**CERI Avignon**

**Le DevOps (S-E06-3010) – Semestre 0**

**TP2 – Docker : Conteneurs et Images**

**ALDENAWI Hamad**

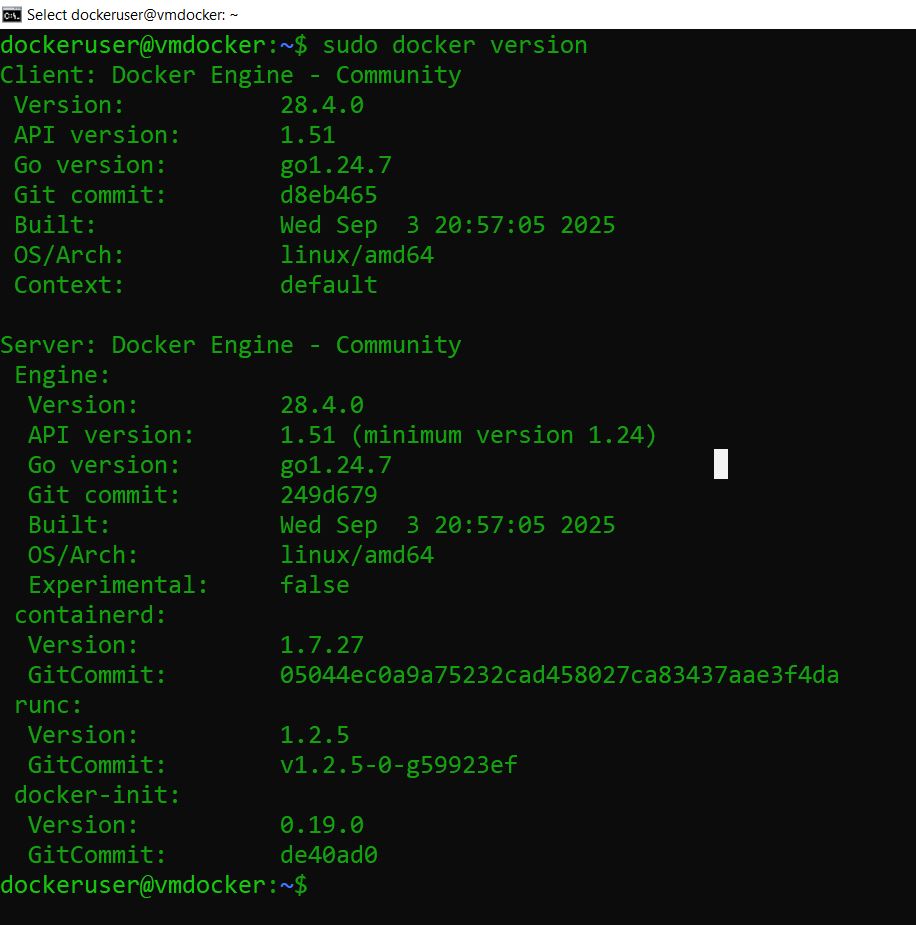
**Étudiant(e) : uapv2603127**

**Fichier de projet**

**Le Date : /09/2025**

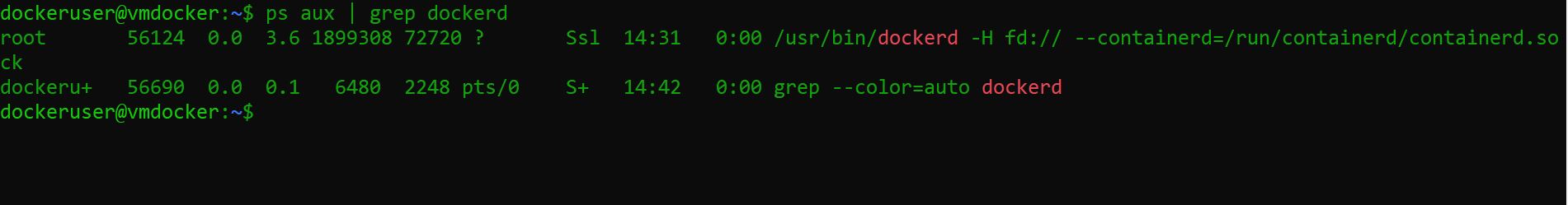
**Partie 2 – Installation et démarrage de Docker**

1. **Pour afficher la version de Docker, y compris le client et le serveur (daemon), utilise la commande suivante :**

sudo docker version

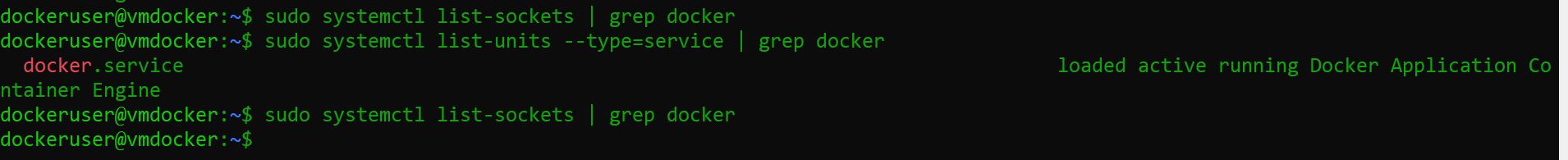
**2-Pour voir les composants du daemon Docker qui tournent :**

**ps aux | grep dockerd**

****

3-Pour voir **les services et sockets utilisés par Docker** ainsi que **l’utilisateur qui les a démarrés**

sudo systemctl list-units --type=service | grep docker



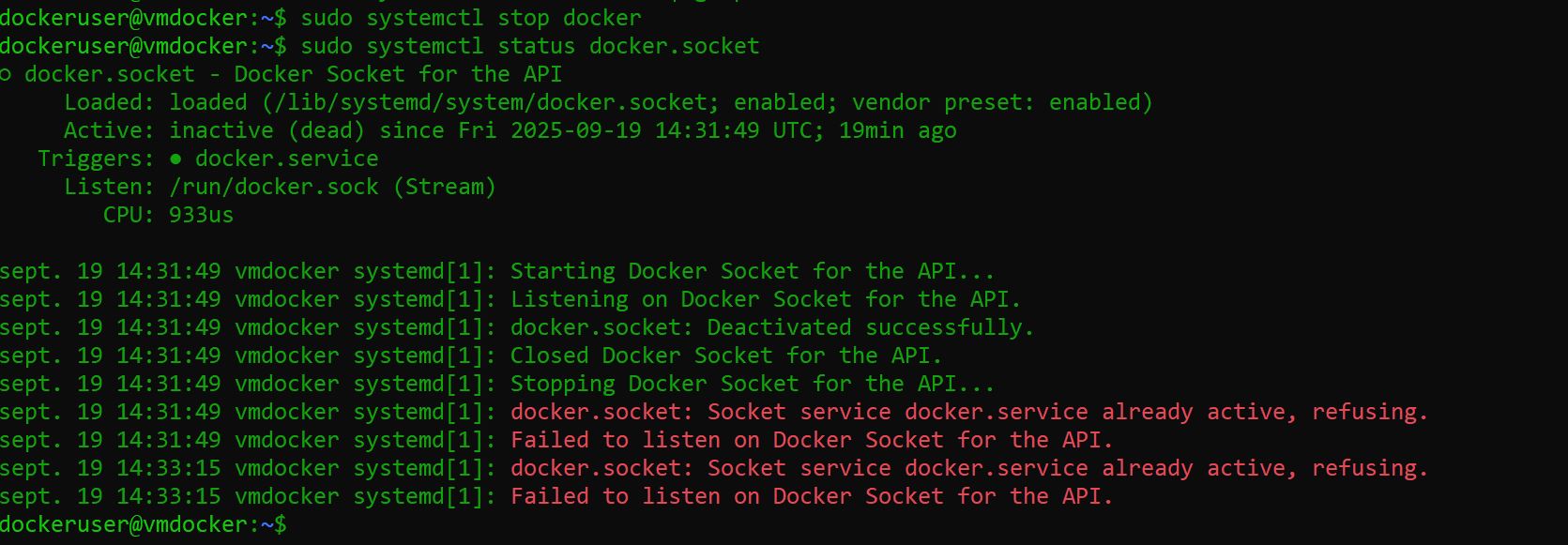
4- Pour arrêter Docker et vérifier le statut du socket :

**1️⃣ Arrêter Docker**

sudo systemctl stop docker

**2️⃣ Vérifier le statut du socket**

sudo systemctl status docker.socket



5-**Désactiver Docker au démarrage**

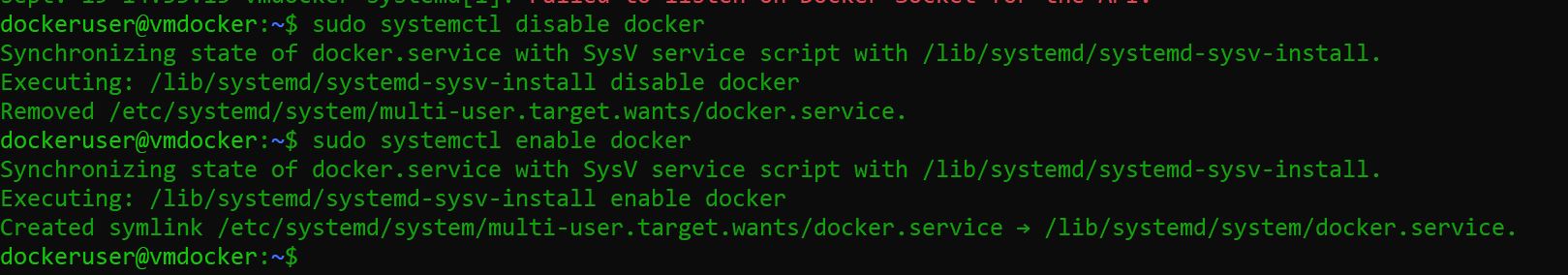
sudo systemctl disable docker

* Empêche Docker de démarrer automatiquement au boot.

**Réactiver Docker au démarrage**

sudo systemctl enable docker

* Permet à Docker de démarrer automatiquement au boot.



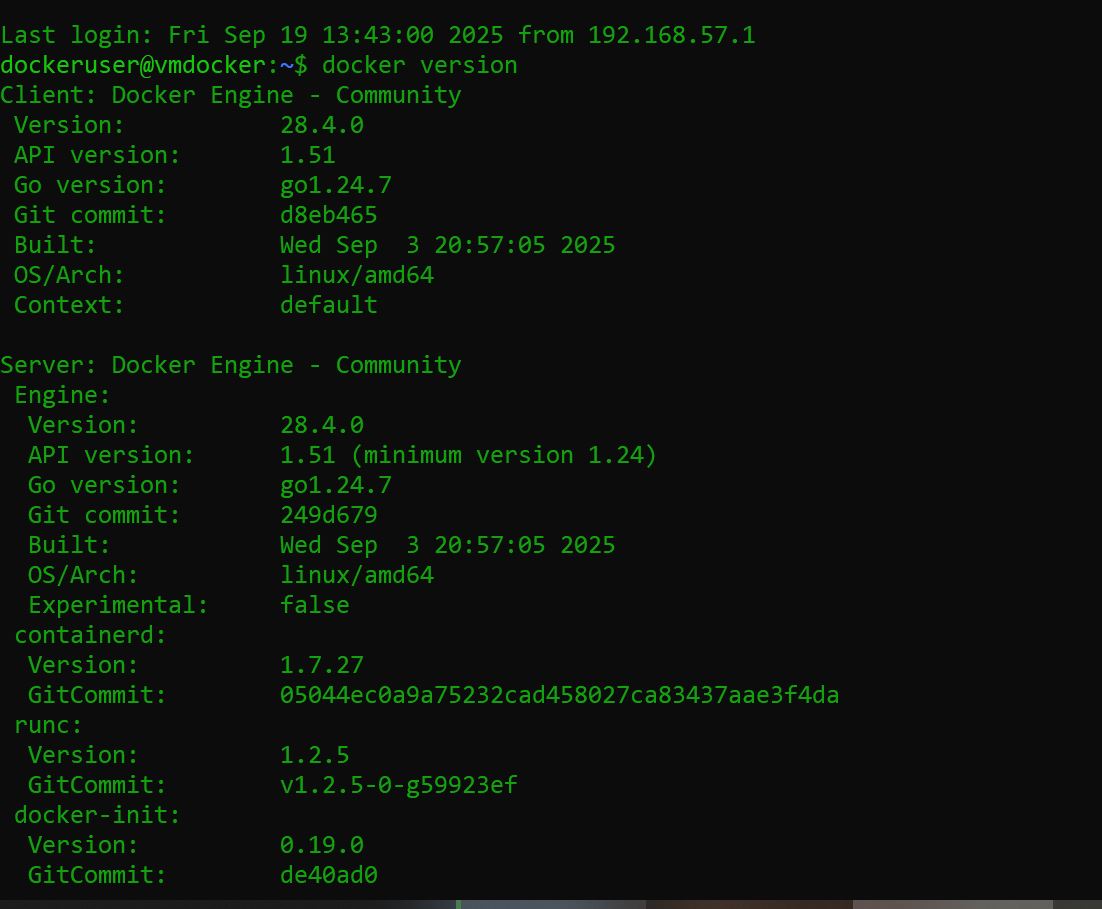
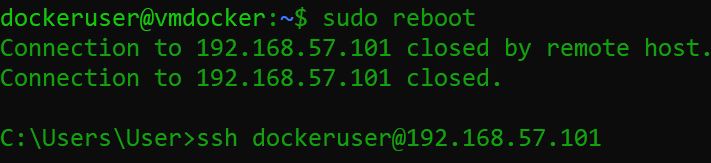
6-**1-Créer un groupe docker**

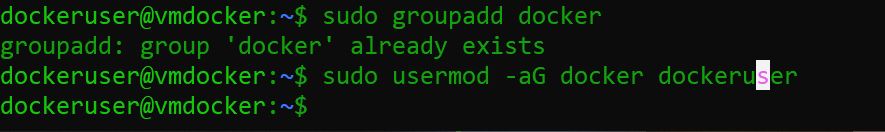
sudo groupadd docker

**2-Ajouter dockeruser au groupe docker**

sudo usermod -aG docker dockeruser

utilise sudo reebot pour déconnecter/reconnecter dockeruser





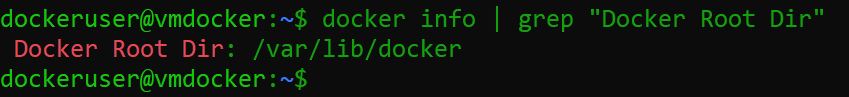
**Partie 3**

**1-dans quel répertoire Docker stocke-t-il ses objets (images, conteneurs, volumes…) ?**

* docker info | grep "Docker Root Dir

**Contenu de ce répertoire**

* containers/ → tous les conteneurs existants et arrêtés
* image/ → toutes les images téléchargées
* volumes/ → les volumes persistants
* overlay2/ ou aufs/ → système de fichiers des conteneurs (selon le driver)
* network/ → configuration des réseaux Docker

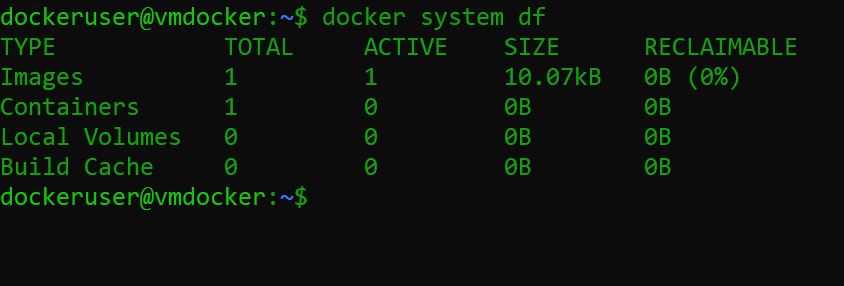


**2-Quels sont les différentes catégories d’objets Docker qui peuvent être stockés ?**

**Objets Docker :**

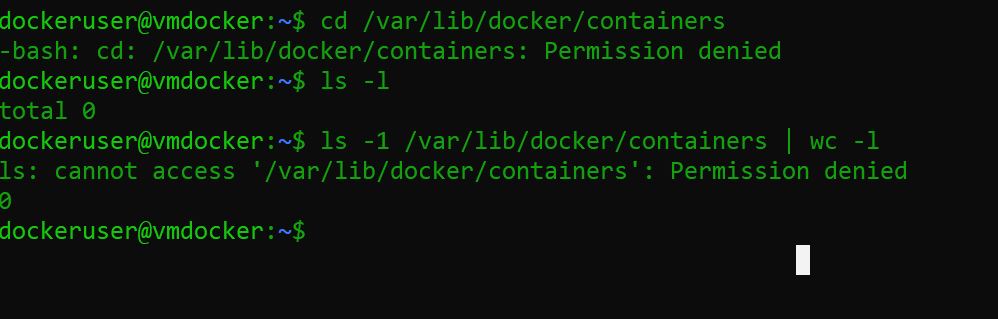
Les objets Docker sont les éléments gérés par Docker pour créer et exécuter des applications : **images** (templates des conteneurs), **conteneurs** (instances actives ou arrêtées), **volumes** (stockage persistant) et **réseaux** (communication entre conteneurs).

docker system df



**3-Pour consulter le contenu du répertoire où Docker stocke les conteneurs :**

* cd /var/lib/docker/containers
* ls -l
* ls -1 /var/lib/docker/containers | wc -l

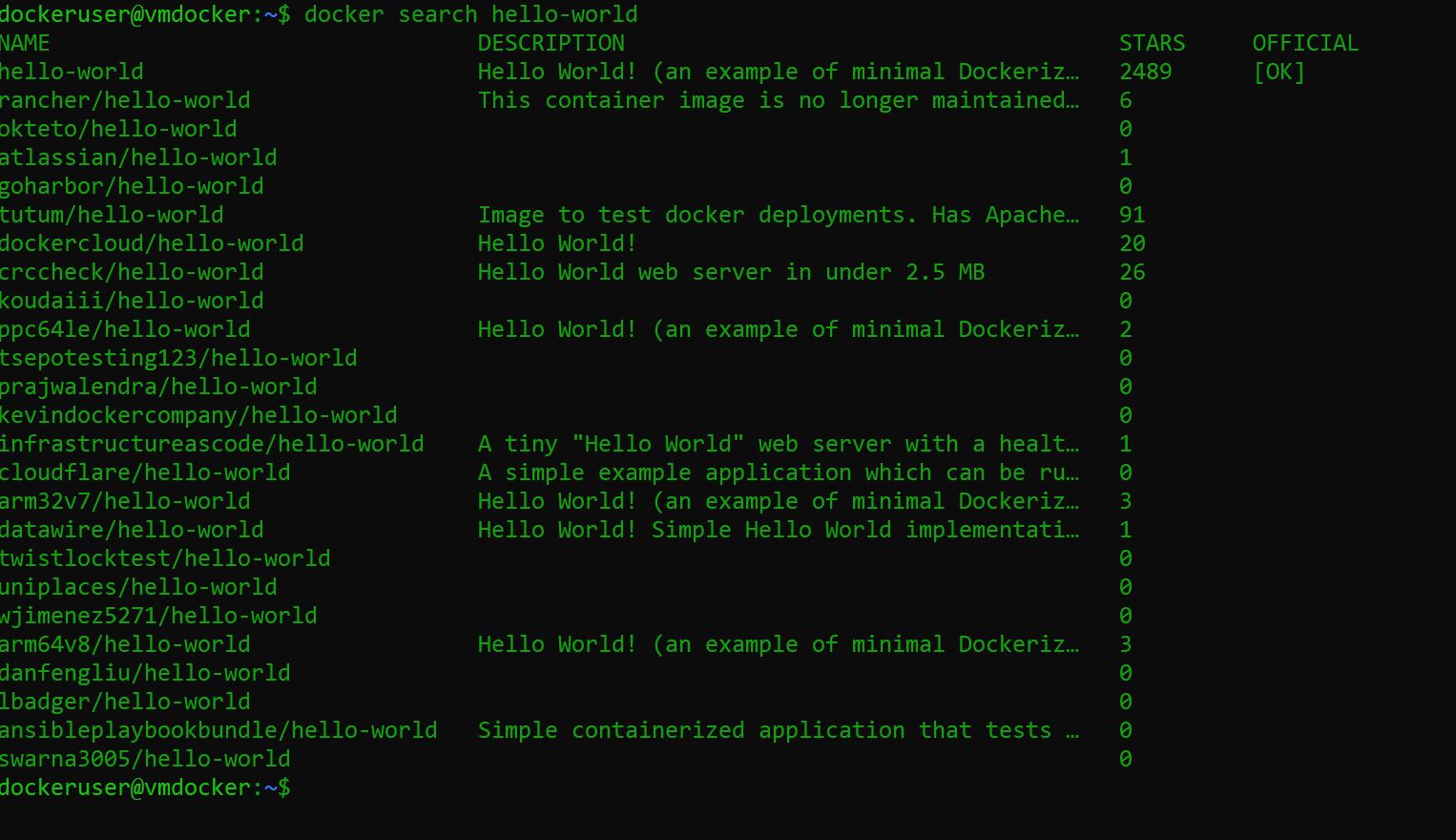


**4-Pour rechercher une image sur Docker Hub, tu peux utiliser la commande :**

* docker search hello-world

**Explication :**

* docker search permet de **chercher des images disponibles sur Docker Hub**.
* hello-world est le nom de l’image que tu veux trouver.



