How to Run Environment Variables In Docker?

written by sysadmin | 21 May 2025

Besides providing application images, Docker also provides database images such as PostgreSQL, MySQL, MariaDB, MongoDB, and so on for its users. As with databases installed on a physical server, which requires entering a username and password to access it, Docker also requires you to enter a username and password to use the database, commonly known as an environment variable.

Problem

How to run environment variables in Docker?

Solution

An environment variable is a dynamically named value that can affect how running processes behave on a computer. They are part of the environment in which a process runs. For example, a running process can query the value of the TEMP environment variable to discover a suitable location to store temporary files, or the HOME or USERPROFILE variable to find the directory structure owned by the user running the process. To find out whether a Docker image can use environment variables, you must check the documentation of the Docker image. Still, in general, Docker images in the form of databases such as MySQL, PostgreSQL, or MongoDB use environment variables.

Please note that if you install an image container that uses an environment variable, for example, installing a MySQL database in a container, but you don’t include an environment variable like in the command below:

```
docker container run -d \
--name db_mysql \
```

```
mysql
```

The container will not run as shown in the image below:

```
sysadmin@docker:~$ docker container run -d \
--name db_mysql \
mysql
e2df3afd6b89191e7ff3bc6d712a5c8bd431e50c97cb0b79bef48aebe6bf9aed
sysadmin@docker:~$
sysadmin@docker:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
sysadmin@docker:~$
sysadmin@docker:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                     PORTS     NAMES
e2df3afd6b89   mysql     "docker-entrypoint.s…"   9 seconds ago   Exited (1) 8 seconds ago             db_mysql
sysadmin@docker:~$
```

Create a container without using environment variables

Docker has parameters that we can use to send environment variables to the application contained in the container by adding the **--env** or **-e** option when we create the container, following the format below:

```
docker container run -d --name container_name --env KEY1="value" --env
KEY2="value" image:tag
```

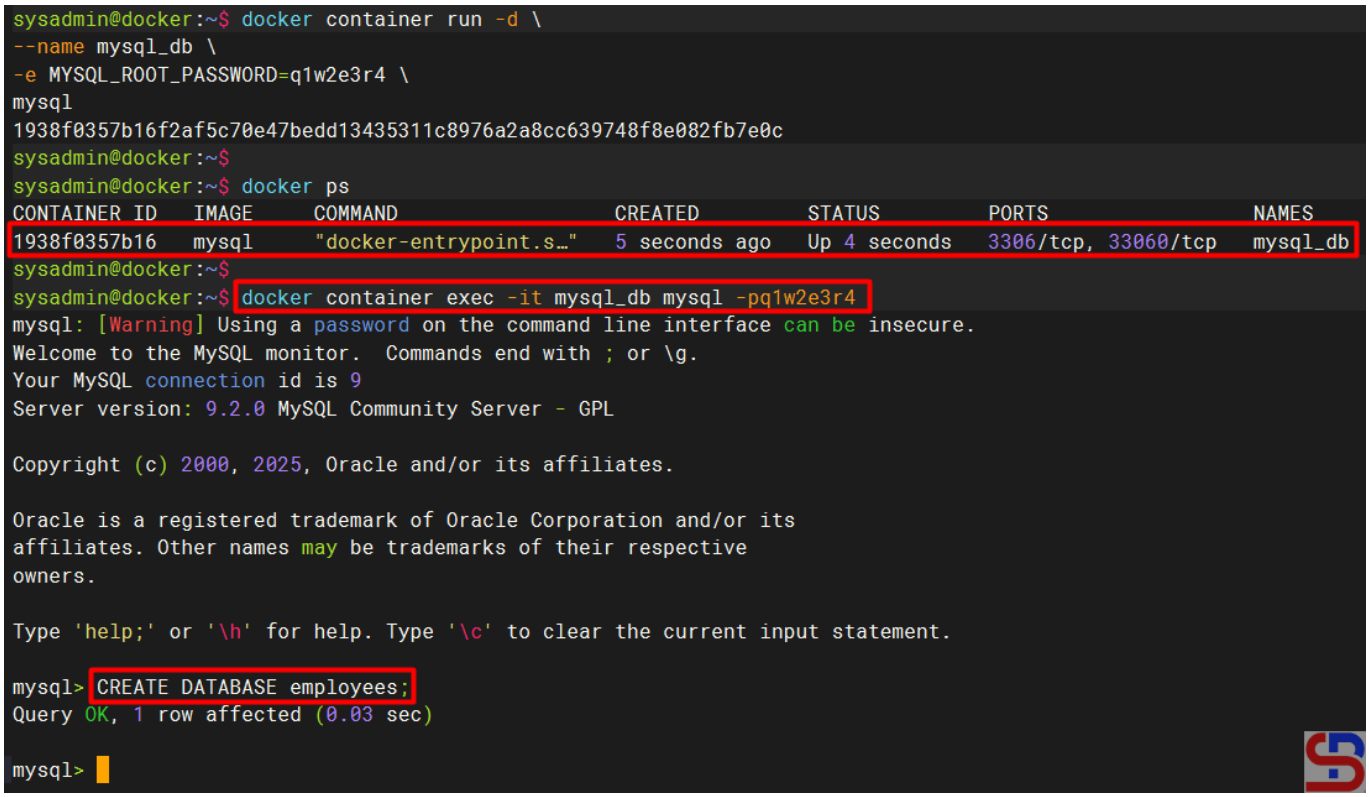
This article will use the MySQL Docker image as a case example. In the documentation, several variables are provided, such as MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, and so on. So, if you want to install MySQL in the container, you have to insert the environment variable. So, if you want to create a MySQL container with root password **q1w2e3r4**, then run the command below:

```
docker container run -d \
--name mysql_db \
-e MYSQL_ROOT_PASSWORD=q1w2e3r4 \
mysql
```

