du Gestionnaire de Conteneurs

Commencez par créer un compte qui sera utilisé pour gérer les conteneurs :

```
[trainee@redhat9 ~]$ su -
Password: fenestros

[root@redhat9 ~]# groupadd fenestros && useradd fenestros -c FenestrOs -d /home/fenestros -g fenestros -s
/bin/bash

[root@redhat9 ~]# passwd fenestros
Changing password for user fenestros.
New password: fenestros
BAD PASSWORD: The password contains the user name in some form
Retype new password: fenestros
passwd: all authentication tokens updated successfully.
```

Devenez l'utilisateur et constatez l'erreur lors de l'exécution de la commande **podman info** :

```
[root@redhat9 ~]# su - fenestros

[fenestros@redhat9 ~]$ podman info
WARN[0000] The cgroupv2 manager is set to systemd but there is no systemd user session available
WARN[0000] For using systemd, you may need to log in using a user session
WARN[0000] Alternatively, you can enable lingering with: `loginctl enable-linger 1001` (possibly as root)
WARN[0000] Falling back to --cgroup-manager=cgroupfs
...
```

Podman est un utilitaire sans état et nécessite une session de connexion complète. Podman doit être utilisé dans une session SSH et ne peut pas être utilisé dans un shell sudo ou su :

```
[fenestros@redhat9 ~]$ exit
logout

[root@redhat9 ~]# ssh fenestros@localhost
The authenticity of host 'localhost (::1)' can't be established.
ED25519 key fingerprint is SHA256:k/cooDrUynjprBohmFjJd22Ii2xlCXFdTHt/HAXpDE4.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
fenestros@localhost's password: fenestros
Register this system with Red Hat Insights: insights-client --register
Create an account or view all your systems at https://red.ht/insights-dashboard
Last login: Thu Oct 31 13:04:30 2024

[fenestros@redhat9 ~]$ podman info
host:
  arch: amd64
  buildahVersion: 1.33.8
  cgroupControllers:
  - memory
  - pids
  cgroupManager: systemd
  cgroupVersion: v2
  conmon:
    package: conmon-2.1.10-1.el9.x86_64
    path: /usr/bin/conmon
    version: 'conmon version 2.1.10, commit: fb8c4bf50dbc044a338137871b096eea8041a1fa'
  cpuUtilization:
    idlePercent: 49.45
    systemPercent: 1.22
    userPercent: 49.33
  cpus: 4
  databaseBackend: sqlite
  distribution:
```

```
    distribution: rhel
    version: "9.4"
...
```

5.2 - Création d'un Conteneur de Serveur Web

Commencez par créer un répertoire dans l'hôte pour contenir les pages à publier par le serveur web :

```
[fenestros@redhat9 ~]$ mkdir www
```

Créez ensuite un conteneur, appelé **webserver** avec un mappage de ports, à partir de l'image **registry.access.redhat.com/ubi8/httpd-24** :

```
[fenestros@redhat9 ~]$ podman run -d --name webserver -p 8080:8080 -v ~/www:/var/www/html:Z
registry.access.redhat.com/ubi8/httpd-24
Trying to pull registry.access.redhat.com/ubi8/httpd-24:latest...
Getting image source signatures
Checking if image destination supports signatures
Copying blob c302a7fbb8d5 done   |
Copying blob 148a3ed2f70e done   |
Copying blob 4de1bba6ee61 done   |
Copying config e11bc11181 done   |
Writing manifest to image destination
Storing signatures
e8afda5806a98f378865230e6d5c900c85afc0ea152609602a65d05c60cd68dc
```

Utilisez la commande **podman generate systemd** pour créer un fichier d'unité de service pour le conteneur **webserveur** :

```
[fenestros@redhat9 ~]$ podman generate systemd --name webserver --new --files

DEPRECATED command:
It is recommended to use Quadlets for running containers and pods under systemd.
```

```
Please refer to podman-systemd.unit(5) for details.
/home/fenestros/container-webserver.service
```

> **Important** - Notez l'utilisation de l'option **–new**. Cette option demande à l'utilitaire podman de configurer le service systemd pour qu'il crée le conteneur au démarrage du service, et qu'il le supprime à l'arrêt du service. Sans l'option –new, l'utilitaire podman configure le fichier d'unité de service pour démarrer et arrêter le conteneur existant sans le supprimer.

Le fichier est créé dans le répertoire courant :

```
[fenestros@redhat9 ~]$ cat container-webserver.service
# container-webserver.service
# autogenerated by Podman 4.9.4-rhel
# Thu Oct 31 13:42:12 CET 2024

[Unit]
Description=Podman container-webserver.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=%t/containers

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman run \
        --cidfile=%t/%n.ctr-id \
        --cgroups=no-conmon \
        --rm \
```

```
        --sdnotify=conmon \
        --replace \
        -d \
        --name webserver \
        -p 8080:8080 \
        -v /home/fenestros/www:/var/www/html:Z registry.access.redhat.com/ubi8/httpd-24
ExecStop=/usr/bin/podman stop \
        --ignore -t 10 \
        --cidfile=%t/%n.ctr-id
ExecStopPost=/usr/bin/podman rm \
        -f \
        --ignore -t 10 \
        --cidfile=%t/%n.ctr-id
Type=notify
NotifyAccess=all

[Install]
WantedBy=default.target
```