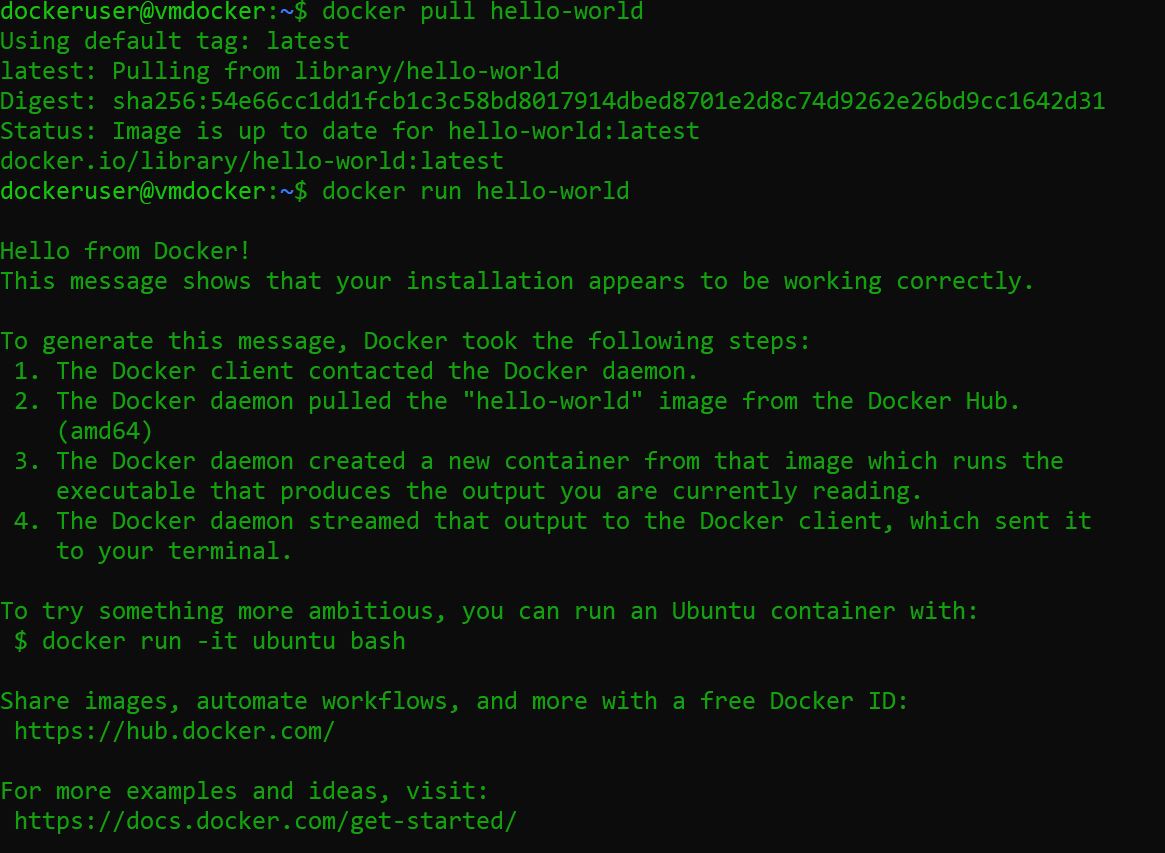
**5-- Faites exécuter un conteneur qui correspond à l’image ayant le plus d’étoiles. Faites une copie d’écran qui enregistre les étapes réalisées par docker.**

* Télécharger l’image: **Télécharger l’image :** **docker pull hello-world**

### **Lancer un conteneur à partir de cette image : docker run hello-world**

**Explication**

* docker run : crée et exécute un conteneur à partir de l’image spécifiée.
* Docker affichera dans le terminal un message de test confirmant que le conteneur fonctionne

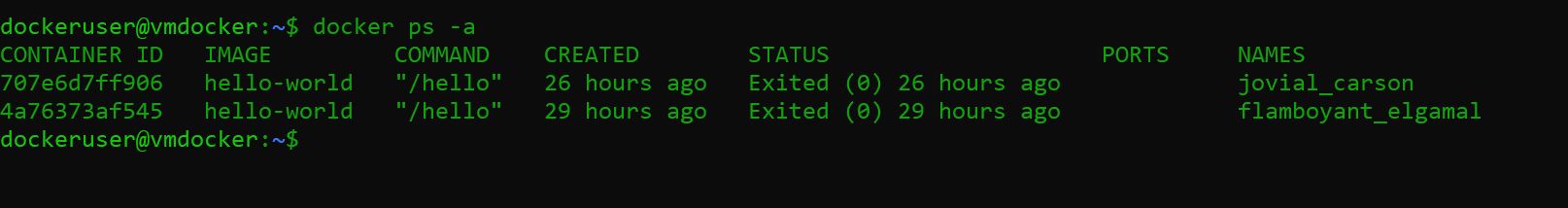
**Résultat:**

**6-1-Vérifiez maintenant le contenu du répertoire des conteneurs.**

**Avec la commande Docker :** docker ps -a

**Explication :**

* docker ps → liste seulement les conteneurs actifs.
* docker ps -a → liste **tous** les conteneurs (actifs et stoppés).

**Résultat:**

**Combien y a-t-il de conteneurs ?**

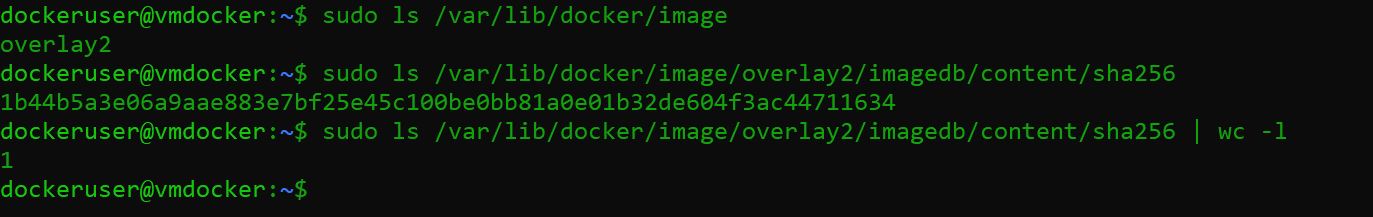
* **Les conteneurs sont stockés dans : sudo ls /var/lib/docker/containers**
* **Chaque sous-répertoire correspond à un conteneur : sudo ls /var/lib/docker/containers | wc -l**

**Résultat:**

**7 - Vérifiez aussi le contenu du répertoire des images.**

* Pour compter le nombre d’images : sudo ls /var/lib/docker/image/overlay2/imagedb/content/sha256 | wc -l

sudo ls /var/lib/docker/image/overlay2/imagedb/content/sha256

**Résultat :**

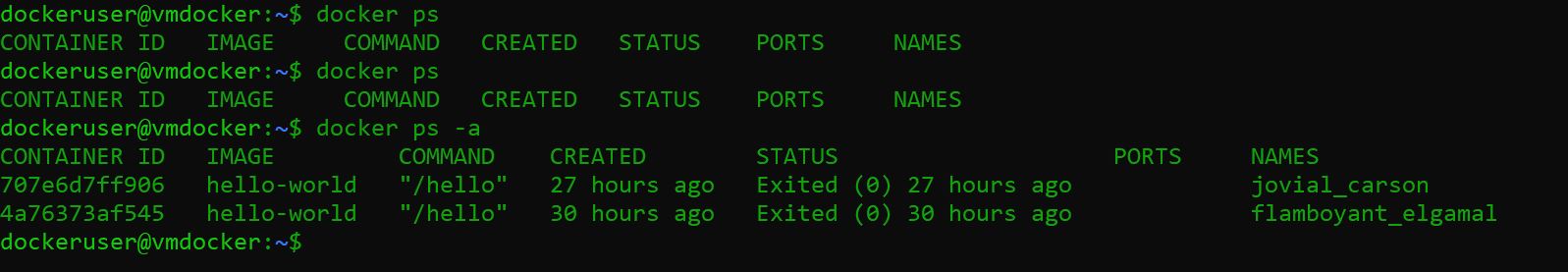
1. **Utilisez la commande docker (sans option) qui permet de lister les conteneurs en exécution.**

docker ps

docker ps -a

**Explication :**

* docker ps affiche **uniquement les conteneurs en cours d’exécution**.
* Dans cet exemple، le conteneur hello-world s’est déjà terminé automatiquement donc il peut apparaître vide.

**Résultat :**

1. **Combien de conteneurs qui s’exécutent ?**

docker ps | wc -l

**Explication :**

* docker ps = liste les conteneurs actifs.
* wc -l = compte le nombre de lignes affichées. Donc il y a 0 conteneur actif .
* Comme la première ligne est l’en-tête, on soustrait 1 pour avoir le nombre réel.

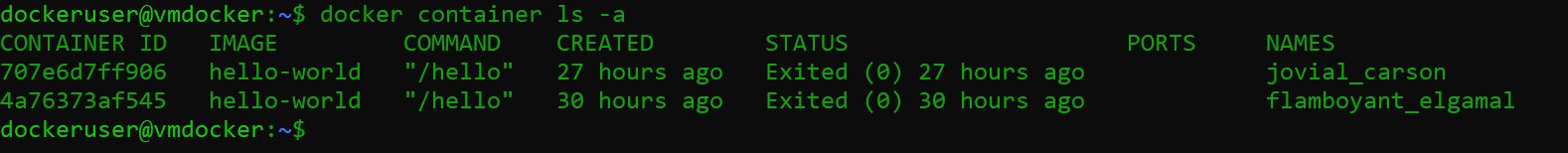
1. **Quel est l’alias de la commande que vous venez d’utiliser ? Remarque : par défaut, si on ne précise pas d’objet, docker considère que l’opération demandée s’applique aux conteneurs.**

**l’alias de la commande docker ps :** docker container ls -a

**Explication :**

* docker ps est un alias pour docker container ls.
* Par défaut, si on ne précise pas d’objet (container, image, etc.), Docker considère que l’opération s’applique aux conteneurs.

Donc docker ps et docker container ls sont équivalentes.

**Résultat :**

1. **Utilisez la commande docker qui permet de lister les images et identifiez l’image utilisée .**

docker images

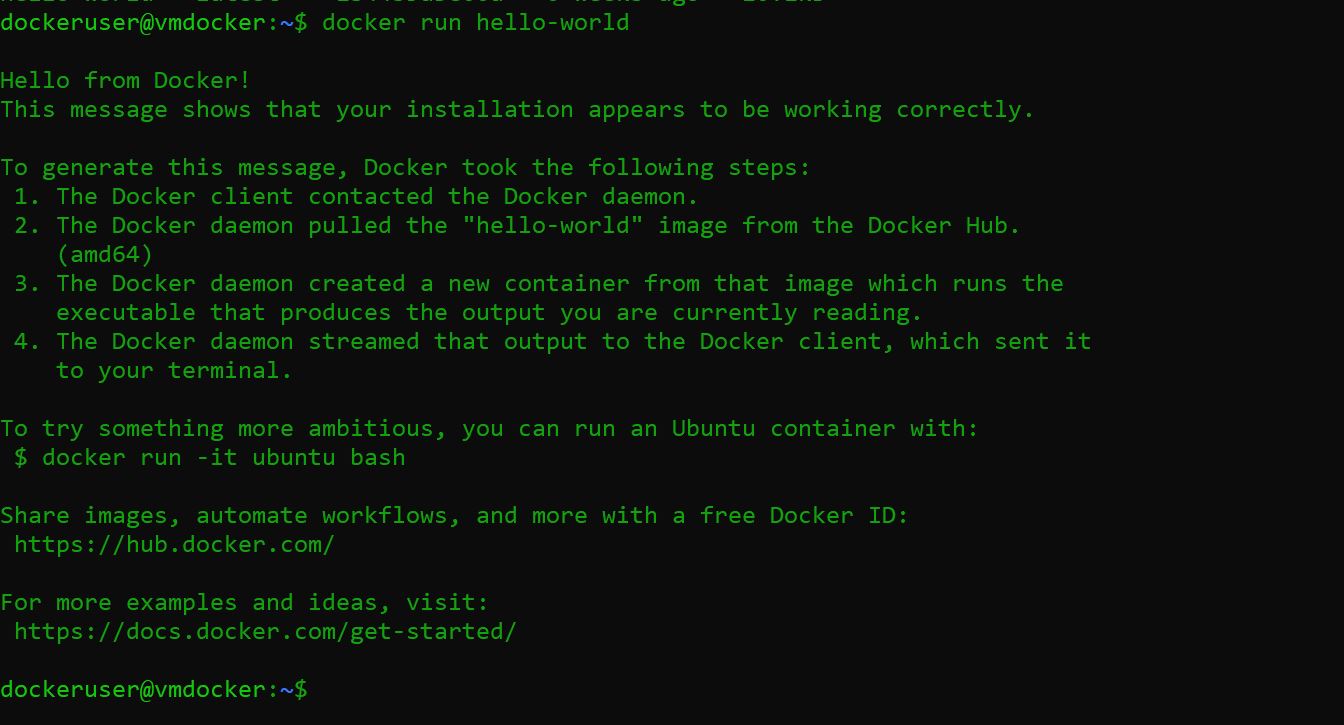
**Explication :**

* docker images liste toutes les images présentes localement.
* Chaque image a un **IMAGE ID** unique (identifiant court ou long) qui sert à référencer l’image pour créer des conteneurs.
* Ici, hello-world est l’image que nous avons utilisée pour lancer le conteneur.

1. **Réexécutez le conteneur et comparez à la précédente exécution : expliquez.**

**Explication :**

* Lors de la **première exécution**, Docker crée un conteneur temporaire à partir de l’image hello-world, exécute le programme et termine automatiquement le conteneur.
* Lors de la **réexécution**, Docker **réutilise la même image locale** pour créer un **nouveau conteneur temporaire**.
* Chaque exécution produit un conteneur **différent**, mais le message affiché reste **identique**, car l’image hello-world fait toujours la même opération (tester l’installation de Docker).
* Conclusion : **le comportement est identique**, mais un nouveau conteneur est créé à chaque fois.

**Résultat**:

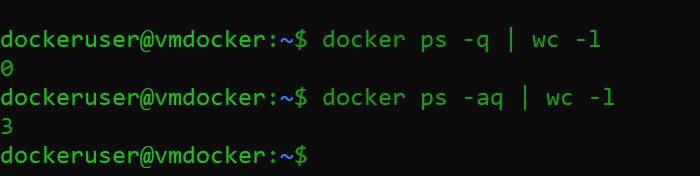
1. **Combien de conteneurs s’exécutent et combien sont stockés localement**

* **Pour les conteneurs en cours d’exécution :** docker ps -q | wc -l
* **Pour tous les conteneurs stockés localement (actifs + stoppés):** docker ps -aq | wc -l

**Explication :**

* docker ps -q → liste uniquement les **IDs des conteneurs actifs**, donc ici **0** car hello-world se termine immédiatement.
* docker ps -aq → liste tous les conteneurs créés sur la machine (actifs et arrêtés). Ici **1** conteneur stocké localement après avoir exécuté hello-world.

Chaque exécution de docker run hello-world crée un nouveau conteneur, même si l’image est déjà présente.

**Résultat :**

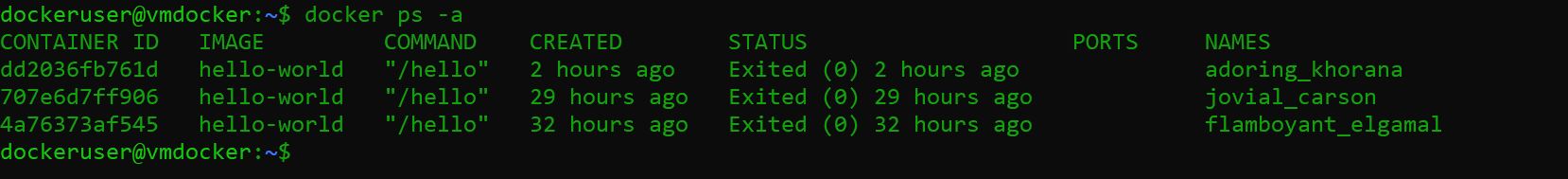
1. **Utilisez une commande qui permet de lister tous les conteneurs et pas uniquement ceux en exécution et observez leur statut.**

### Lister tous les conteneurs (actifs et stoppés) et observer leur statut : docker ps -a

**Explication :**

* docker ps -a affiche **tous les conteneurs**, pas seulement ceux en cours d’exécution.
* **STATUS** indique l’état de chaque conteneur :
  + Exited (0) → conteneur terminé normalement
  + Up X minutes → conteneur en cours d’exécution
* Ici, le conteneur hello-world s’est exécuté puis **s’est terminé automatiquement**, d’où Exited (0).

Cela permet de suivre l’historique de tous les conteneurs créés à partir d’images Docker.

**Résultat :**

1. **Quel est le nom de ces conteneurs ?**

docker ps -a --format "{{.Names}}"

**Explication :**

* Chaque conteneur Docker reçoit **un nom unique** automatiquement si vous n’en spécifiez pas lors de la création.
* Le nom peut être utilisé pour **référencer le conteneur** dans d’autres commandes .

**Résultat :**

**17- Utilisez une option pour afficher l’information de façon non tronquée.**

docker ps -a --no-trunc

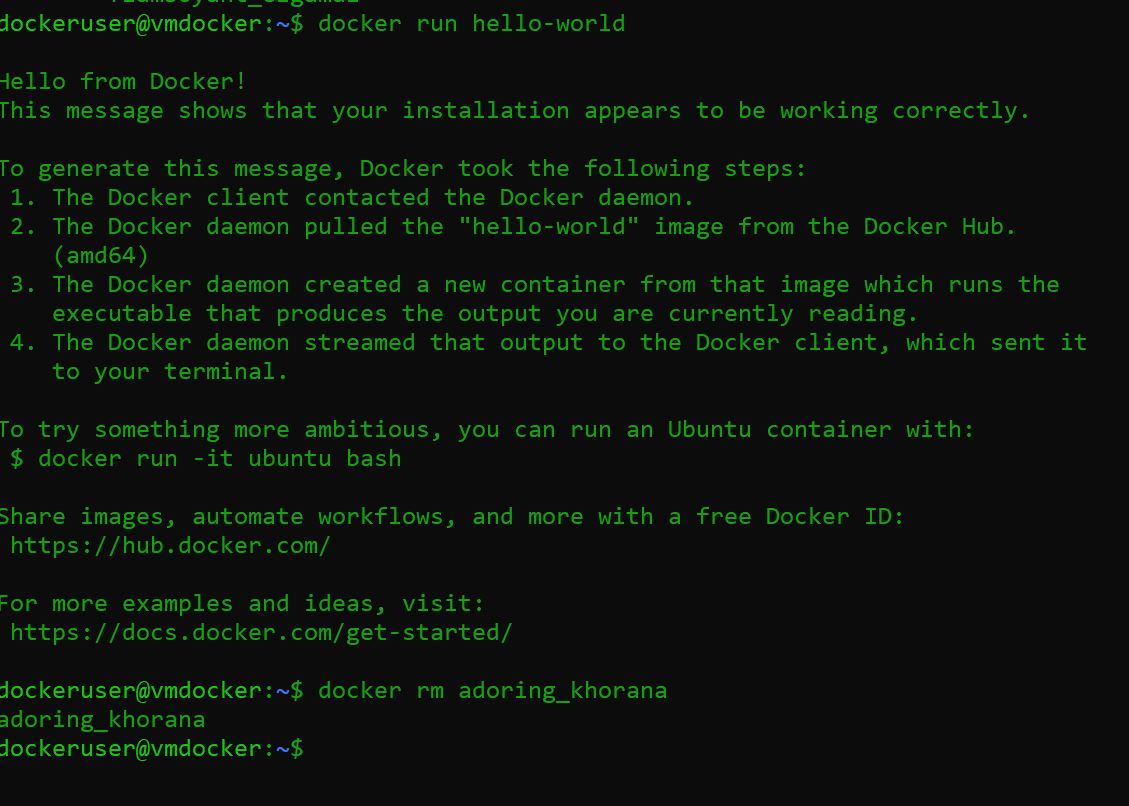
**Explication :**

* L’option --no-trunc **affiche toutes les informations complètes** des conteneurs sans tronquer les ID, noms ou commandes.
* Utile pour voir le **CONTAINER ID complet**, le **nom complet**, et la **commande exacte** utilisée. Par défaut, Docker tronque certaines colonnes pour faciliter la lecture.

**Résultat :**

**18-** **Exécutez une nouvelle fois l’image, puis supprimez ce dernier conteneur en utilisant son nom.**

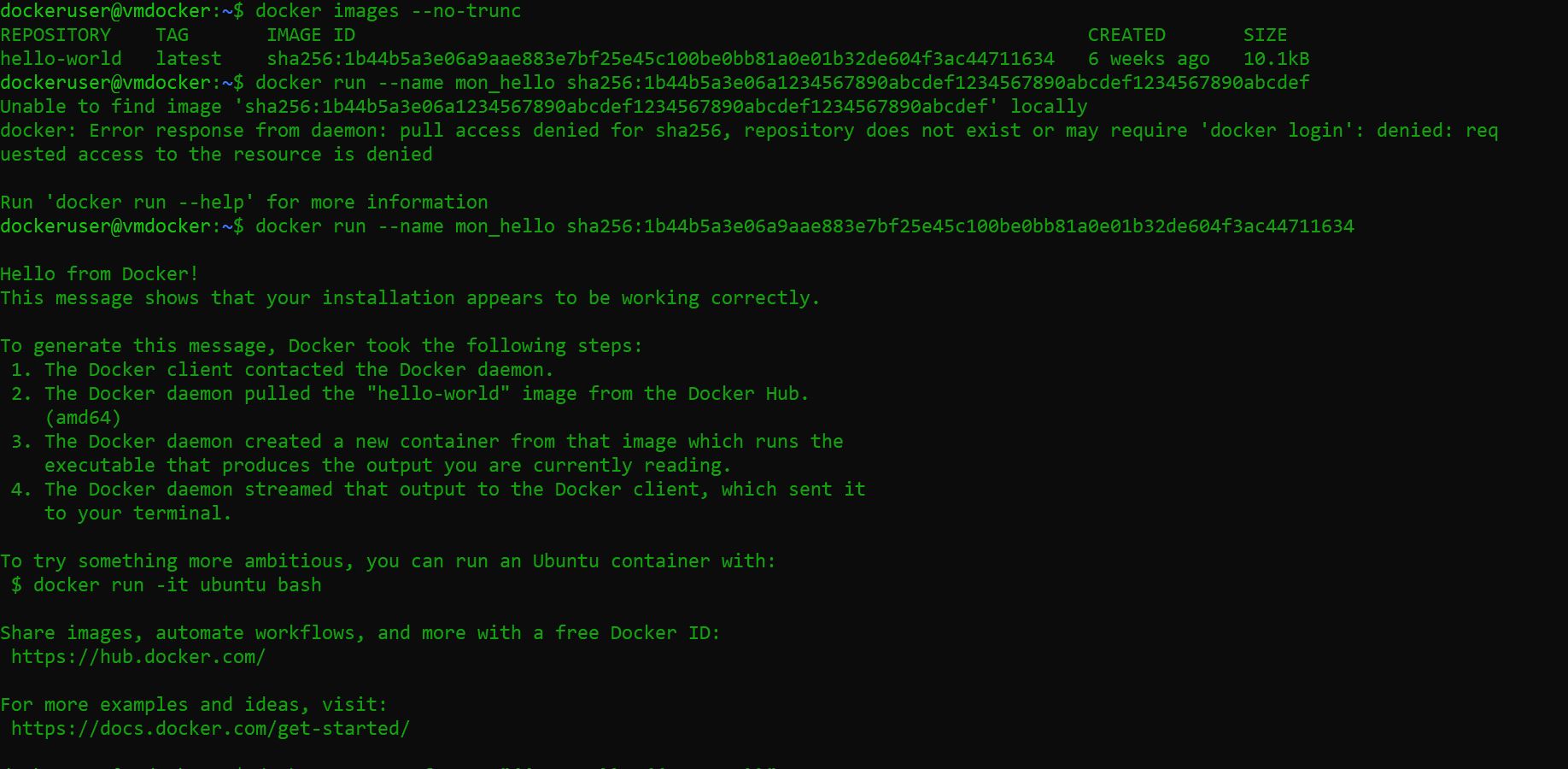
* docker run hello-world
* Supprimer le conteneur en utilisant son nom : docker rm adoring\_khorana