After that, try to access the MySQL database by running the command below:

```
docker container exec -it db_mysql mysql -pq1w2e3r4
```

You will enter the database in the container, like in the image below:



Create the container using environment variables

And you can use database commands as usual, like in the image above.

Note

Besides using the **-e** or **--env** parameter arguments, you can use a file to save the environment variables, commonly known as Env-File, using the **VAR=VALUE** format. Use the format below to run the container that uses Env-File like in the format below:

```
docker container run -d --name container_name --env-file=filename
container_name
```

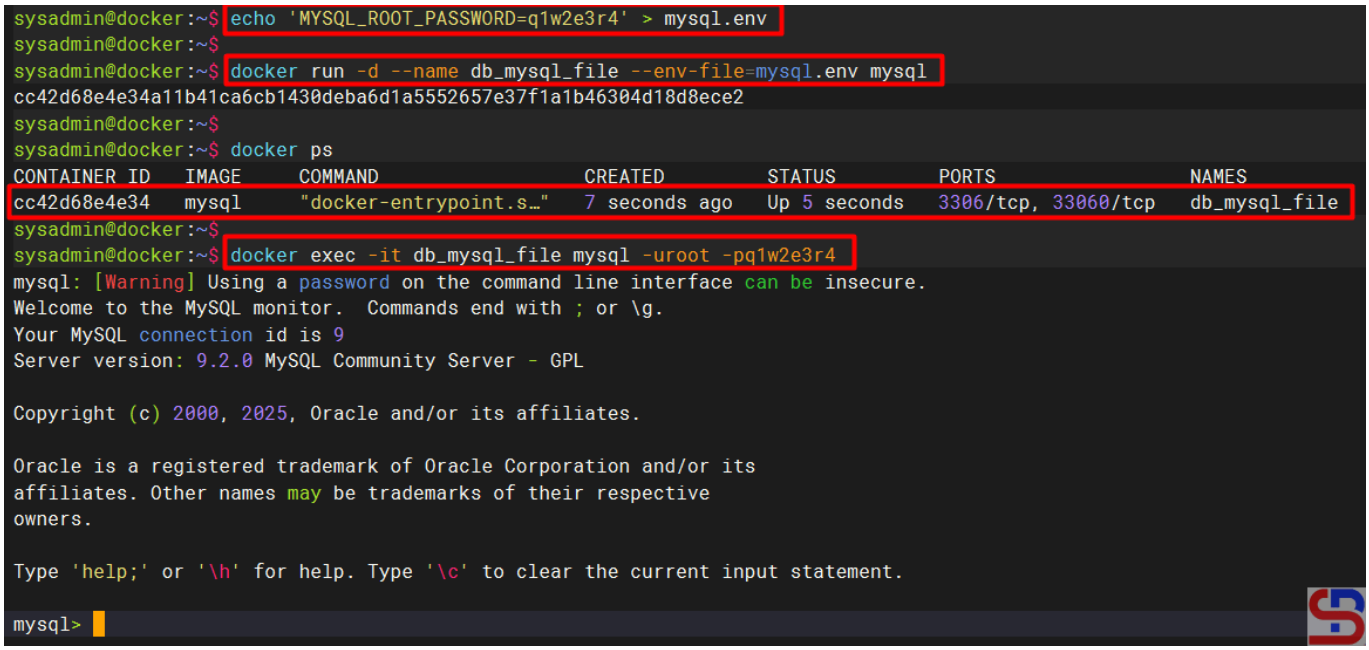
First, you create the file, which usually ends with .env or .env.prod or .env.dev, and I create mysql.env. After that, you add to the file the script below:

```
MYSQL_ROOT_PASSWORD=q1w2e3r4
```