Pour que systemd puisse prendre en charge le service, le fichier d'unité de service doit être placé dans le répertoire **$HOME/.config/systemd/user/** :

```
[fenestros@redhat9 ~]$ mkdir -p ~/.config/systemd/user/

[fenestros@redhat9 ~]$ mv container-webserver.service ~/.config/systemd/user/
```

Pour créer le service, utilisez la commande **systemctl –user daemon-reload**. Notez que l'utilisation de l'option **–user** informe systemctl de regarder dans le répertoire **$HOME/.config/systemd/user/** et non dans le répertoire **/etc/systemd/system/** :

```
[fenestros@redhat9 ~]$ systemctl --user daemon-reload
```

Vérifiez le statut du service :

```
[fenestros@redhat9 ~]$ systemctl --user status container-webserver.service
○ container-webserver.service - Podman container-webserver.service
```

```
     Loaded: loaded (/home/fenestros/.config/systemd/user/container-webserver.service; disabled; preset:
disabled)
     Active: inactive (dead)
       Docs: man:podman-generate-systemd(1)
```

Dans l'état et par défaut, si ce service était activé et démarré, celui-ci s'arrêterait à la fin de la session courante pour être démarrer de nouveau lors de la prochaine ouverture de session. Ce comportement est configuré par la valeur de la directive **Linger** :

```
[fenestros@redhat9 ~]$ loginctl show-user fenestros
UID=1001
GID=1001
Name=fenestros
Timestamp=Thu 2024-10-31 13:09:25 CET
TimestampMonotonic=240322297212
RuntimePath=/run/user/1001
Service=user@1001.service
Slice=user-1001.slice
Display=7200
State=active
Sessions=7200
IdleHint=no
IdleSinceHint=1730379006672000
IdleSinceHintMonotonic=242763627399
Linger=no
```

Afin que le service reste démarré après la fermeture de la session et qu'il réagisse comme un service de système, il faut modifier la valeur de Linger à **yes** :

```
[fenestros@redhat9 ~]$ loginctl enable-linger

[fenestros@redhat9 ~]$ loginctl show-user fenestros
UID=1001
GID=1001
Name=fenestros
```

```
Timestamp=Thu 2024-10-31 13:09:25 CET
TimestampMonotonic=240322297212
RuntimePath=/run/user/1001
Service=user@1001.service
Slice=user-1001.slice
Display=7200
State=active
Sessions=7200
IdleHint=no
IdleSinceHint=1730379080672000
IdleSinceHintMonotonic=242837627399
Linger=yes
```

Créez un fichier **index.html** dans le répertoire **$HOME/www/** :

```
[fenestros@redhat9 ~]$ echo "Hello World" > ~/www/index.html
```

Supprimez le conteneur actuel **webserver** en utilisant l'option **-f** de la commande **podman rm** :

```
[fenestros@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                         COMMAND               CREATED         STATUS
PORTS                   NAMES
e8afda5806a9  registry.access.redhat.com/ubi8/httpd-24:latest  /usr/bin/run-http...  44 minutes ago  Up 44
minutes  0.0.0.0:8080->8080/tcp  webserver

[fenestros@redhat9 ~]$ podman rm -f webserver
webserver
```

Activez et démarrez le service utilisateur **container-webserver** :

```
[fenestros@redhat9 ~]$ systemctl --user enable --now container-webserver
Created symlink /home/fenestros/.config/systemd/user/default.target.wants/container-webserver.service →
/home/fenestros/.config/systemd/user/container-webserver.service.
```

```
[fenestros@redhat9 ~]$ systemctl --user status  container-webserver
● container-webserver.service - Podman container-webserver.service
     Loaded: loaded (/home/fenestros/.config/systemd/user/container-webserver.service; enabled; preset: disabled)
     Active: active (running) since Thu 2024-10-31 13:59:23 CET; 15s ago
       Docs: man:podman-generate-systemd(1)
   Main PID: 235072 (conmon)
      Tasks: 16 (limit: 48799)
     Memory: 10.7M
        CPU: 132ms
     CGroup: /user.slice/user-1001.slice/user@1001.service/app.slice/container-webserver.service
             ├─235055 /usr/bin/slirp4netns --disable-host-loopback --mtu=65520 --enable-sandbox --enable-seccomp
--enable-ipv6 -c -r 3 -e 4 --netns-type=path /run/user/1001/netns/netns-633c20d2-eef3-b41e-f850->
             ├─235057 rootlessport
             ├─235064 rootlessport-child
             └─235072 /usr/bin/conmon --api-version 1 -c
ed306cecef9e73b928ceb98308aba3e3b4c255c4517212139c87a428548c31e3 -u
ed306cecef9e73b928ceb98308aba3e3b4c255c4517212139c87a428548c31e3 -r /usr/bin/crun -b>

[fenestros@redhat9 ~]$ podman ps -a
CONTAINER ID  IMAGE                                           COMMAND               CREATED         STATUS
PORTS                   NAMES
ed306cecef9e  registry.access.redhat.com/ubi8/httpd-24:latest  /usr/bin/run-http...  29 seconds ago  Up 28
seconds  0.0.0.0:8080->8080/tcp  webserver
```

Déconnectez-vous du compte fenestros et vérifiez que le serveur Web fonctionne :

```
[fenestros@redhat9 ~]$ exit
logout
Connection to localhost closed.

[root@redhat9 ~]# curl http://localhost:8080
Hello World
```