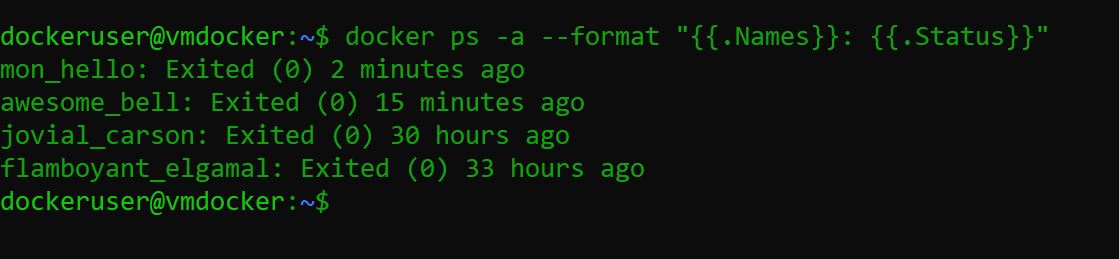
**Résultat :**

**19- Exécutez une nouvelle fois l’image en utilisant son sha256 en donnant un nom au nouveau conteneur, puis constatez.**

* **Identifier le SHA256 de l’image hello-world:** docker images --no-trunc
* **Exécuter un conteneur avec le SHA256 et donner un nom :** docker run --name mon\_hello sha256:1b44b5a3e06a9aae883e7bf25e45c100be0bb81a0e01b32de604f3ac44711634

**Vérifier le conteneur :** docker ps -a --format "{{.Names}}: {{.Status}}"

**Résultat :**



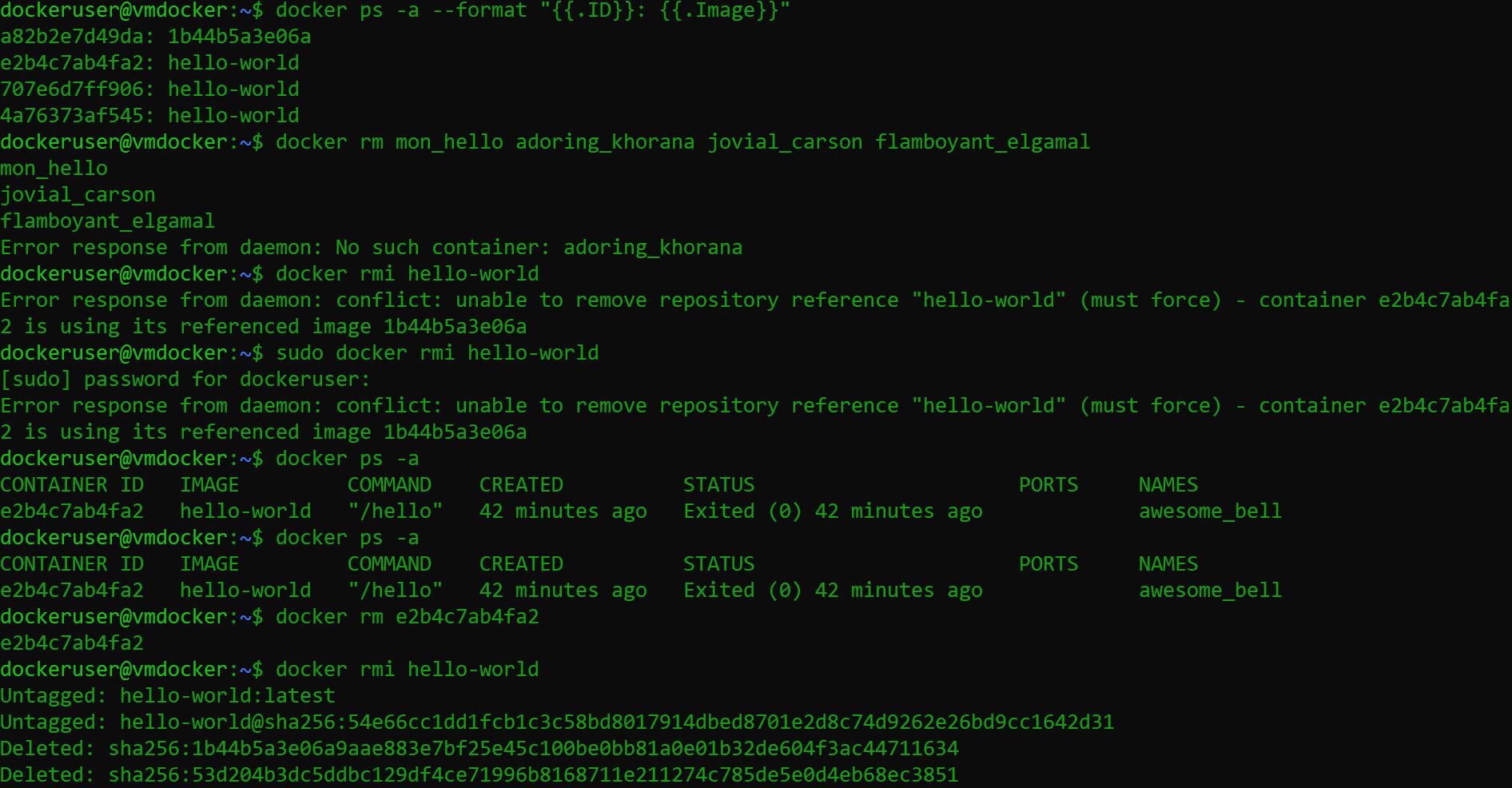
**20- Faites ce qu’il faut pour supprimer l’image (sans forcer).**

* Vérifier les conteneurs qui utilisent l’image : docker ps -a --format "{{.ID}}: {{.Image}}

**Explication :**  
On voit que plusieurs conteneurs (hello-world ou le sha256) utilisent encore cette image.

Donc **Docker refusera la suppression sans d’abord supprimer ces conteneurs**.

* Supprimer les conteneurs liés à l’image : docker rm mon\_hello adoring\_khorana jovial\_carson flamboyant\_elgamal
  + On supprime les conteneurs un par un (ou en liste).
  + Une fois supprimés, l’image n’a plus de conteneurs qui en dépendent.
* Supprimer l’image sans forcer : docker rmi hello-world
  + docker rmi supprime l’image si **aucun conteneur ne l’utilise**.

**Résultat :**

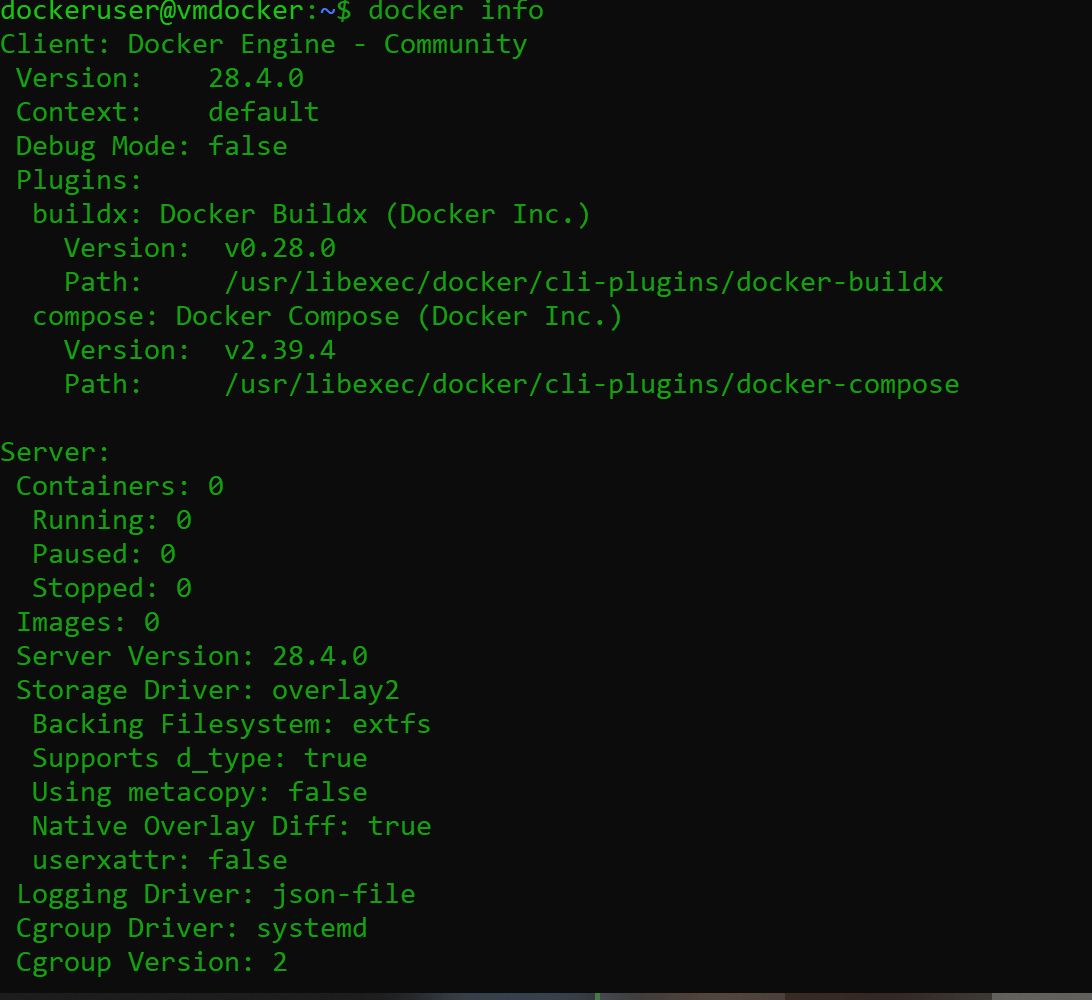
**21- Utilisez une commande qui donne des infos globales sur le système et observez le nombre de conteneurs et d’images.**

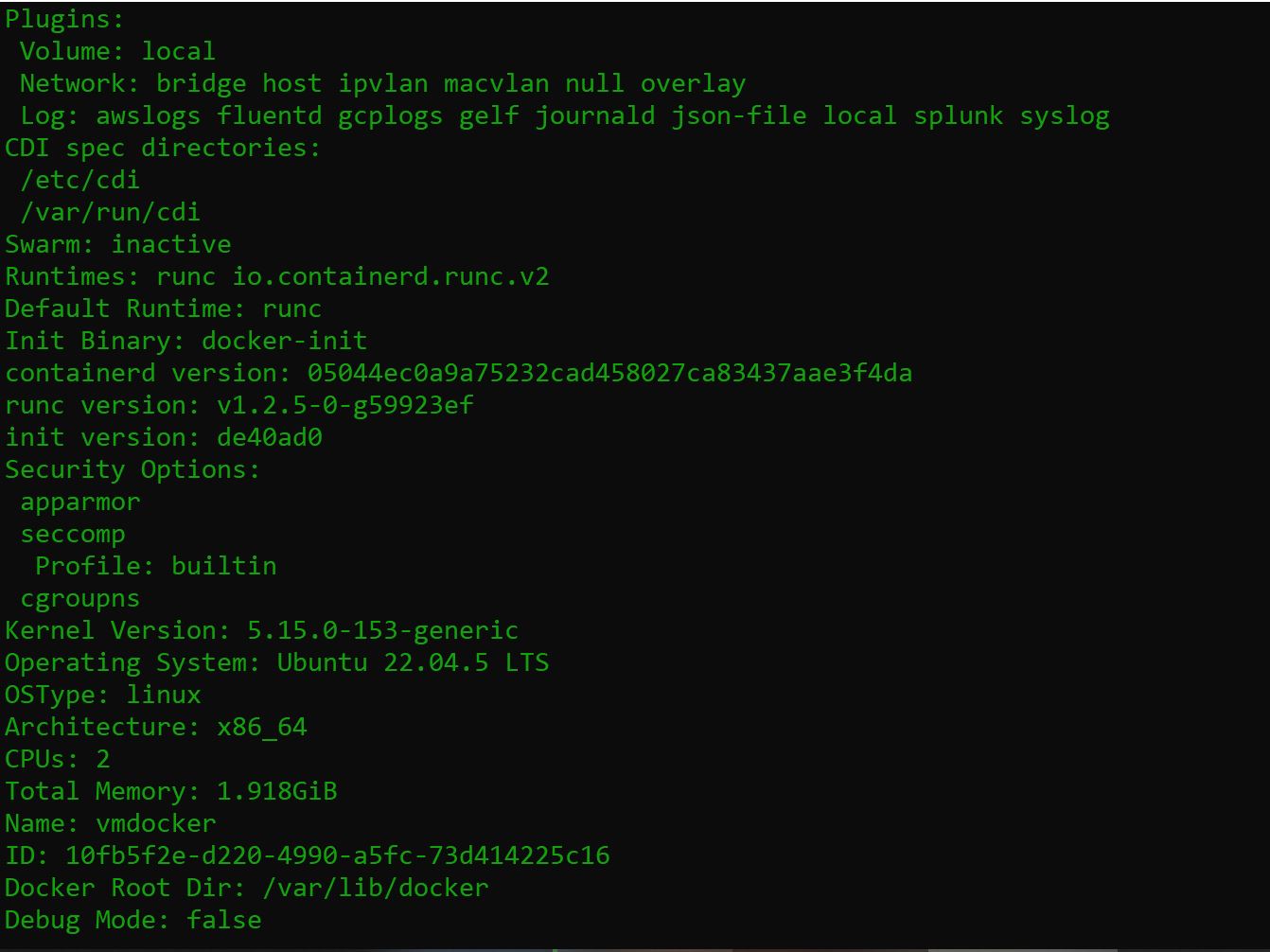
docker info

**Explication :**

* **Containers** → indique le nombre total de conteneurs présents localement, avec leur état (en cours d’exécution, en pause ou arrêtés).
* **Images** → indique le nombre total d’images stockées localement.

**Résultat :**



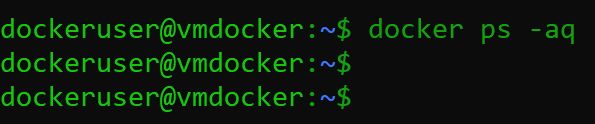


**22- Quelle commande docker pouvez-vous utiliser pour n’afficher que les IDs des conteneurs ?**

docker ps -aq -----🡪 [aucune sortie]

**Explication :**  
L’option -a permet d’afficher tous les conteneurs et -q affiche uniquement leurs identifiants. Ici, la sortie vide signifie qu’il n’existe aucun conteneur localement.

**Résultat :**

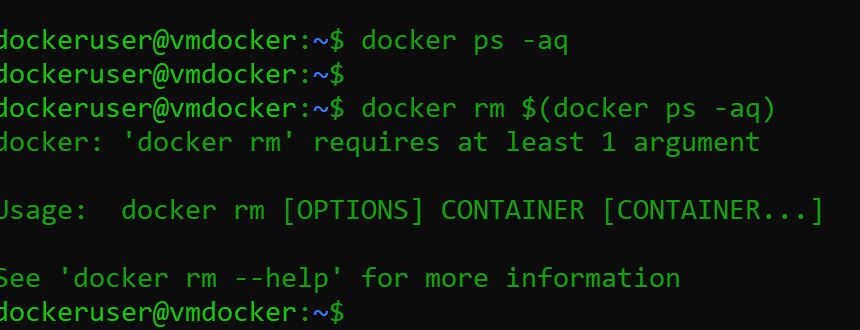


**23- - Utilisez une possibilité du bash pour exploiter les résultats de cette dernière commande afin de supprimer tous les conteneurs en une seule commande.**

docker rm $(docker ps -aq)

**Explication :**

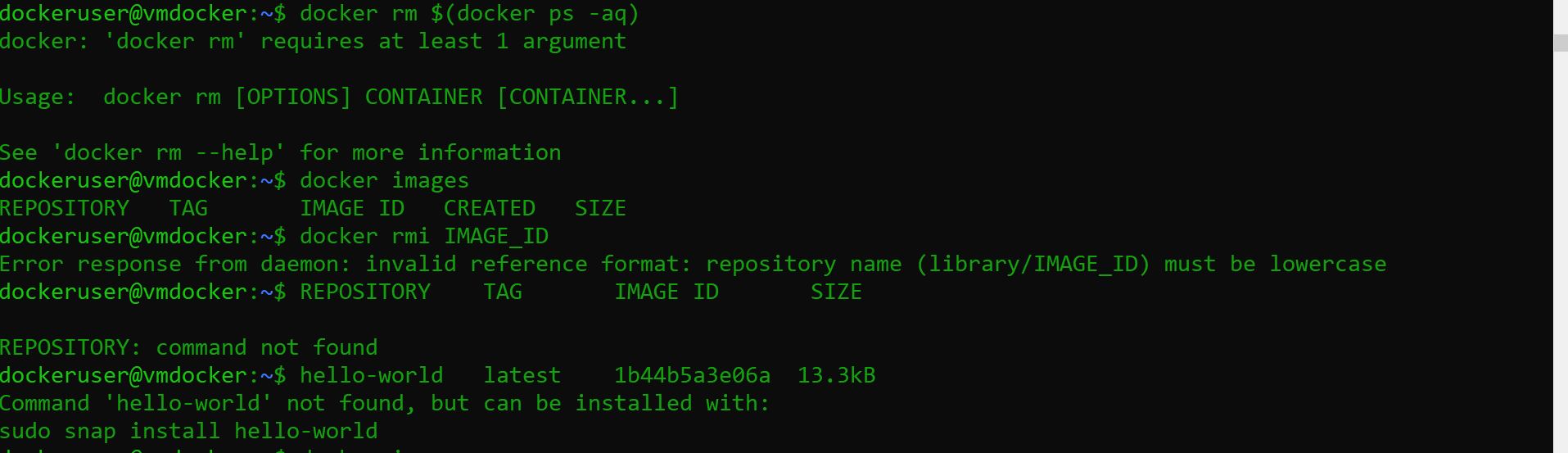
* docker ps -aq → récupère tous les IDs des conteneurs.
* $( ... ) → substitution de commande Bash : insère les résultats de docker ps -aq comme arguments.
* docker rm → supprime les conteneurs listés.

**Résultat :**

**24- - Supprimez maintenant l’image et constatez.**

docker rmi hello-world

* **Explication :**  
  L’image hello-world a déjà été supprimée précédemment, donc il n’y a plus rien à supprimer. La commande renvoie une erreur indiquant que l’image n’existe pas .

**Résultat :**

**25- Faites à nouveau exécuter un conteneur, que vous nommerez hello1, pour la même image hello-world, que se passe-t-il ?**

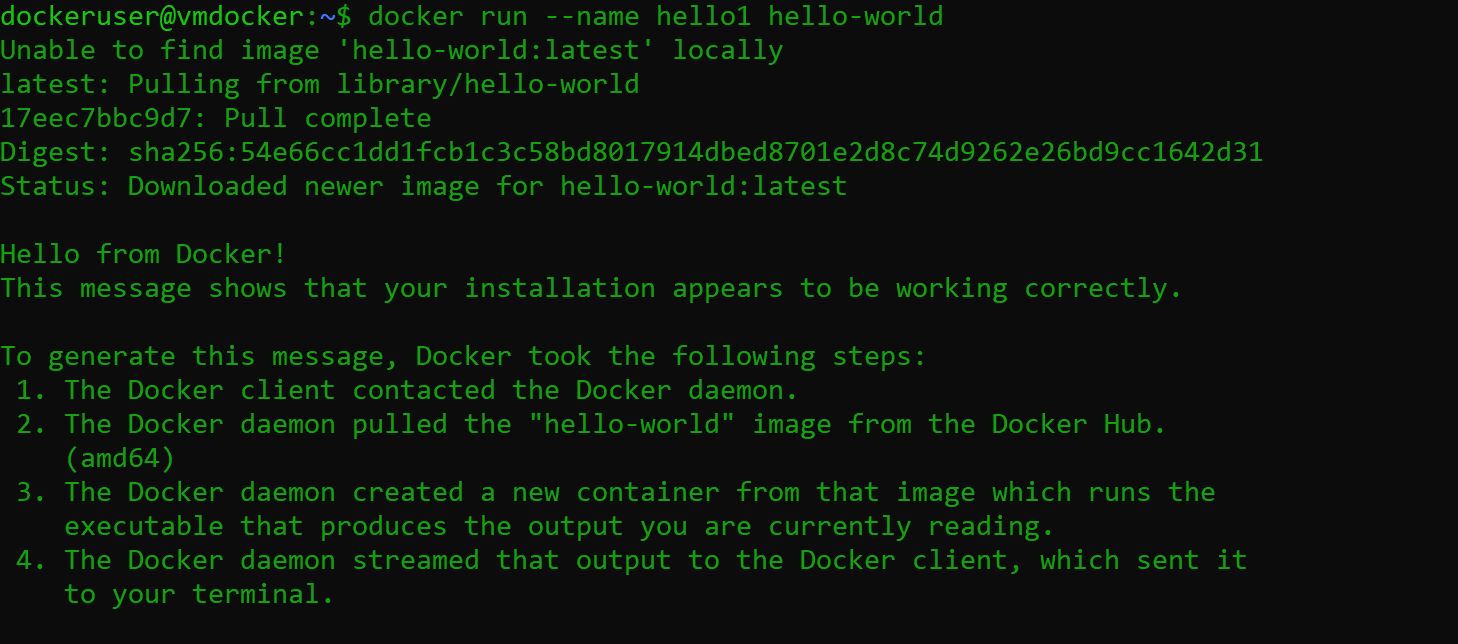
il va créer le conteneur nommé hello1 et l’exécuter car hello-world **n’est pas présente localement**

docker run --name hello1 hello-world

**Explication :**

* docker run → crée et exécute un conteneur.
* --name hello1 → donne un nom explicite au conteneur (hello1).
* hello-world → l’image à utiliser.

**Résultat :**

* 

**26- Quel est l’identifiant de l’image ? Constatez**

* Pour connaître l’ID de l’image : docker images

**Résultat :**

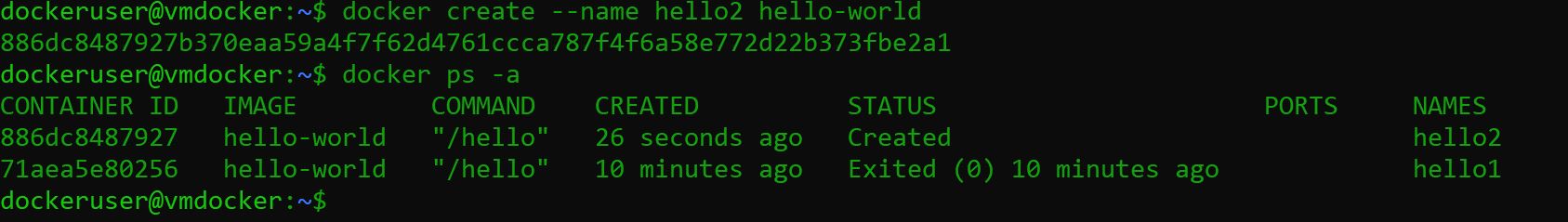


**27- Créez un conteneur, que vous nommerez hello2, mais sans le démarrer, puis observez son statut.**

* **Créer un conteneur nommé hello2 sans le démarrer et observer son statut :** docker create --name hello2 hello-world
* docker ps -a

**Explication :**

* docker create → crée un conteneur à partir d’une image **sans l’exécuter**.
* --name hello2 → nom donné au conteneur.
* hello-world → image utilisée.
* Le conteneur est créé mais **reste à l’état Created**.

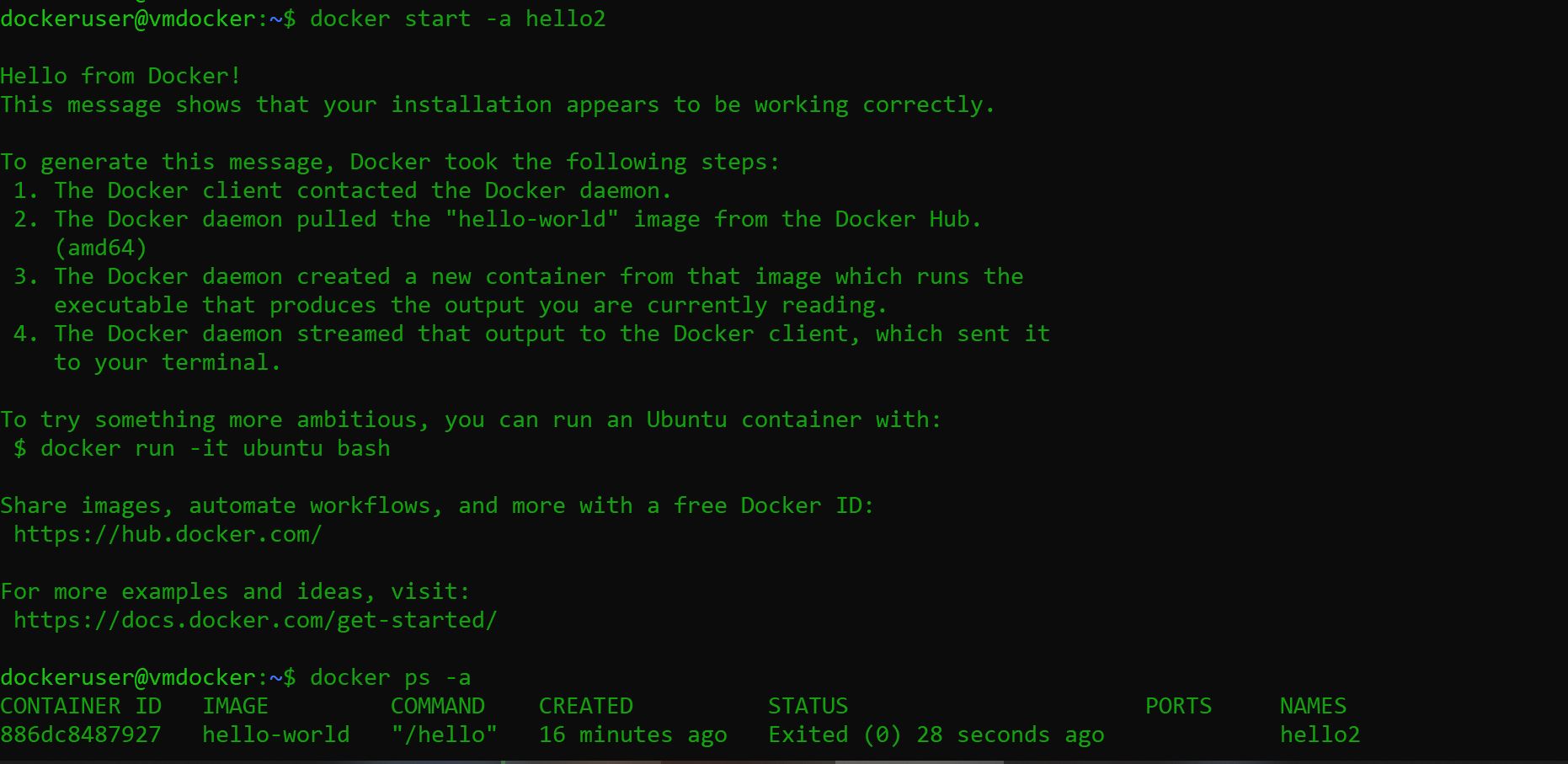
Résultat :

**28- - Démarrez-le ensuite en utilisant son nom et constatez son statut ensuite. Quel est l’affic**

* Démarrer le conteneur hello2 et observer son statut et affichage : docker start -a hello2

**Explication :**

* + docker start → démarre un conteneur existant.
  + -a (attach) → affiche directement dans le terminal la sortie du conteneur.
  + hello2 → nom du conteneur à démarrer.
  + **Ce qui se passe :**
  + Le conteneur s’exécute.
  + Comme hello-world est une image très simple, elle **affiche son message puis se termine immédiatement**.
  + Le conteneur passe ensuite à l’état Exited (0).
* docker ps -a

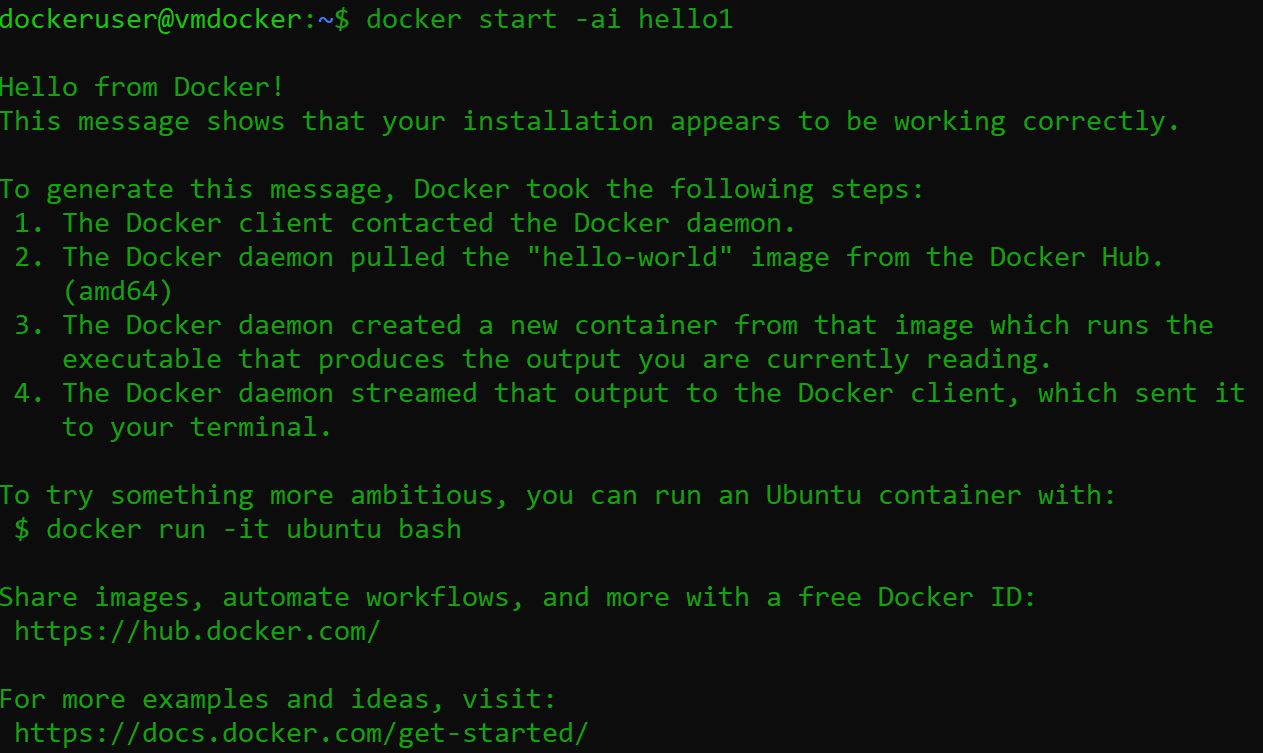
Résultat :

**29- Utilisez la même option pour démarrer le conteneur hello1.**

* docker start -ai hello1

**Explication :**

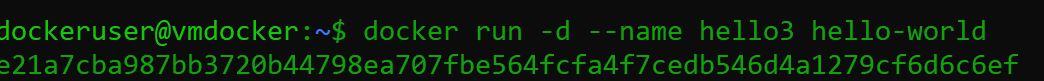
* Comme pour **hello2**, l’option -ai attache l’entrée/sortie au terminal.
* Le conteneur **hello1** exécute l’image **hello-world**, affiche son message, puis passe en état **Exited (0)**.

Résultat :

**30- Exécutez maintenant un nouveau conteneur nommé hello3, mais en faisant en sorte qu’il n’affiche pas d’information sur la sortie standard (en background donc). Constatez dans la liste (totale) des conteneurs .**

* + **Un identifiant de conteneur long s’affiche :** docker run -d --name hello3 hello-world

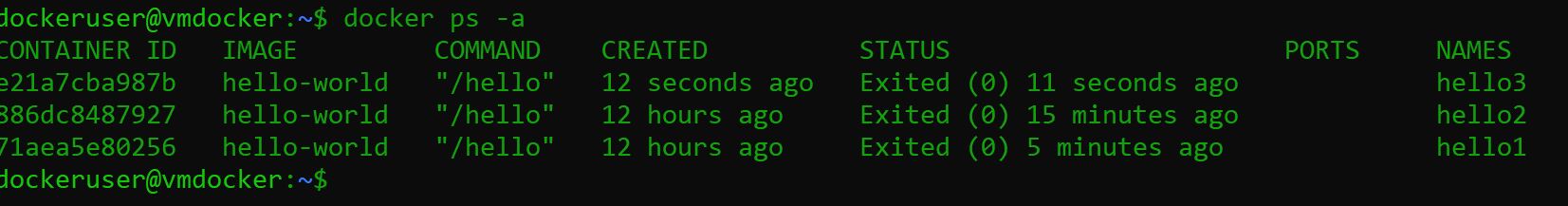
**Résultat :**



* + **Vérification :** docker ps -a

**Explication :**

* L’option **-d** signifie *detached mode* → le conteneur s’exécute en arrière-plan.
* L’image **hello-world** s’arrête immédiatement après avoir affiché son message, donc son statut sera **Exited (0)**.
* On le retrouve dans la liste des conteneurs avec le nom **hello3**.

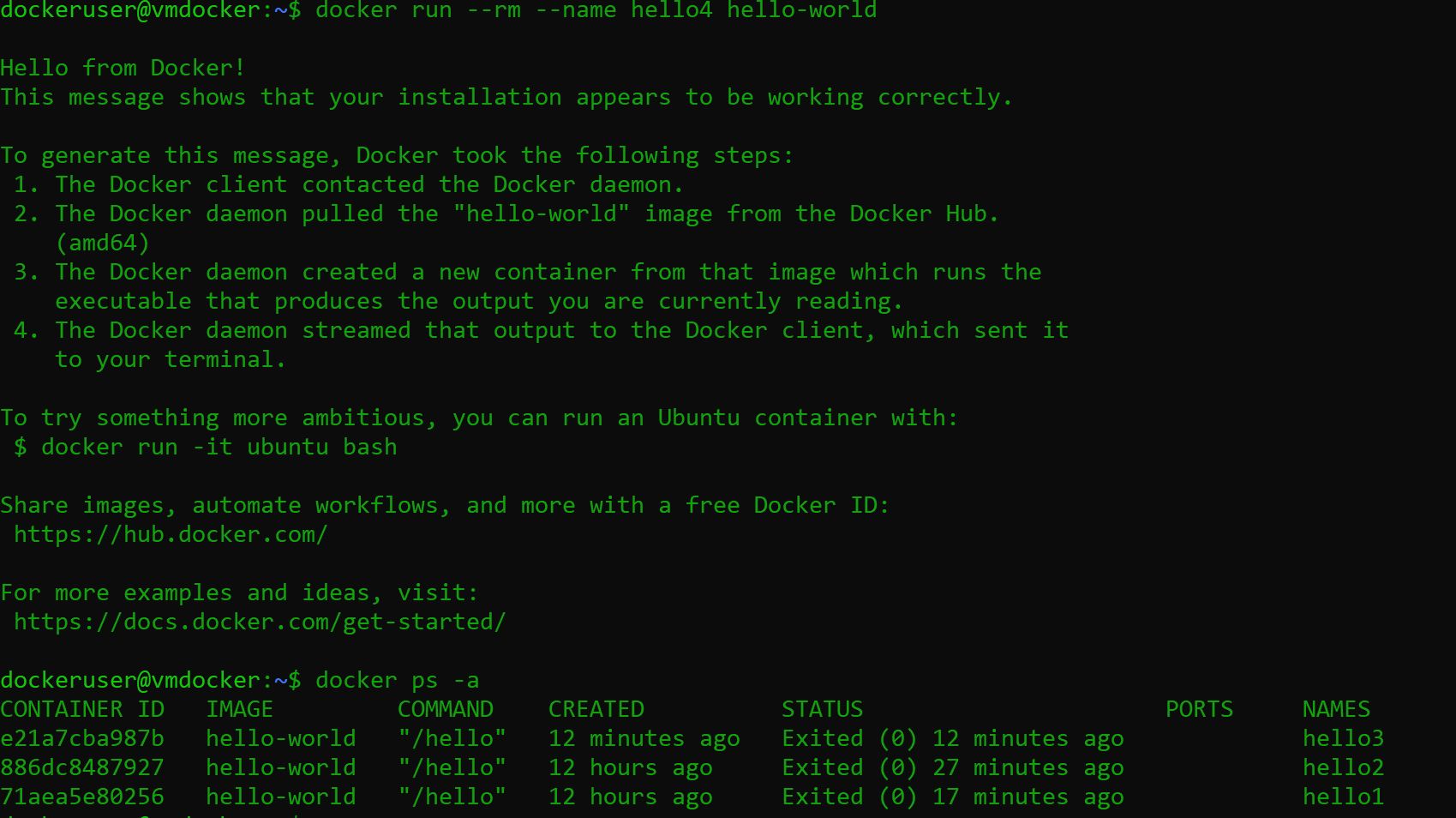
**Résultat :**

**31- Exécutez maintenant un nouveau conteneur nommé hello4, mais en faisant en sorte que ce conteneur ait totalement disparu après son exécution. Constatez dans la liste (totale) des conteneurs.**

* docker run --rm --name hello4 hello-world
* docker ps -a

**Explication :**

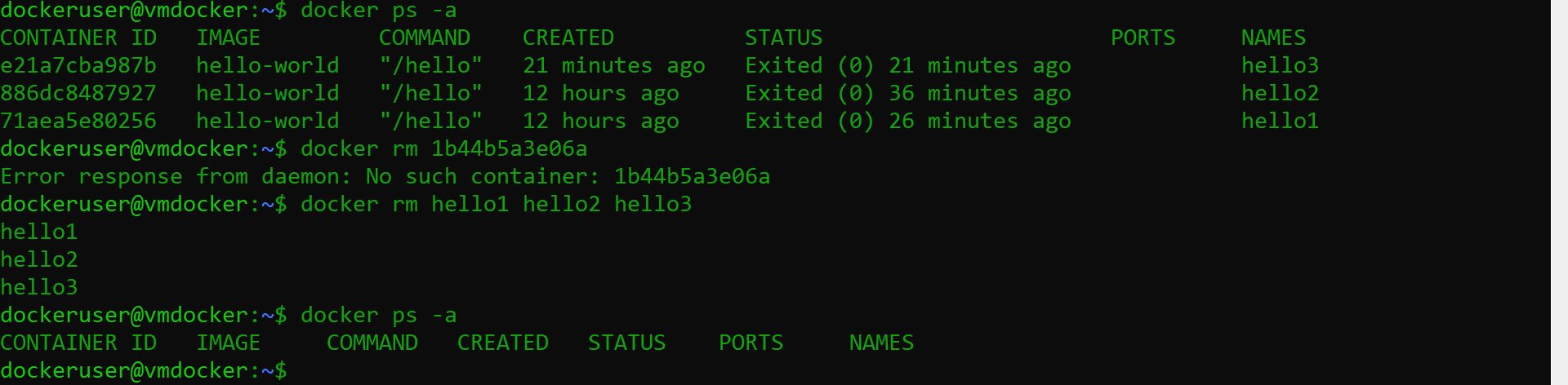
* L’option **--rm** supprime automatiquement le conteneur dès qu’il s’arrête.
* Contrairement à hello1, hello2 et hello3 qui restent visibles avec le statut **Exited**, le conteneur **hello4** n’apparaît plus du tout dans la liste.

Résultat :

**32- Vous pouvez supprimer l’image hello-world avant de passer à la partie suivante.**

* docker rm hello1 hello2 hello3
* docker ps -a

Résultat :



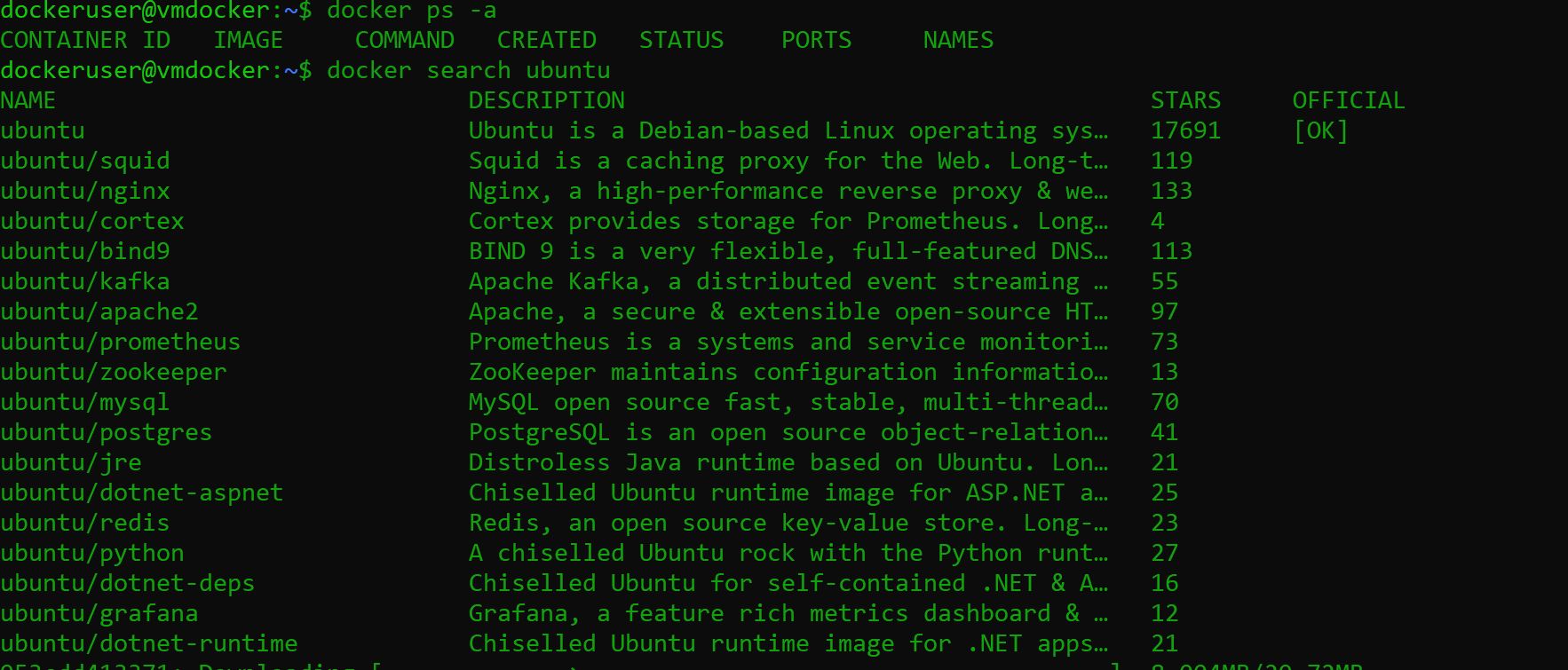
**Partie 4 – DockerHub**

**1-Types d’images sur DockerHub et classification**

|  |  |
| --- | --- |
| *Type d’image* | *Description* |
| Official Images | Maintenues par docker et éditeur officiel(nginx ,ubuntu……) |
| Verified Publisher Images | Créées par des éditeurs ou organisations de confiance, avec badge de vérification. |
| Community Images | Créées par la communauté Docker, moins garanties. |
| |  | | --- | | **Private Images** |  |  | | --- | |  | | |  | | --- | |  |  |  | | --- | | Créées par un utilisateur et visibles uniquement pour lui . | |

* **Retrouver l’image officielle Ubuntu** 🡪**Commande Docker pour rechercher :** docker search ubuntu

**Résultat :**

****

**2-Téléchargement de l’image officielle :** docker pull ubuntu

Résultat :

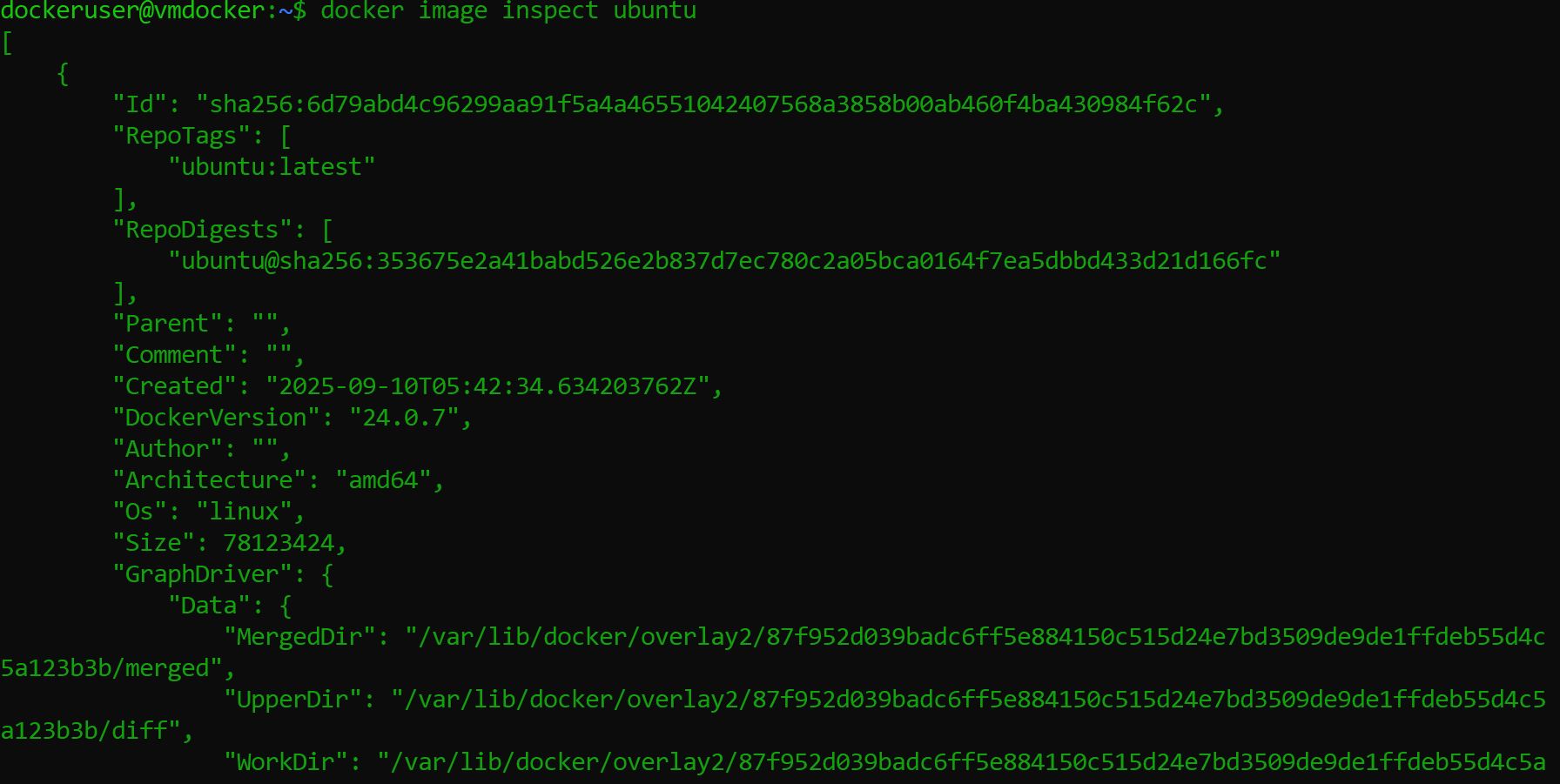


**3- Consultez les différentes versions proposées, comment les distingue-t-on ?**

* **Pour voir les versions disponibles d’une image (par exemple Ubuntu) :**docker image inspect ubuntu

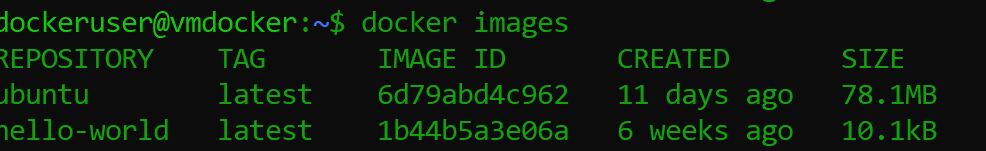
**Explication :**

* Le **tag** latest indique que c’est la **dernière version stable** téléchargée.
* L’attribut "org.opencontainers.image.version": "24.04" précise la **version exacte de l’image**, ici Ubuntu 24.04.
* "Architecture": "amd64" → l’architecture CPU de l’image.
* "Os": "linux" → le système d’exploitation utilisé.
* "Size": 78123424 → taille de l’image en octets.
* "GraphDriver" et "RootFS" → localisation et couches de fichiers de Docker pour cette image.

Résultat :

**4- Quelle est la version la plus récente ? Quelle différence a-t-elle avec la version latest ? Quels sont leurs identifiants respectifs ?**

* Docker images

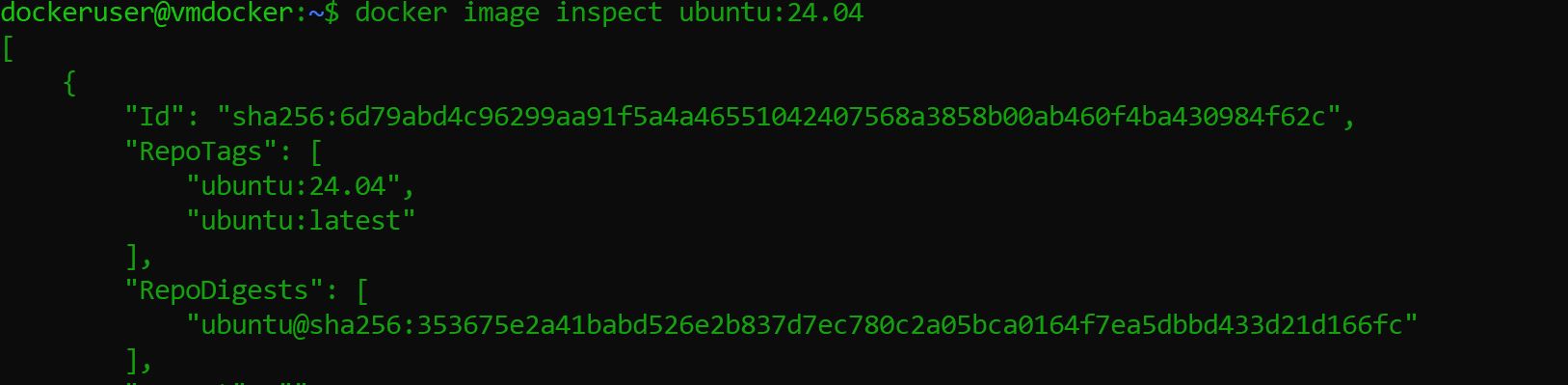
Résultat :

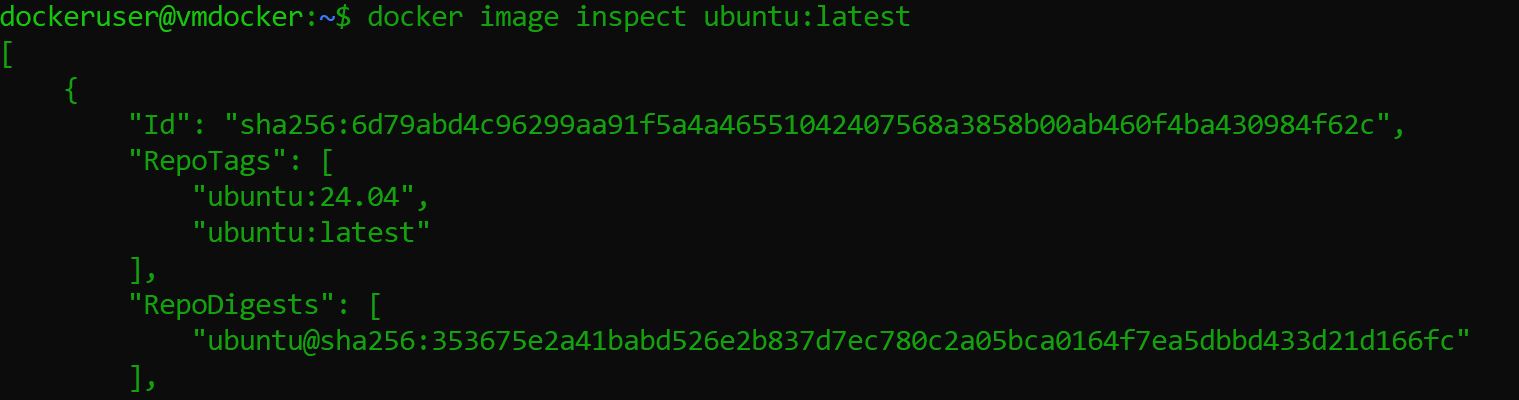
**5- Quelle est la version la plus récente ? Quelle différence a-t-elle avec la version latest ? Quels sont leurs identifiants respectifs ?**

* La version la plus récente est : Ubuntu 24.04

 diffrence avec la version latest :

* **24.04** = une version précise (tag numérique fixé).
* **latest** = un tag générique qui pointe en général vers la version stable par défaut, mais qui ne correspond pas toujours à la version numérique la plus récente.
* Les identifiants respectifs :

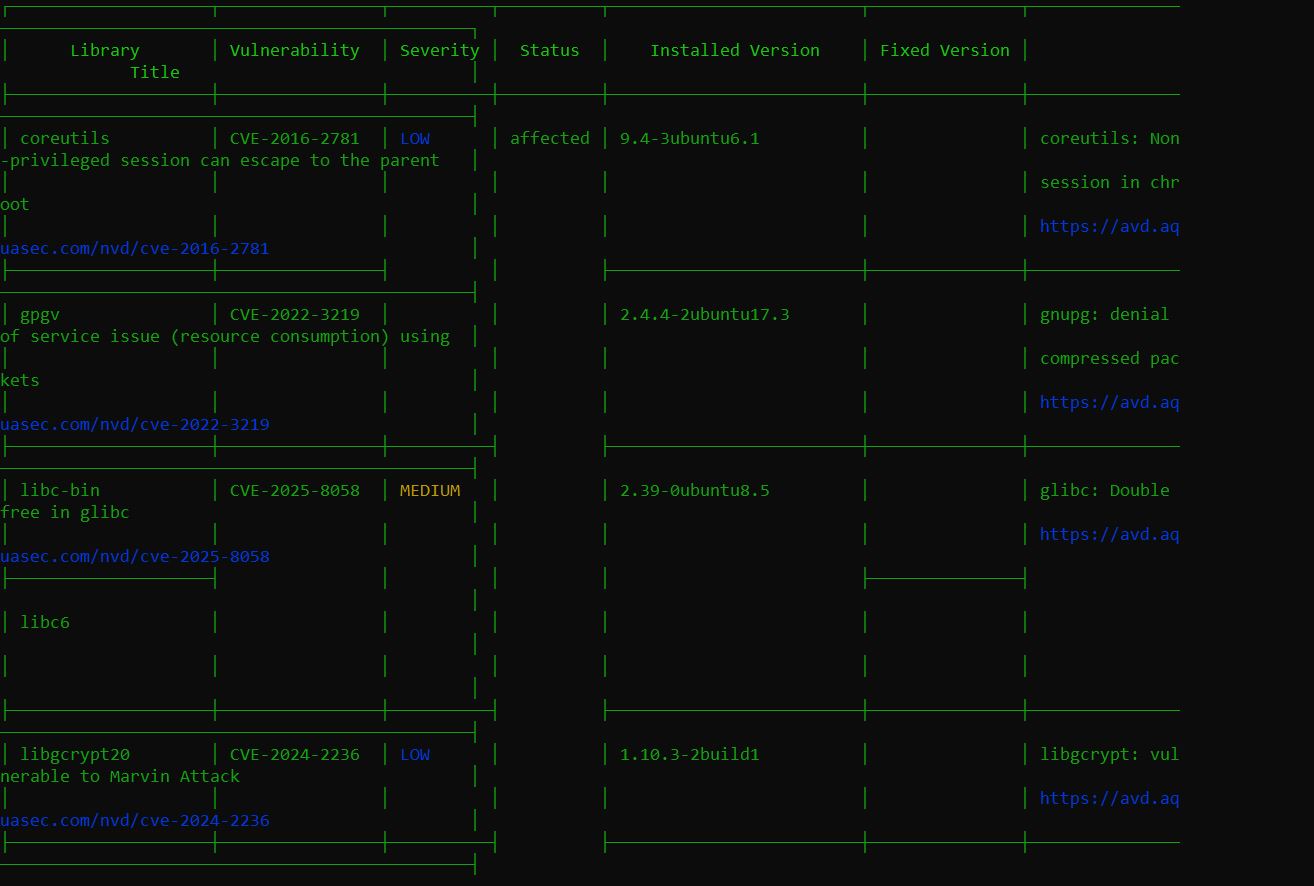
**1-Ubuntu 24.04** → docker image inspect ubuntu:24.04

**2- Ubuntu:latest** → docker image inspect ubuntu:latest ****

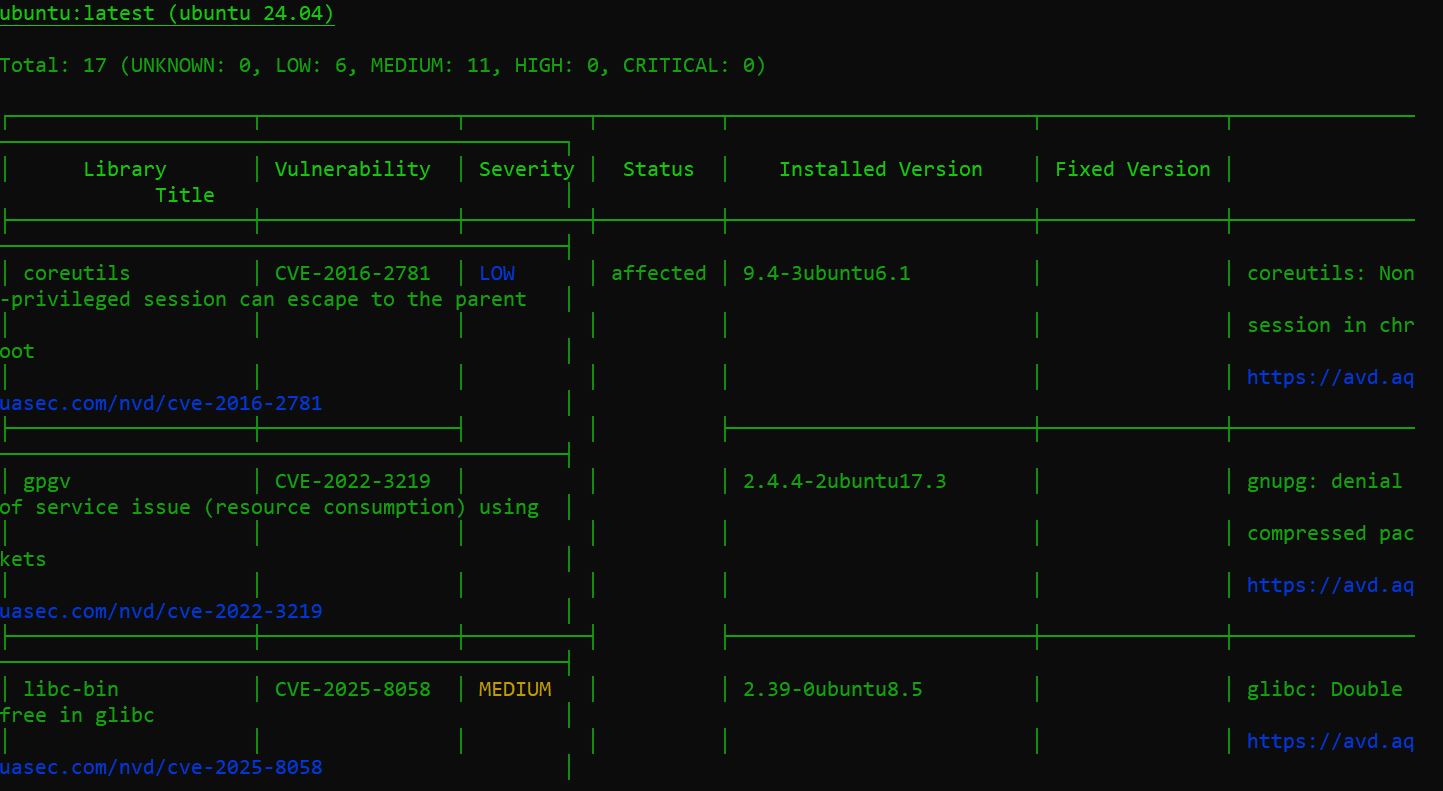
**6- Observez comment ces vulnérabilités sont classées.**

* **test de sécurité ubuntu :24.04 🡪** trivy image ubuntu:24.04

**Résultat :**

****

* **ubuntu :latest** 🡪 trivy image ubuntu:latest

**Résultat :**

**7- Comparez la version latest avec la version noble : quel pourrait être l’intérêt de cette situation ?**

* **latest :** pointe vers la dernière version publiée.
* **24.04 :** est une version fixe et spécifique.

**Explication :**

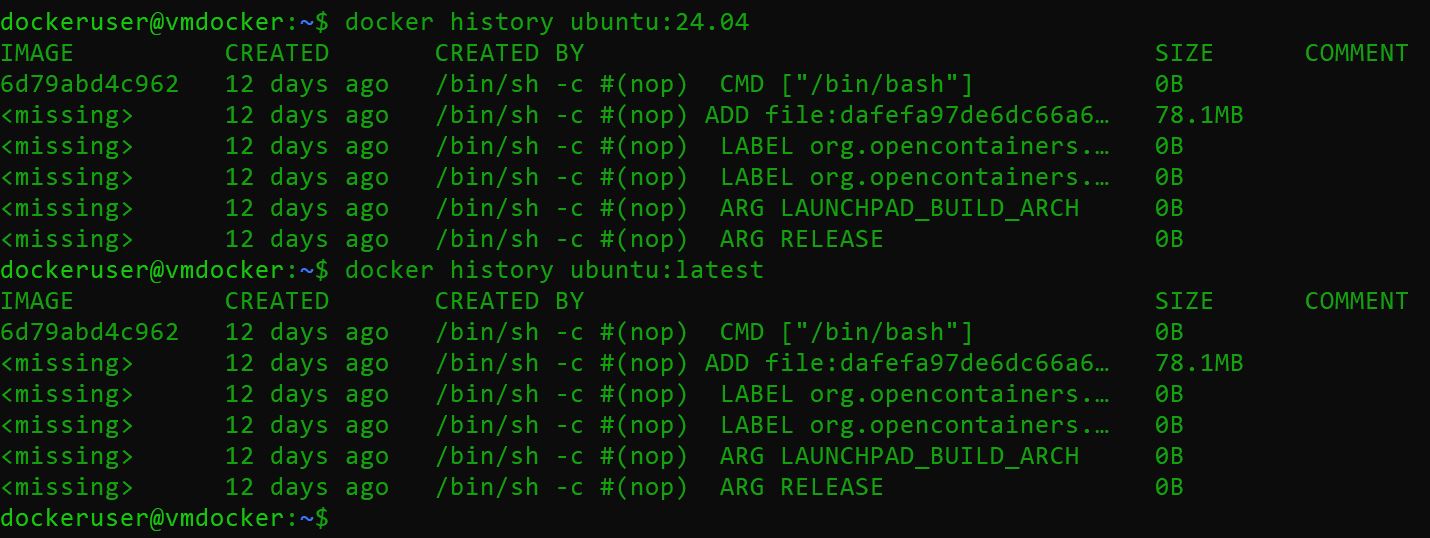
* **Latest :** est pratique pour récupérer automatiquement les dernières mises à jour.
* **24.04 :** permet de garantir la stabilité et la reproductibilité d’un environnement, utile pour la production ou un projet où l’on ne veut pas que l’image change.

**8-**  **En cliquant sur le tag de chacune des 2, consultez les couches de ces images, et observez à partir de quand elles diffèrent.**

* docker history ubuntu:latest
* docker history ubuntu:24.04

**Explication :**

* Cela permet de voir où latest a été modifiée par rapport à la version stable 24.04.
* Comprendre les couches est utile pour identifier les changements dans l’image sans avoir à reconstruire entièrement la base.

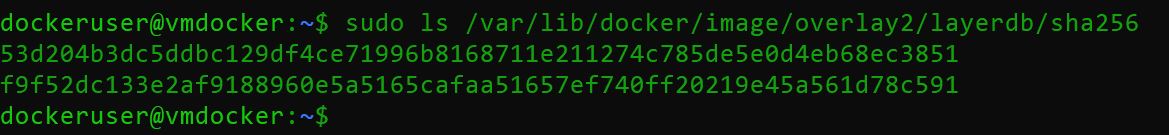
**Résultat :**

**9- Placez-vous dans le répertoire /var/lib/docker/image/overlay2/layerdb/sha256 et listez son contenu.**

* sudo ls /var/lib/docker/image/overlay2/layerdb/sha256

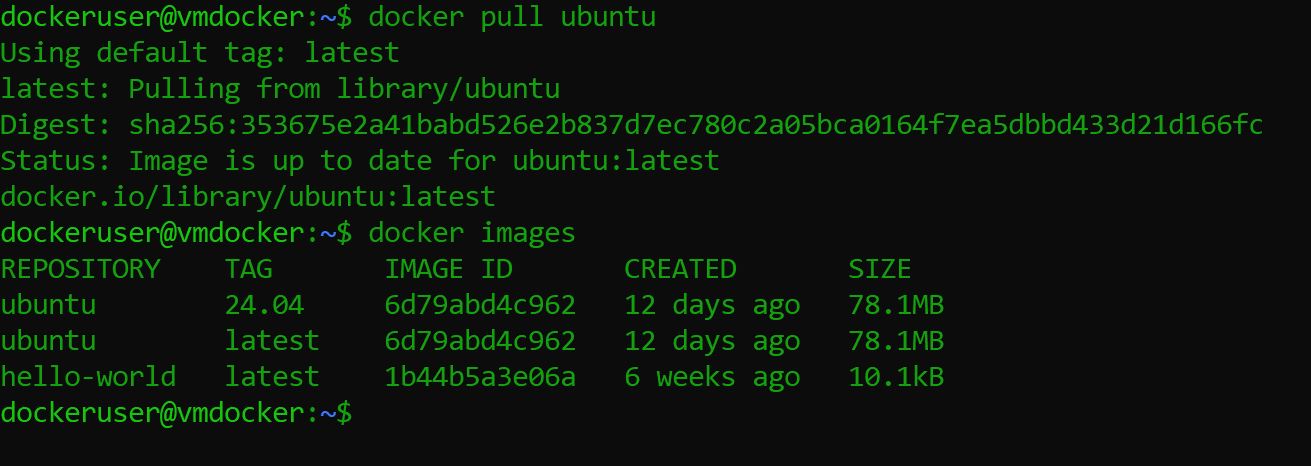
Explication :

* + - utilise ces couches pour **réutiliser des parties d’images entre conteneurs**, et accélérant la création de nouveaux conteneurs.
    - Ces fichiers ne sont pas les images elles-mêmes

**Résultat :**

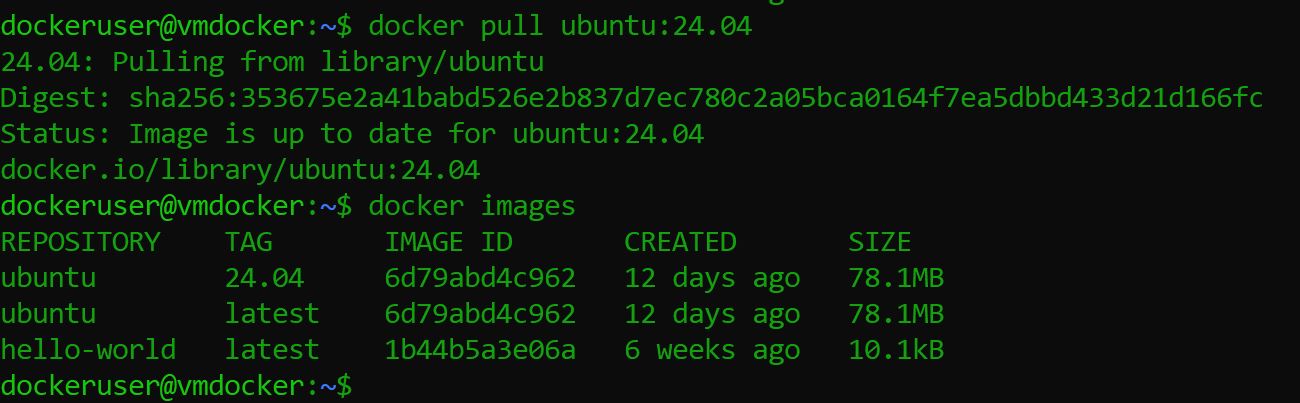
**10- Téléchargez l’image ubuntu : par défaut, quelle est celle qui est téléchargée.**

* Docker pull ubuntu
* Docker images

**Résultat :**

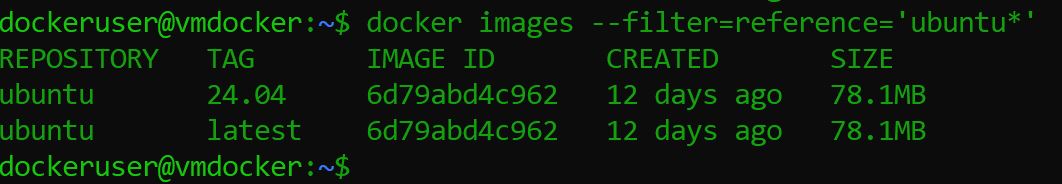
**11** **- Téléchargez maintenant l’image ubuntu la plus récente.**

* Docker pull ubutnu :24.04
* Docker images

Résultat :

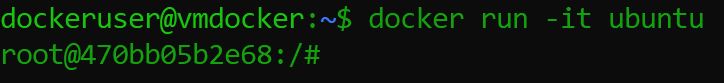
**12- Utilisez un filtre pour n’afficher que les images référençant ubuntu.**

* docker images --filter=reference='ubuntu\*'

**Résultat  :**

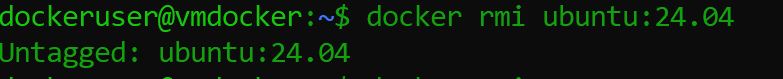
**13 - Identifiez la commande qui est lancée lorsqu’on exécute un conteneur à partir de l’image ubuntu.**

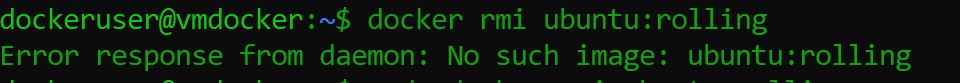
* Docker run -it ubuntu
* L’option -it permet de lancer le conteneur en mode interactif avec un terminal attaché.

**Résultat :**

**14- Pour la suite, vous pouvez supprimer l’image ubuntu:rolling.**

* Docker rmi ubuntu :24.04

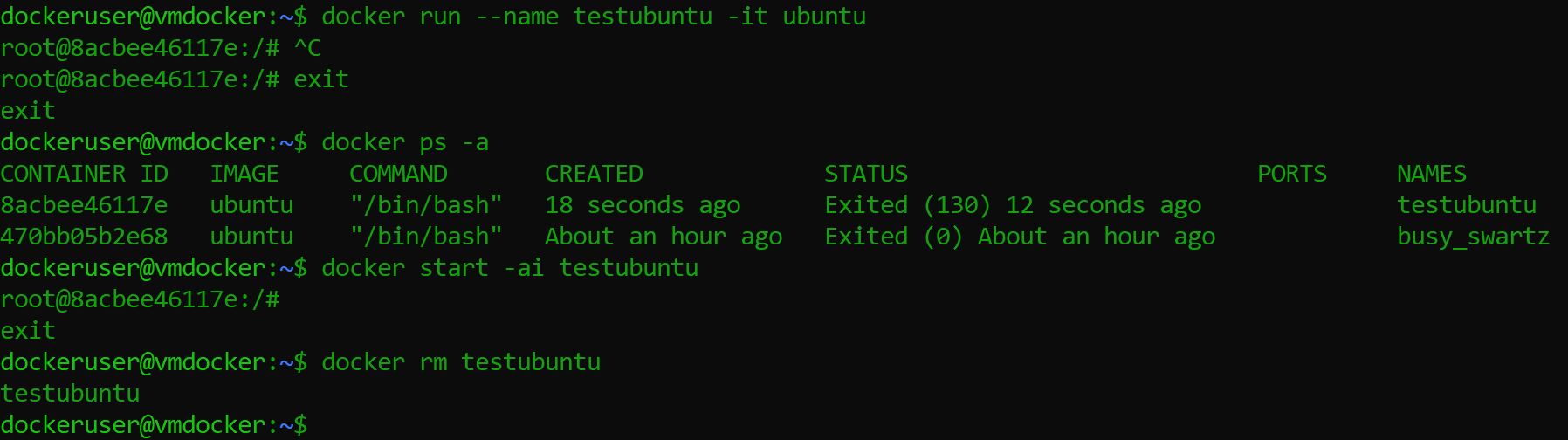
**Résultat :**



**Partie 5 – Interagir avec un conteneur**

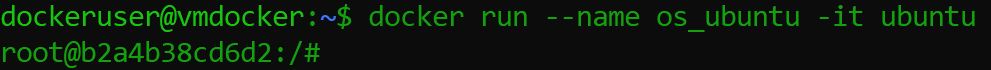
**1-** **Lancez un conteneur à partir de l’image ubuntu et constatez son statut. Essayez de redémarrer ce même conteneur en interactif. Enfin supprimez le.**

* **un nouveau conteneur nommé testubuntu🡪**docker run --name testubuntu -it ubuntu
* **liste tous les conteneur 🡪** docker ps -a
* **redémarre le même conteneur et ouvre une session interactive à l’intérieur🡪** docker start -ai testubuntu
* **supprime le conteneur🡪** docker rm testubuntu

**Résultat :**

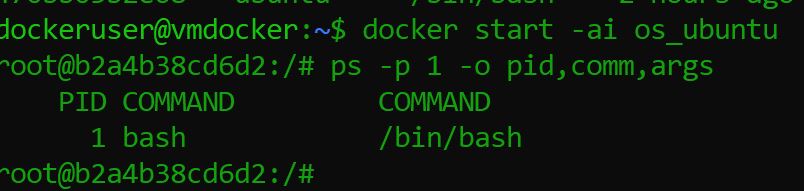
1. **Lancez, à partir de l’image ubuntu, un conteneur nommé os\_ubuntu en interactif et attaché à un terminal.**

* **nouveau conteneur à partir de l’image Ubuntu avec le nom os\_ubuntu, en mode interactif (-it) et attaché au terminal :** docker run --name os\_ubuntu -it ubuntu

**Résultat :**

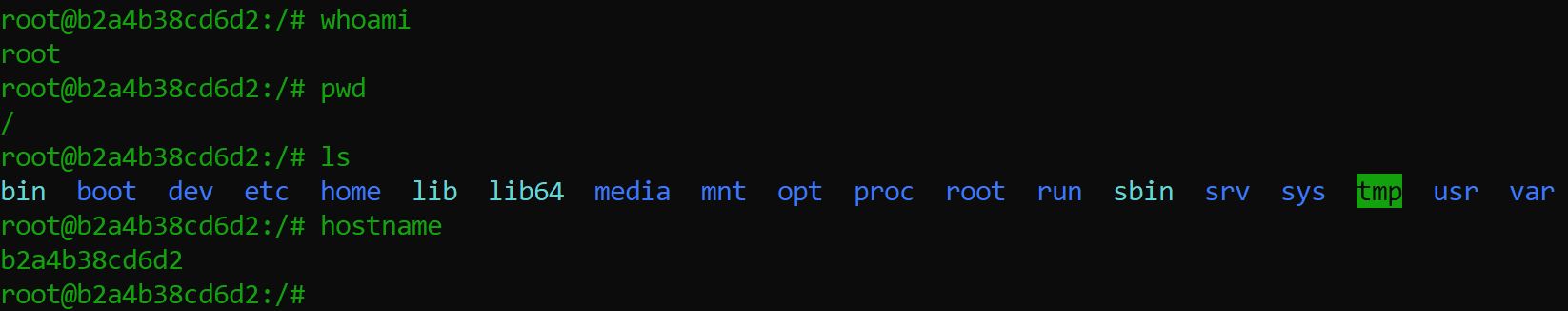
1. **Quelle est la commande qui correspond au processus de PID 1 de ce conteneur ?**

* **dans le conteneur affichez le processus PID 1🡪** docker start -ai os\_ubuntu🡪**(root)** ps -p 1 -o pid,comm,args

**Résultat :**

1. **Dans cet OS, exécutez les commandes whoami, pwd, ls et hostname. Notez son ID.**

* Affiche root**, donc vous êtes connecté en tant que super utilisateur 🡪** **whoami**
* Le répertoire est /**, c’est la racine du système de fichiers 🡪pwd**
* liste le contenu **du répertoire racine 🡪** ls
* affiche l’ID **du conteneur 🡪**hostname

**Résultat :**

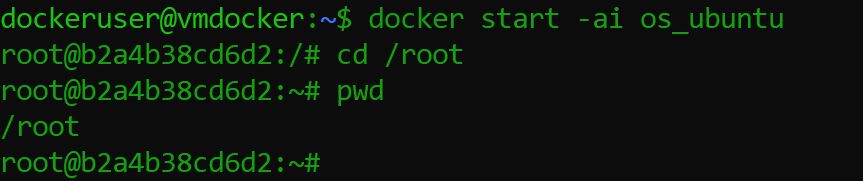
**5- Ouvrez un deuxième terminal, connectez-vous en ssh sur la VM et observez le statut du conteneur en cours d’exécution. Que constatez-vous (à part le statut) ?**

* docker ps

**Résultat :**

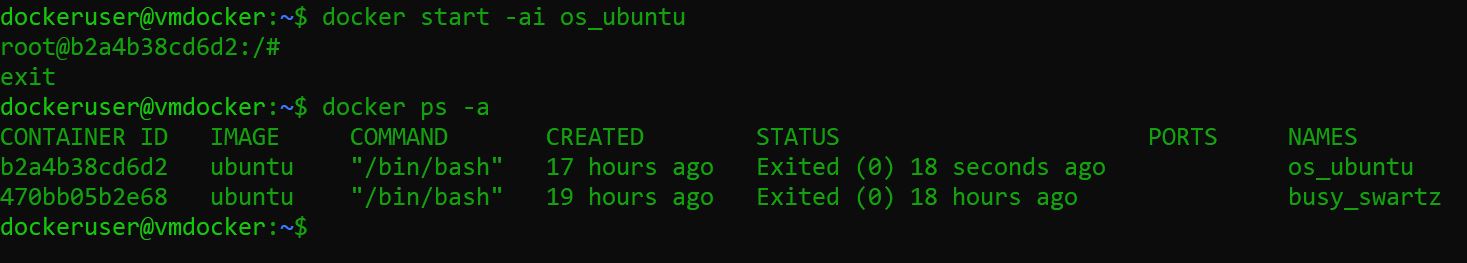
1. **Revenez dans le conteneur sur le premier terminal. Déplacez-vous dans le répertoire home.**

* **docker start -ai os\_ubuntu**
  + **cd /root**
  + **pwd**

**Résultat :**

1. **Quittez ce conteneur avec la commande unix classique puis observez le statut du conteneur.**

* Exit
* Docker ps -a

**Résultat :**

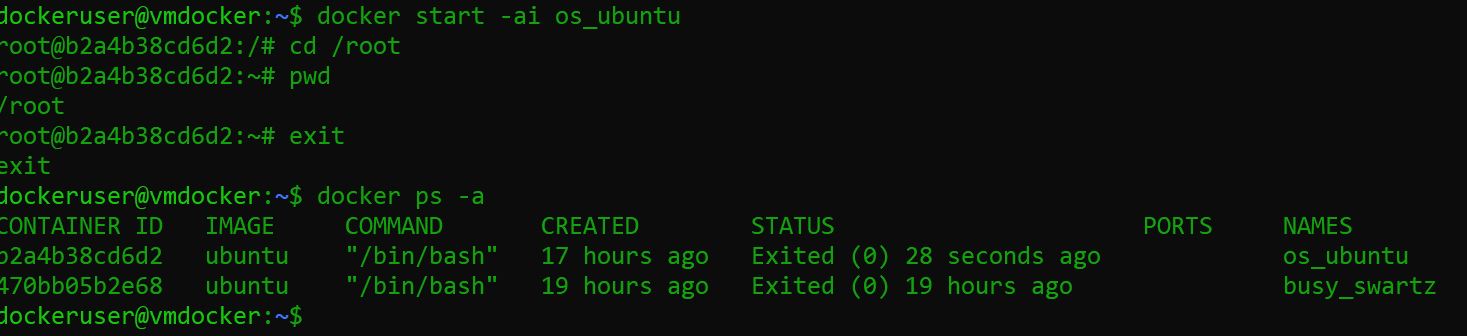
1. **Démarrez ce conteneur en interactif. Dans quel répertoire vous trouvez-vous ? Pourquoi ?**

* docker start -ai os\_ubuntu
  + - **cd /root**
    - **pwd**
    - **exit**
* **docker ps -a**

**Pourquoi ?**

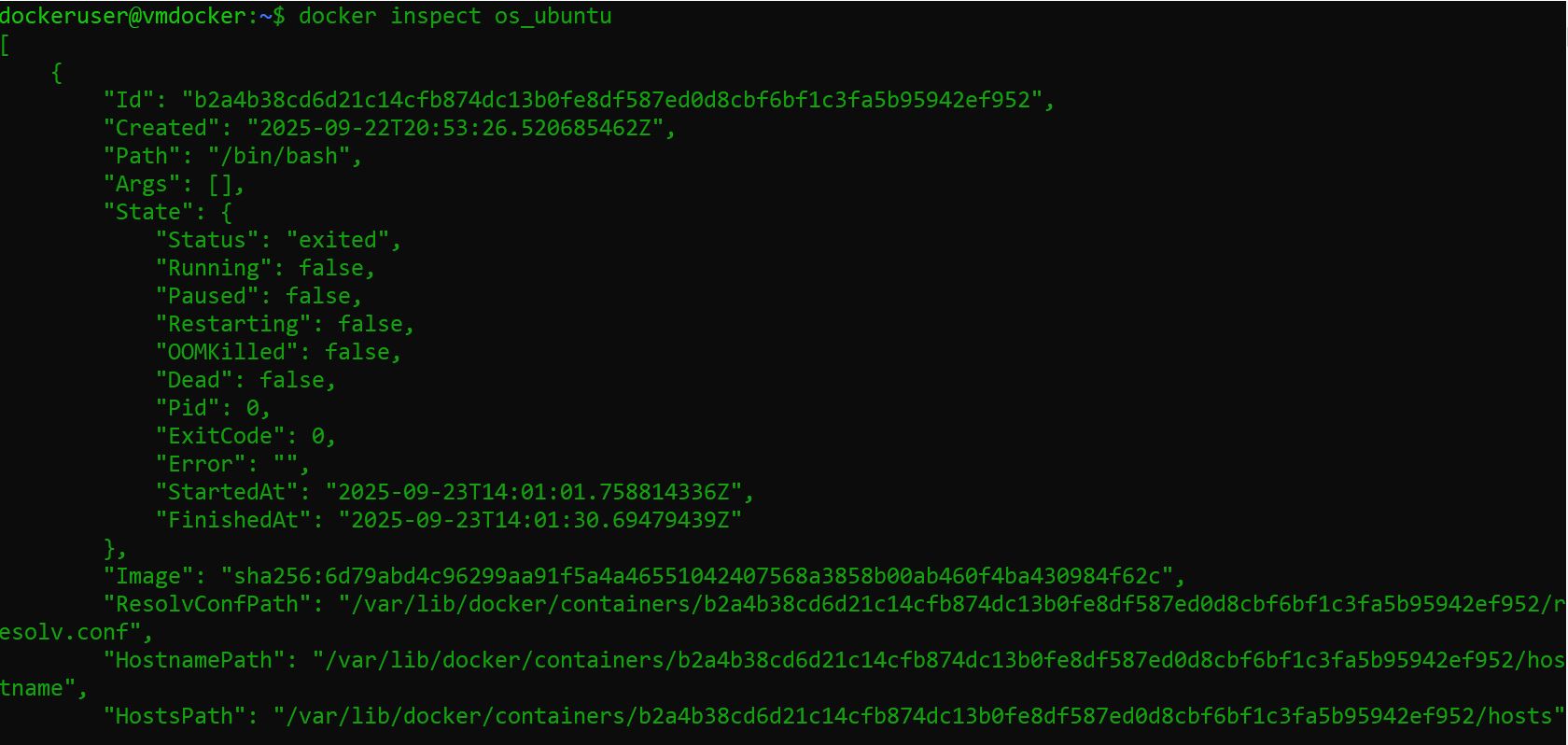
Parce que par défaut, l’image Ubuntu n’a pas de commande personnalisée ni de répertoire de travail configuré.

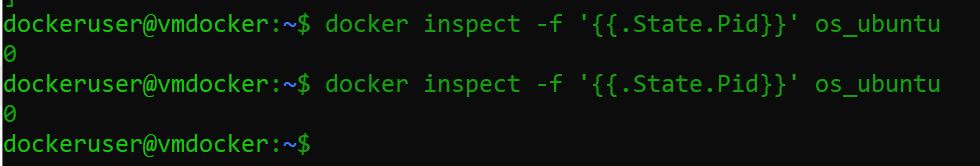
* Le conteneur exécute /bin/bash comme processus PID 1.
* Le répertoire courant est donc défini par défaut sur /.

**Résultat :**

1. **Dans le second terminal, utilisez une commande qui permet d’inspecter le conteneur. Constatez qu’il est en cours d’exécution. Retrouvez son Pid.**

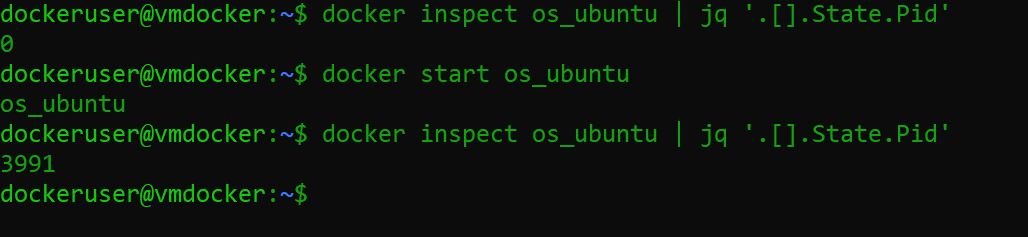
* **inspecter conteneur🡪** docker inspect os\_ubuntu
* **uniquement le PID 🡪** docker inspect -f '{{.State.Pid}}' os\_ubuntu

**Résultat** :****



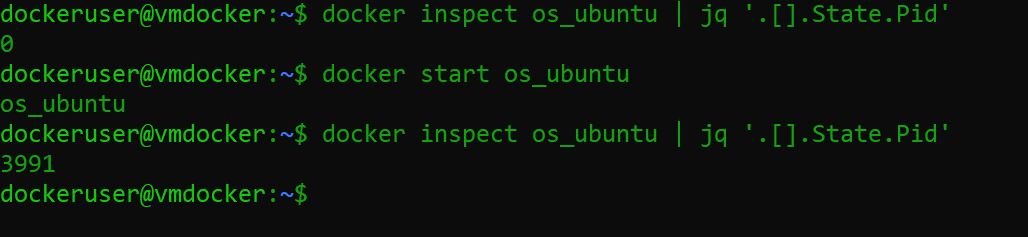
1. **Vous pouvez aussi essayer d’utiliser jq pour obtenir cette information, par exemple voir : https://blog.madrzejewski.com/jq-traiter-parser-json-shell-cli/ (ou tout lien de votre choix).**

* **Pour obtenir le PID du conteneur 🡪** docker inspect os\_ubuntu | jq '.[].State.Pid'

**Résultat :**

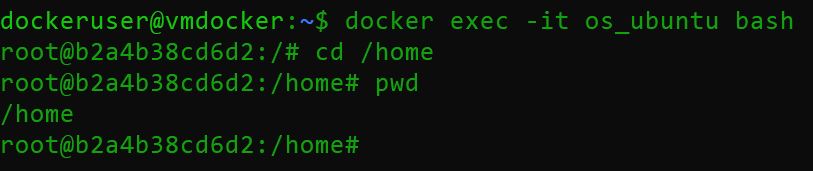
**10-Dans ce même terminal, retrouvez le processus dont le PID est celui que vous venez d’identifier et constatez.**

* **voir le processus sur le système 🡪** ps -p 3991 -f

**Résultat :**

1. **Revenez dans le conteneur (dans le premier terminal) et déplacez-vous à nouveau dans home.**

* **Démarrer le conteneur 🡪**docker start os\_ubuntu
* **Revenir dans le conteneur 🡪** docker exec -it os\_ubuntu bash
* **se déplacer dans le répertoire /home :** cd /home
* **vérifie dans /home :**pwd

**Résultat** **:**

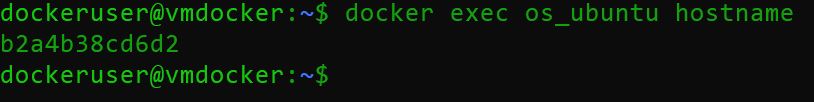
1. **Quittez maintenant ce conteneur en utilisant la combinaison de touches Ctrl-p Ctrl-q. Quel est maintenant le statut du conteneur ?**

* docker ps -a

**Résultat :**

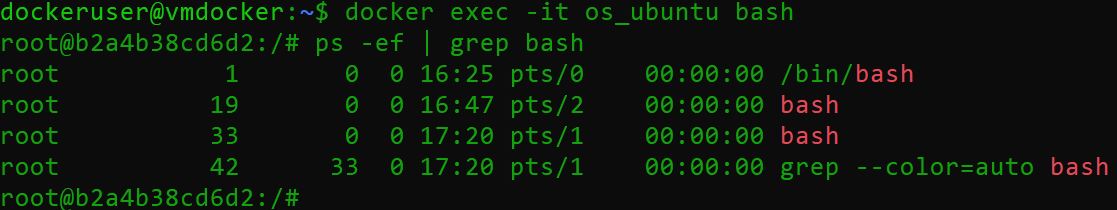
1. **Utilisez une commande docker pour exécuter la commande Unix hostname dans ce conteneur (sans le passer en foreground)**

* docker exec os\_ubuntu hostname

**Résultat :**

1. **Utilisez une commande docker pour exécuter la commande Unix bash en interactif dans ce conteneur. Vérifiez le nombre de processus bash qui tournent dans le conteneur.**

* **Lancer un shell Bash interactif dans le conteneur :** docker exec -it os\_ubuntu bash
* ps -ef | grep bash

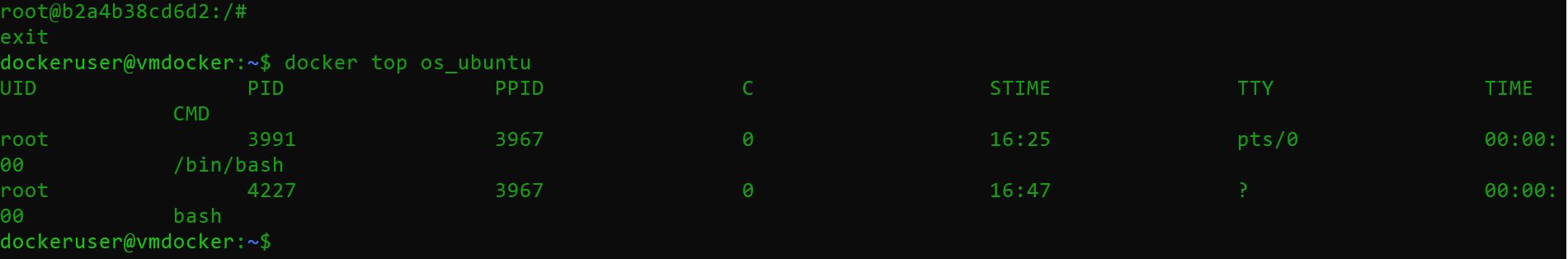
**Résultat :**

1. **Quittez à nouveau ce conteneur sans l’arrêter, puis utilisez une commande docker qui affiche les processus du conteneur.**

* **voir tous les processus qui tournent dans le conteneur 🡪**docker top os\_ubuntu

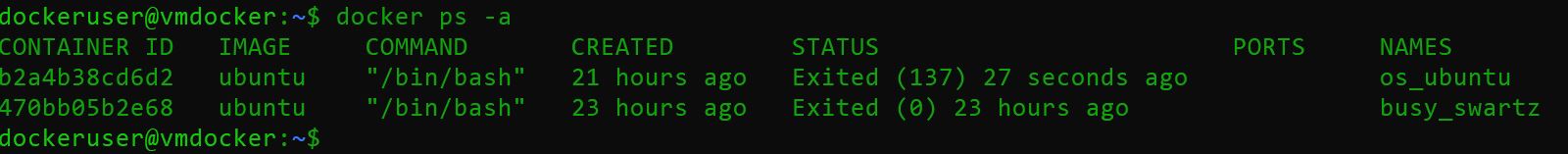
**Explication :**

* docker top <nom\_conteneur> → affiche tous les processus en cours dans ce conteneur.
* Tu verras les PID, l’utilisateur, la commande exécutée et d’autres infos, similaire à la commande ps dans Linux.

**Résultat :**

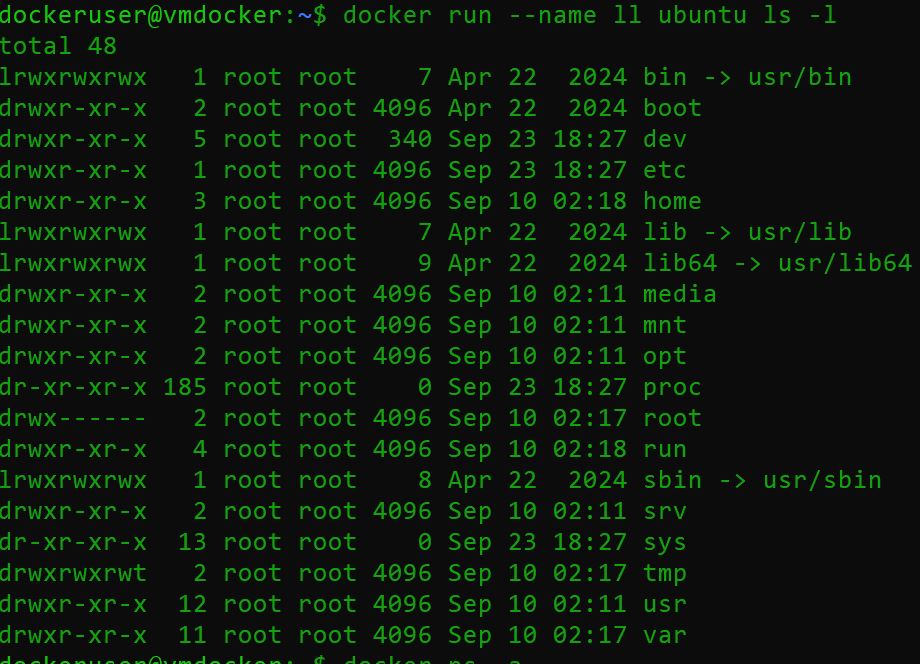
1. **Revenez dans le conteneur et arrêtez-le en le quittant.**

* docker stop os\_ubuntu
* docker ps -a

**Résultat :**

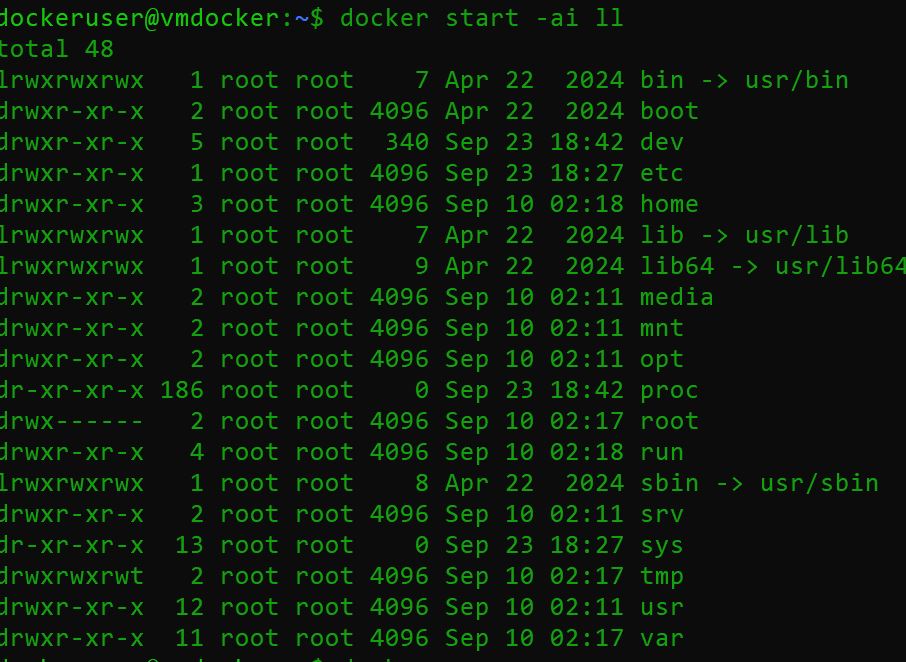
1. **Lancez maintenant un nouveau conteneur nommé ll à partir de l’image ubuntu dont la commande est maintenant ls -l.**

* docker run --name ll ubuntu ls -l

**Résultat :**

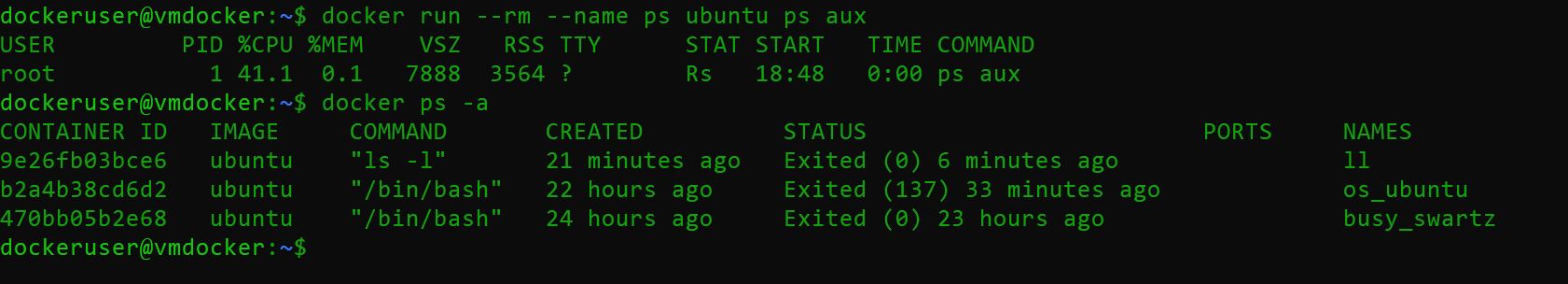
1. **Redémarrez ce conteneur pour obtenir le même affichage.**

* docker start -ai ll

**Résultat :**

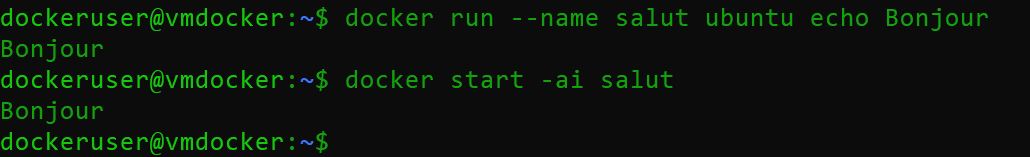
1. **Lancez un nouveau conteneur nommé ps avec la commande ps aux, mais en faisant en sorte que ce conteneur disparaisse après son exécution. Constatez le PID et constatez le statut**.

* **Lancer un conteneur éphémère pour exécuter ps aux🡪** docker run --rm --name ps ubuntu ps aux
* docker ps -a
* **Explication :**
  + docker run → crée et lance un conteneur.
  + --rm → **supprime automatiquement** le conteneur après l’exécution de la commande.
  + --name ps → nom du conteneur pour le repérer pendant l’exécution.
  + ubuntu → image utilisée.
  + ps aux → commande Unix pour lister tous les processus du conteneur.

**Résultat :**

1. **Lancez un nouveau conteneur nommé salut avec la commande echo Bonjour, puis relancez ce conteneur.**

* **un conteneur nommé salut avec la commande echo Bonjour 🡪** docker run --name salut ubuntu echo Bonjour
* **Relancer le conteneur pour voir la même sortie🡪** docker start -ai salut

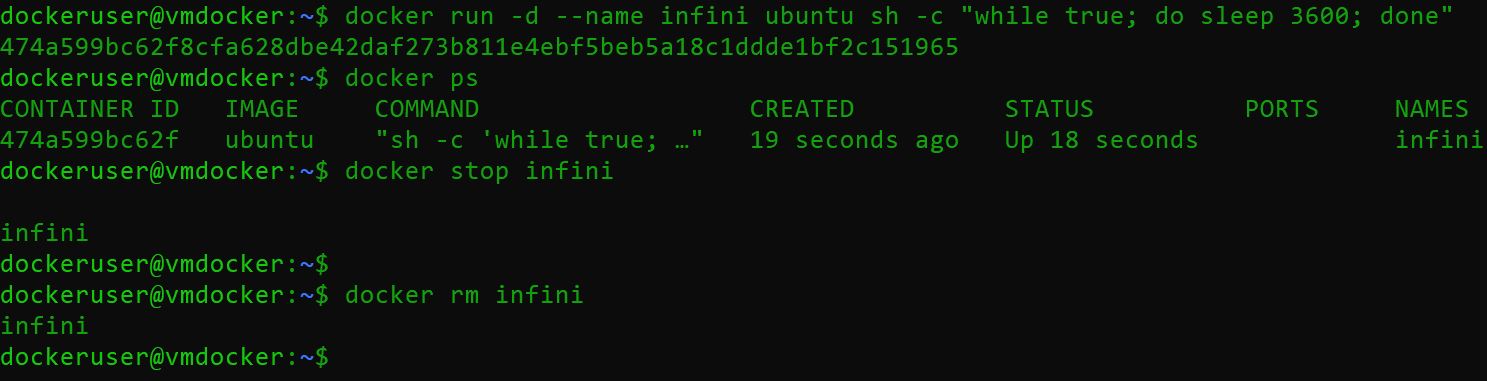
**Résultat :**

1. **Lancez un nouveau conteneur avec une commande infinie (sh -c "while true ; sleep 3600 ; done"), puis faites en sorte de le supprimer.**

* **Infinite 🡪** docker run -d --name infini ubuntu sh -c "while true; do sleep 3600; done"
* docker ps
* arrête-le 🡪docker stop infini
* supprime-le 🡪docker rm infini

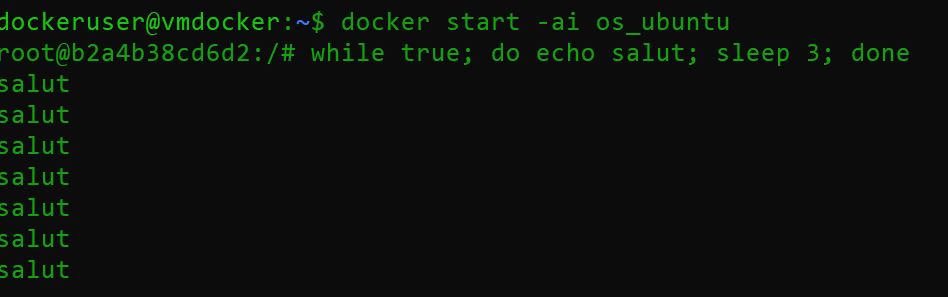
**Explication :**

* -d → lance le conteneur en arrière-plan (détaché).
* --name infini → donne le nom infini au conteneur.
* ubuntu → image de base.
* sh -c "while true; do sleep 3600; done" → boucle infinie qui dort 1h à chaque tour, donc le conteneur reste actif indéfiniment.

**Résultat** :

1. **Démarrez le conteneur os\_ubuntu en interactif. Dans ce shell, lancez une commande qui affiche salut toutes les 3 secondes.**

* **Démarrer le conteneur 🡪** docker start -ai os\_ubuntu
* **LOOP🡪** while true; do echo salut; sleep 3; done

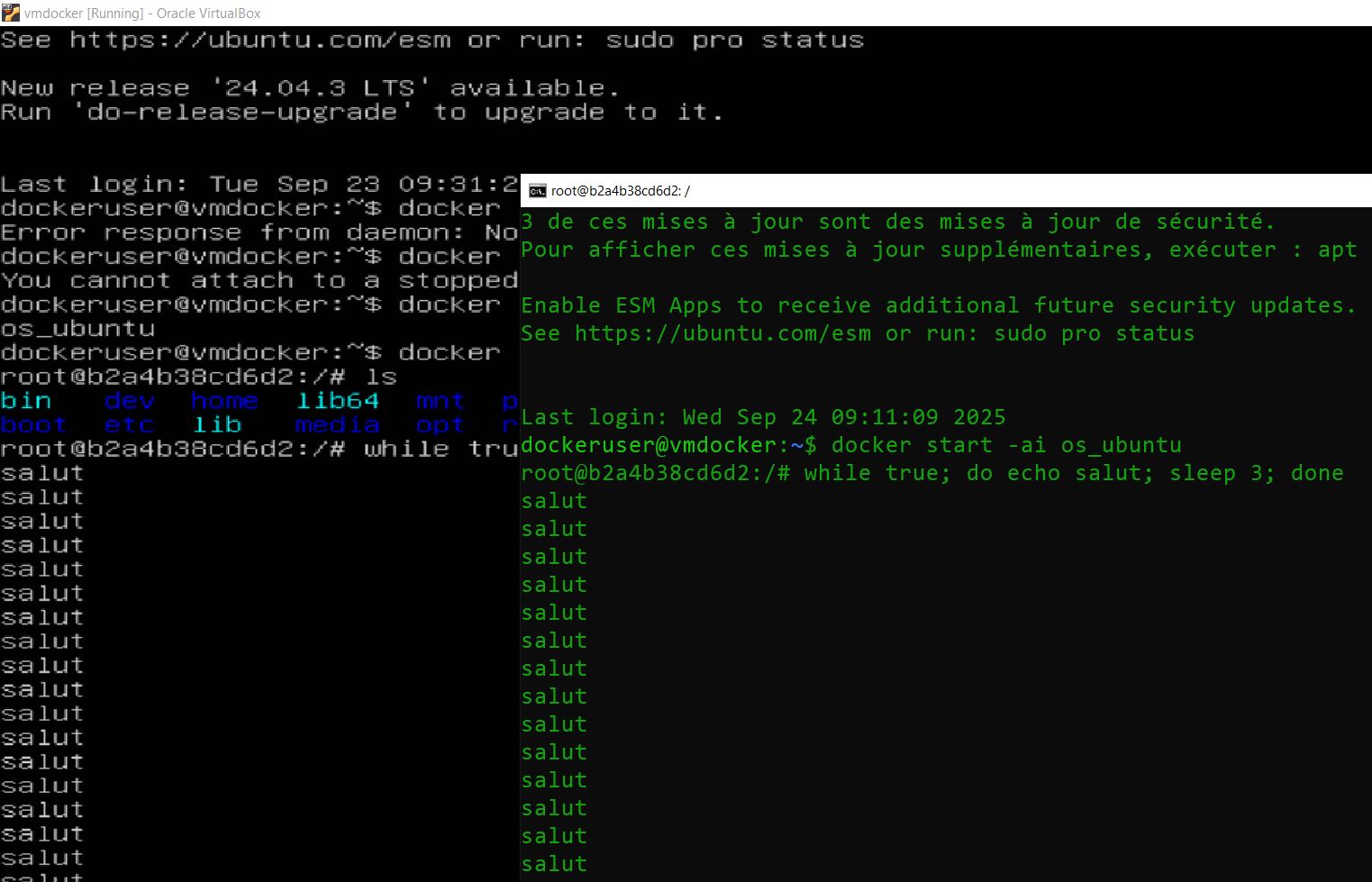
**Résultat** :****

1. **Dans le deuxième terminal connecté en ssh à la VM, utilisez une commande docker pour vous attacher au conteneur qui tourne dans le premier terminal puis constatez. Interrompez le processus qui effectue le salut, lancez une commande basique (par exemple ls) et constatez dans l’autre terminal.**

* **Docker start os\_ubuntu**
* **Docker attach os\_ubuntu**
* **Ls**
* **Loop : while true ; do echo salut; sleep 3; done**
* **Ctrl c stop loop**
* **Ctrl d exit**

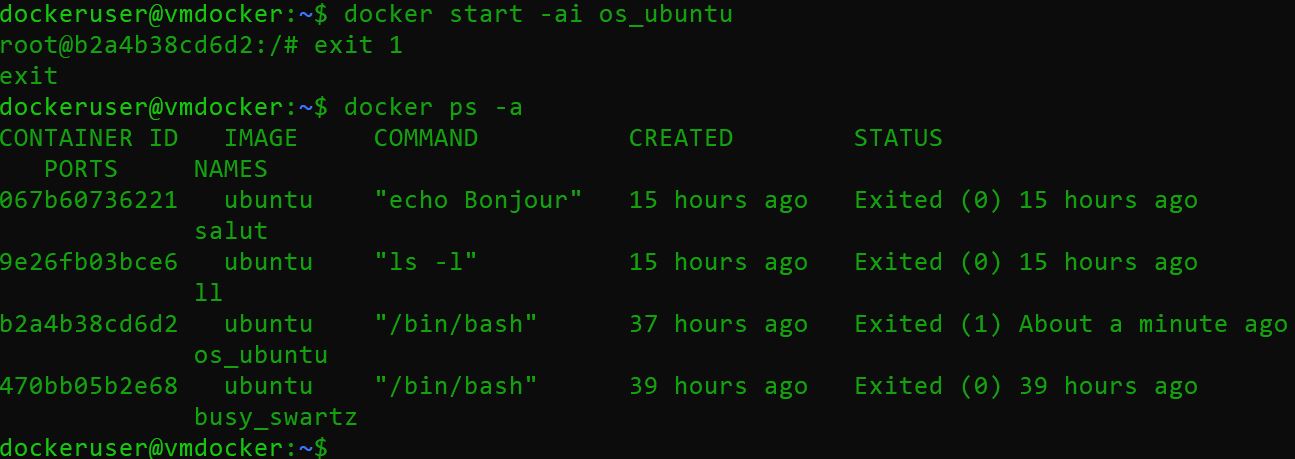
**Conclusion:**

* Les deux terminaux voient la même sortie puisque les deux sont attachés au même conteneur.
* L’interruption d’un processus dans un terminal se reflète dans l’autre.
* Toute commande lancée est visible en simultané sur les deux attaches.

**Résultat** : ****

1. **Quittez ce conteneur en utilisant la commande bash exit avec un code retour non nul. Constatez le statut de ce conteneur.**

* **code retour non nul 🡪Exit 1**
* **Docker ps -a**

**Résultat :**

1. **Quittez ce conteneur en utilisant la commande bash exit avec un code retour non nul. Constatez le statut de ce conteneur.**

* **Le conteneur os\_ubuntu est déjà arrêté (Exited)**🡪docker logs os\_ubuntu
* Docker affiche tout le contenu du log qui a été généré avant que le conteneur se termine.

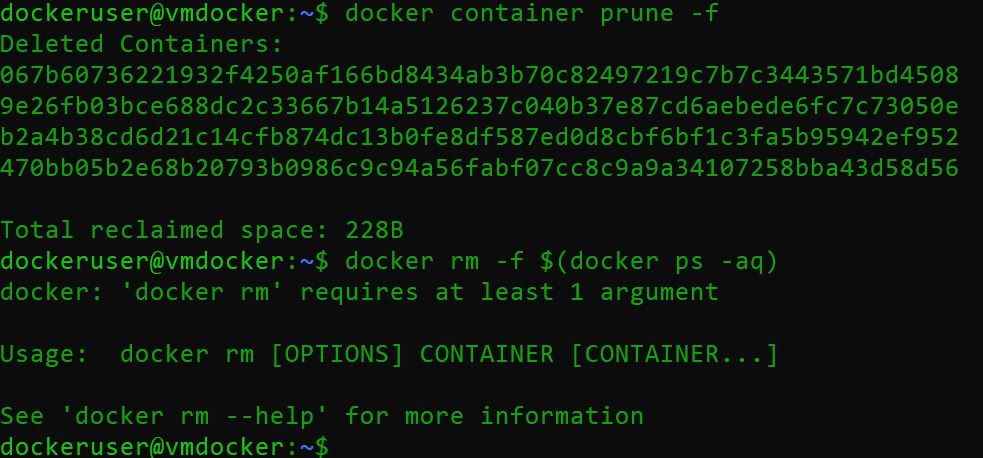
1. **Inspectez le conteneur à la recherche du fichier contenant son journal (log). Vous pouvez le consulter avec jq .**

* docker inspect os\_ubuntu | jq '.[0].LogPath'

**Résultat :**

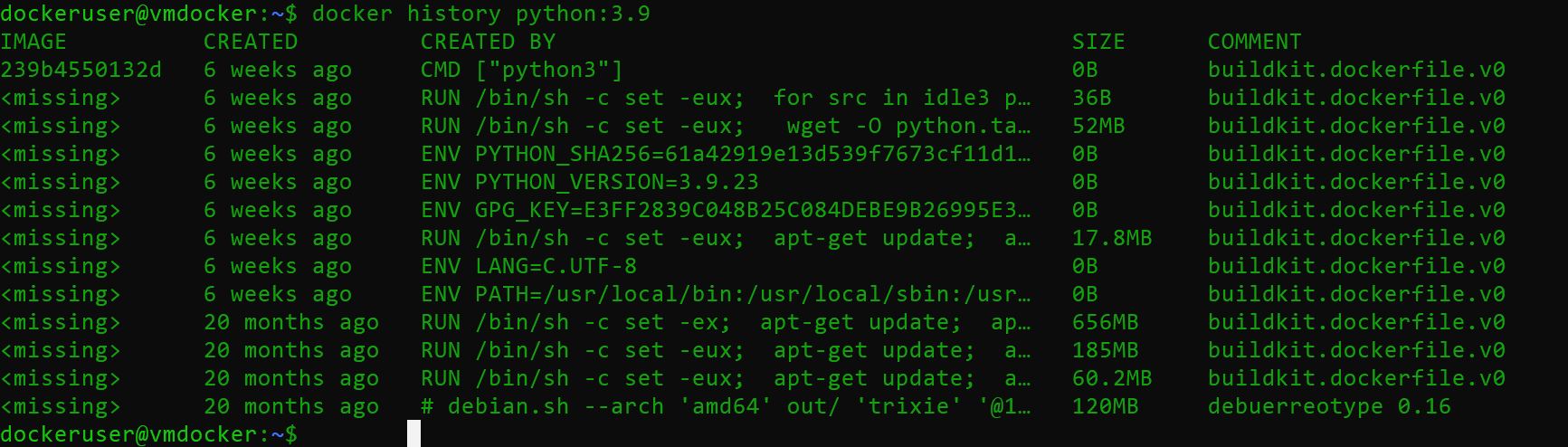
1. **Utilisez une commande docker pour supprimer tous les conteneurs arrêtés. A la fin de cette partie, arrêter et supprimer tous les conteneurs actifs (une seule commande).**

* Pour supprimer tous les conteneurs arrêtés 🡪 docker container prune -f
* Pour supprimer tous les conteneurs actifs 🡪 docker rm -f $(docker ps -aq)

**Résultat :**

**Partie 6 – Inspection et manipulation d’images**

* 1. **Téléchargez l’image python:3.9. - Combien de couches ont été téléchargées ?**
* docker pull python:3.9
* docke history python:3.9

**Résultat :**

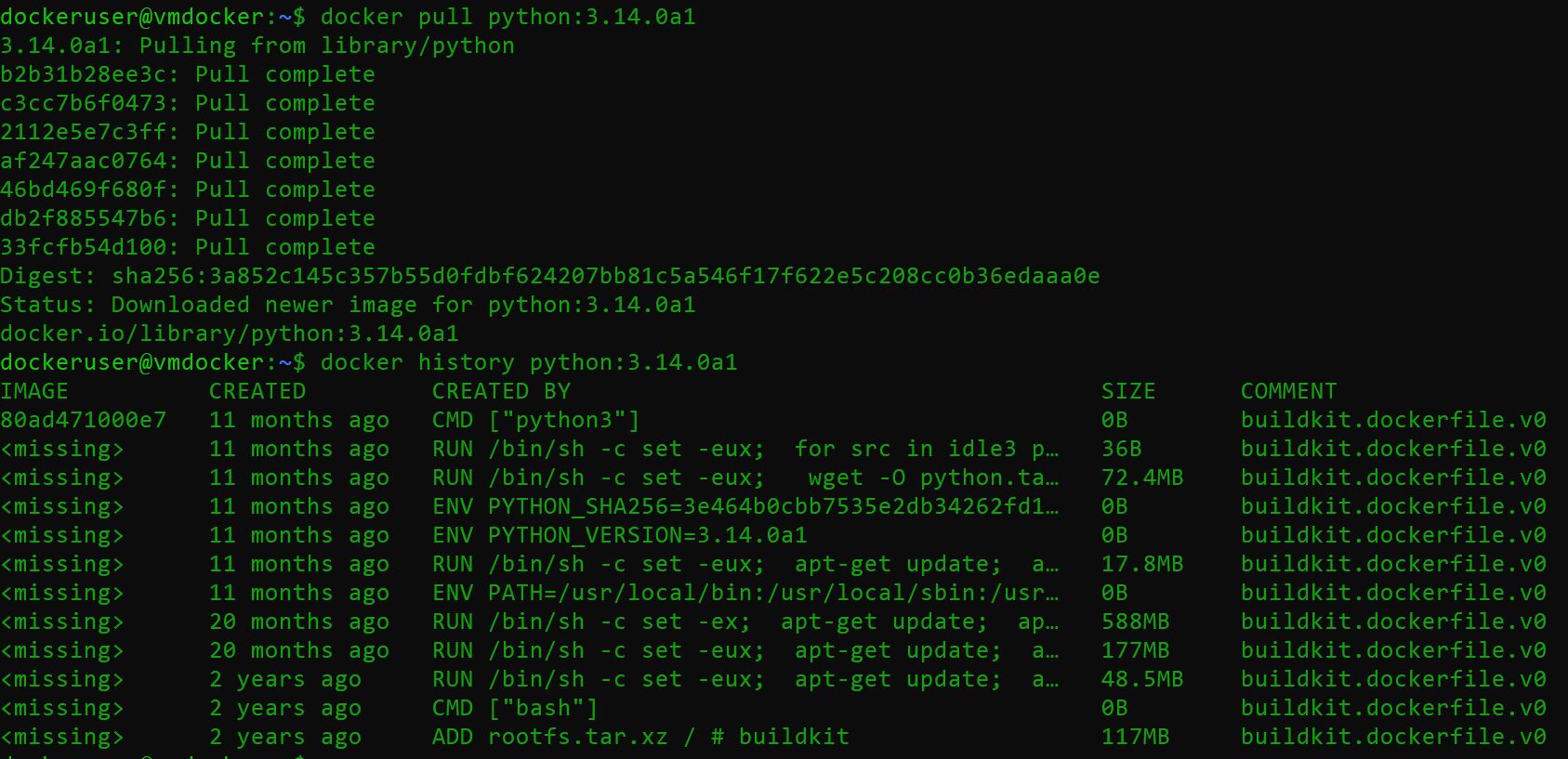
* 1. **Observez les couches de cette image (il existe aussi une option non tronquée)**
* docker history --no-trunc python:3.9

**Explication:**

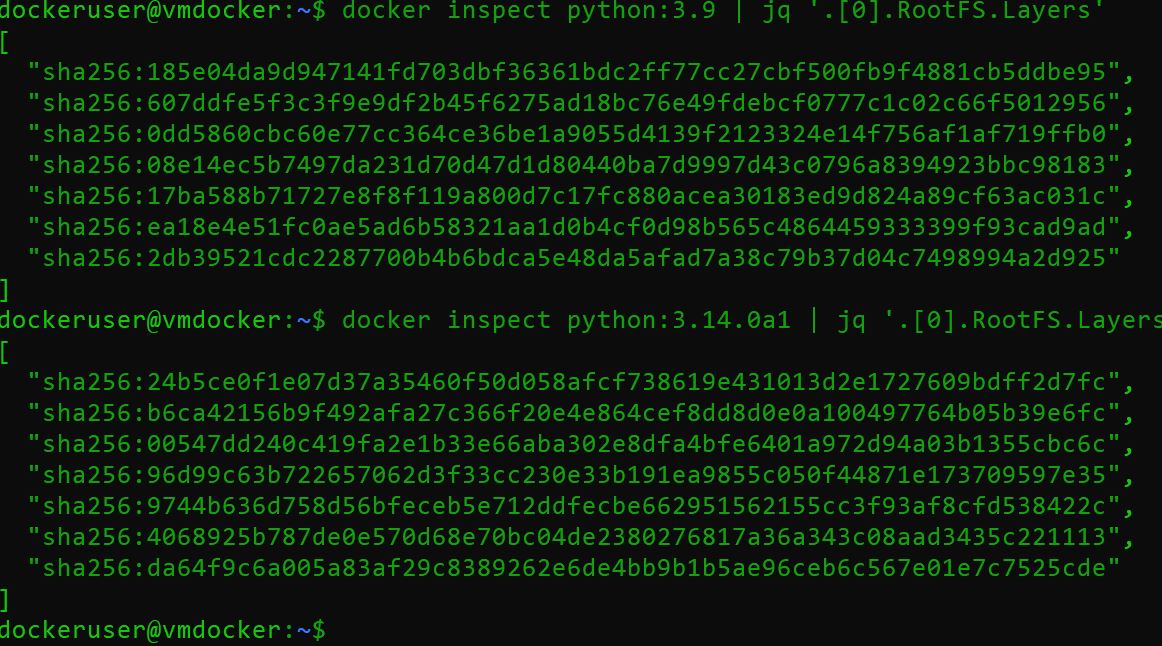
* + **no-trunc montre tous les détails sans couper les colonnes.**
  + **Chaque ligne représente un layer de l’image, avec sa taille, la commande qui l’a créé, et un éventuel commentaire.**

**Résultat :**

* 1. **Quelle est la dernière couche ?**
* # debian.sh --arch 'amd64' out/ 'trixie' '@1757289600'
  1. **Puis téléchargez python : 3.14.0a1. Combien de couches ont été téléchargées ? Expliquez .**
* **télécharger l’image 🡪** docker pull python :3.14.0a1
* **couches ont été téléchargées 🡪** docker history python:3.14.0a1

**Résultat :**

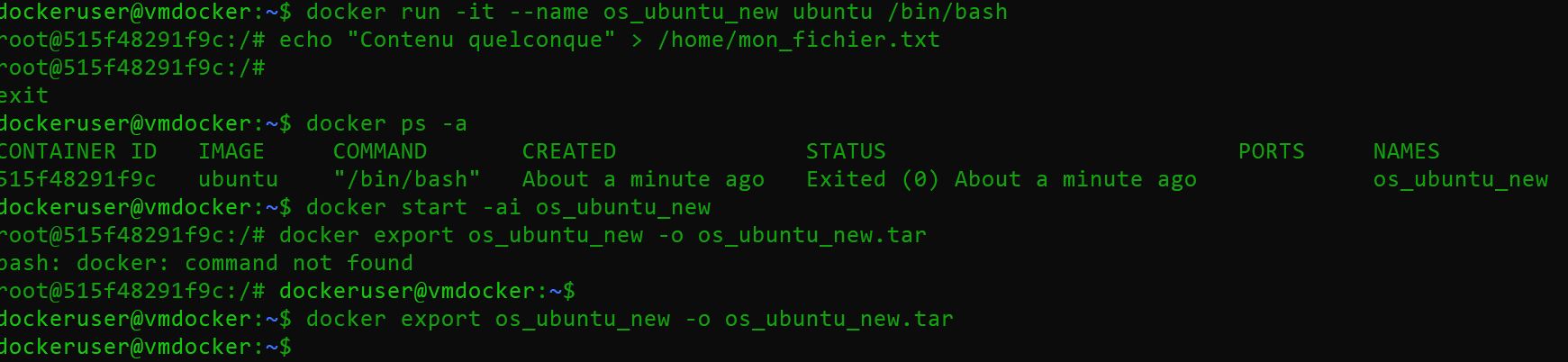
* 1. **Utilisez l’inspection d’une image et l’outil jq pour afficher les couches des 2 images python, observez les couches communes.**
* **Inspecter l’image et afficher les layers avec jq :** docker inspect python:3.9 | jq '.[0].RootFS.Layers' 🡪 docker inspect python:3.14.0a1 | jq '.[0].RootFS.Layers'
* Pas des couches communes

**Résultat :**

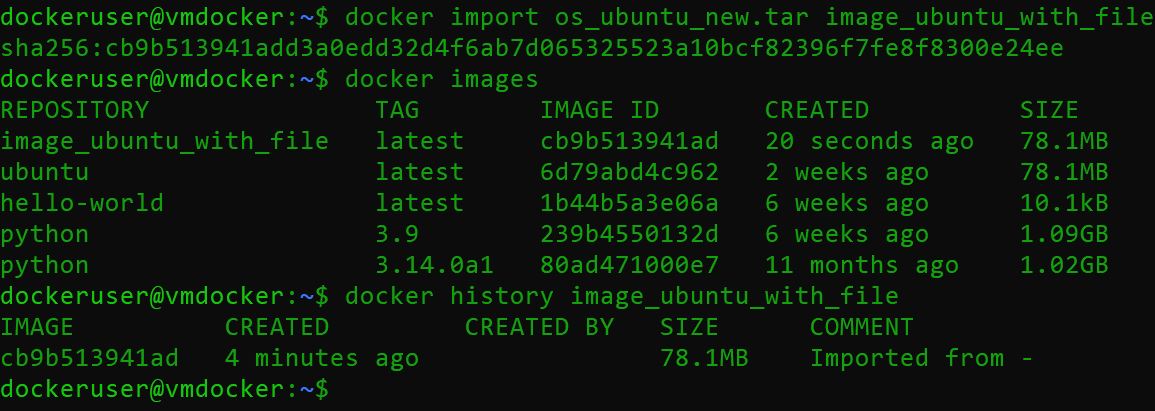
* 1. **Recréez un nouveau conteneur os\_ubuntu à partir de l’image ubuntu et ajoutez dans le répertoire home un nouveau fichier, avec un contenu quelconque. Quittez-le sans l’arrêter. Utilisez une commande docker pour exporter le système de fichiers dans une archive tar.**
* **nouveau conteneur et ajouter un fichier 🡪** docker run -it --name os\_ubuntu\_new ubuntu /bin/bash
* **crée un fichier mon\_fichier.txt dans /home 🡪** echo "Contenu quelconque" > /home/mon\_fichier.txt
* **Exporter le système de fichiers en archive tar🡪** **docker export os\_ubuntu\_new -o os\_ubuntu\_new.tar**

**Explication :**

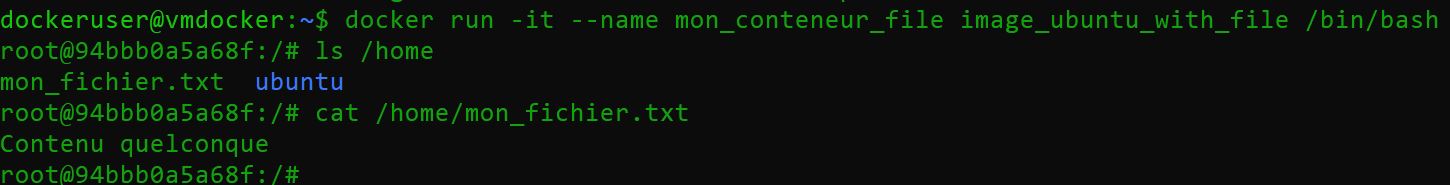
* **Le fichier os\_ubuntu\_new.tar contient maintenant tout le système de fichiers du conteneur, y compris /home/mon\_fichier.txt.**

**Résultat :**

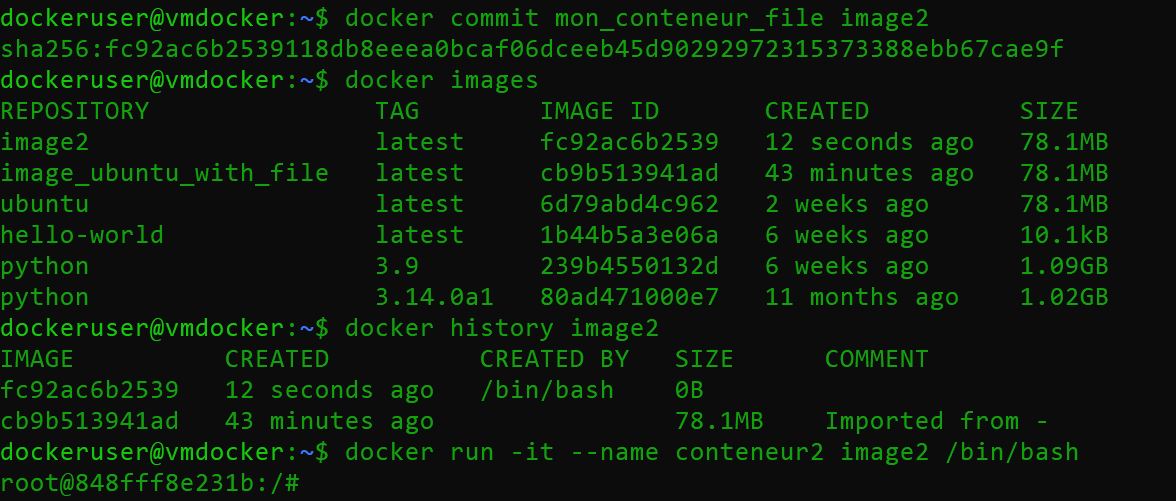
* 1. **Créez une image image\_ubuntu\_with\_file en important l’archive. Comment est-elle taggée ? Observez ses couches (son historique).**
* **importer l’archive pour créer une nouvelle image 🡪** docker import os\_ubuntu\_new.tar image\_ubuntu\_with\_file
* **Vérifier l’image et son tag 🡪**docker images
* **Observer les couches 🡪** docker history image\_ubuntu\_with\_file

**Résultat :**

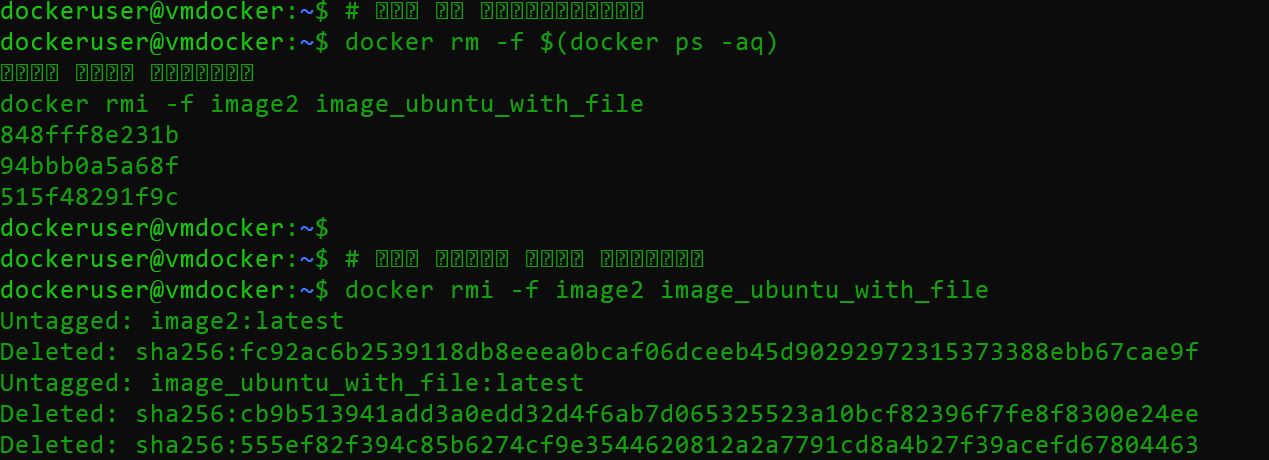
* 1. **Créez un conteneur à partir de cette image. Si vous avez un bash, vous pouvez vérifier que le fichier créé précedemment existe.**
* **Créer un conteneur à partir de l’image 🡪** **docker run -it --**name mon\_conteneur\_file image\_ubuntu\_with\_file /bin/bash
* **Vérifier que le fichier existe 🡪** **ls /home 🡪cat /home/mon\_fichier.txt**

**Résultat :**

* 1. **A partir du conteneur os\_ubuntu, committez maintenant une image image2 et consultez ses couches. Utilisez maintenant cette image pour créer un conteneur.**
* **docker commit mon\_conteneur\_file image2 🡪** docker commit mon\_conteneur\_file image2
* **Vérifier les couches 🡪** docker history image2
* **Créer un conteneur à partir de l’image committee 🡪** docker run -it --name conteneur2 image2 /bin/bash

**Résultat :**

* 1. **A la fin de cette partie, il n’est plus nécessaire de conserver les images crées.**
* **Supprimer tous les conteneurs 🡪** docker rm -f $(docker ps -aq)
* **Supprimer toutes les images🡪** docker rmi -f image2 image\_ubuntu\_with\_file

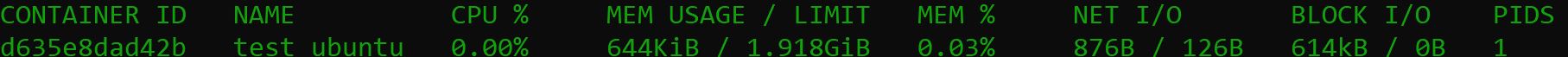
**Résultat :**

**Partie 7 – Quelques informations générales**

**1-Utilisez une commande docker pour consulter la consommation des conteneurs en exécution.**

* docker run -d --name test\_ubuntu ubuntu sleep 300
* docker stats



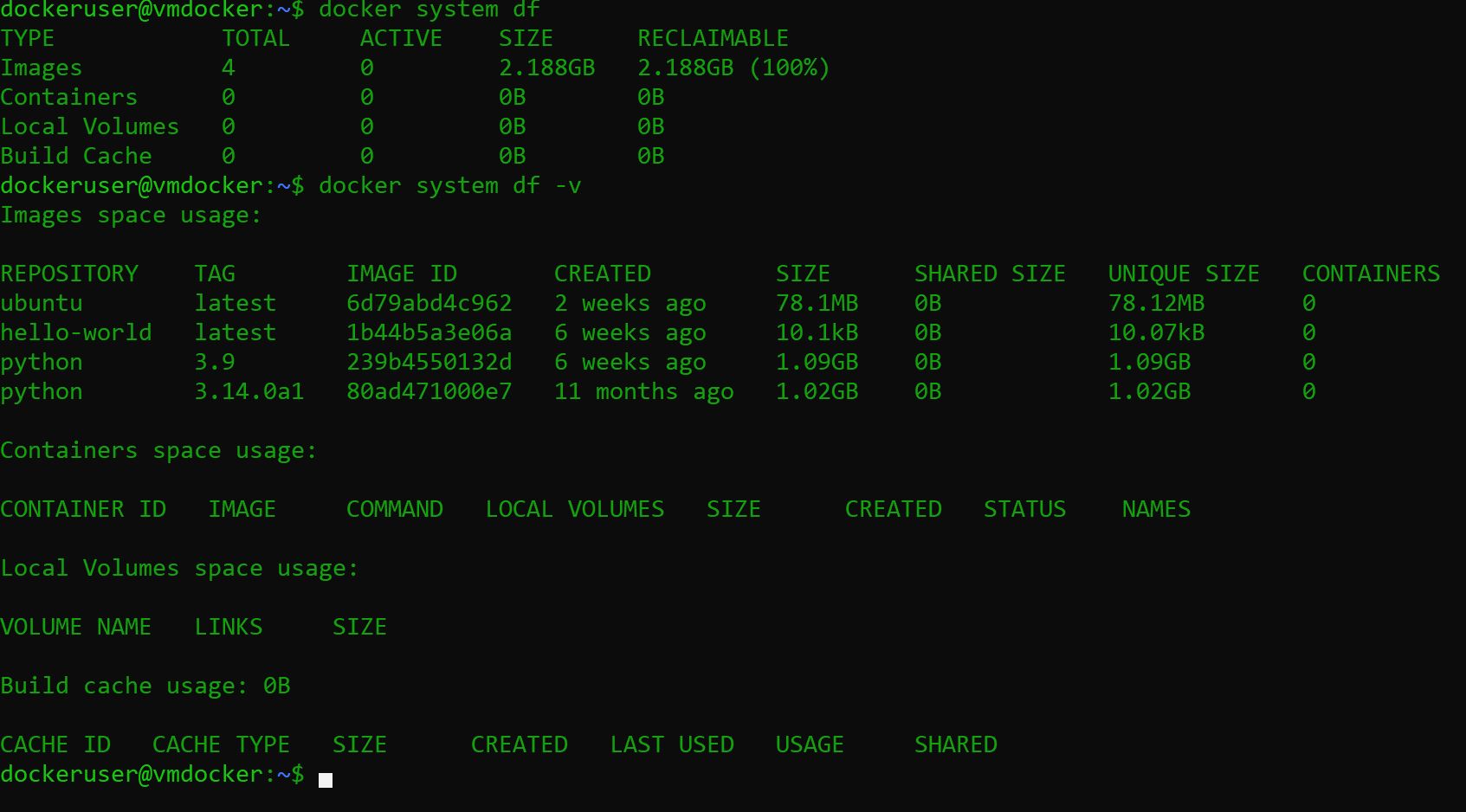


**2- Utilisez une commande docker pour voir la consommation disque des différents objets docker. Essayez aussi la version détaillée.**

* **commande de base 🡪**docker system df
* **Version détaillée 🡪** docker system df -v

**Affiche tous les objets en détail, y compris :**

* **chaque image avec ses couches et taille**
* **chaque conteneur et son état**
* **chaque volume et son utilisation exacte**

****