Run the command below to create a new container for MySQL using the environment file:

```
docker run -d --name db_mysql_file --env-file=mysql.env mysql
```

The MySQL container will be created, and you can access the database in the container like in the command below:



Create the container with the Env-File

References

youtube.dimas-maryanto.com
youtube.com
stackoverflow.com

How to Access a Container in Docker?

written by sysadmin | 21 May 2025

After you install Docker and learn some basic Docker commands to set up a container, this article will explain

how to access a container in Docker.

Problem

How to access a container in Docker?

Solution

There are 2 ways to access a container in Docker:

A. Via CLI

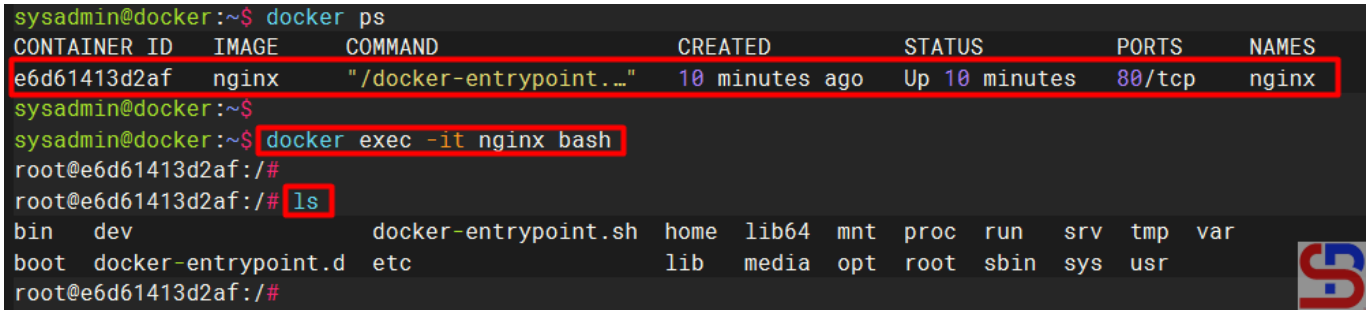
If you want to access a container in Docker, use the format below:

```
docker exec -it container_id/container_name shell
```

where the **-i** option is interactive to keep the input active, the **-t** option is an argument for pseudo -tty (terminal access) allocation, and **shell** is the program contained in the container and it can be different to be different to the code used in the container likes bash or sh shell, but for more details, please look at the documentation of each image). For example, there is an nginx container that is running, and if you want to access the nginx container, you can use the command below:

```
docker exec -it nginx bash
```

After that, you should be able to access the container as shown below:



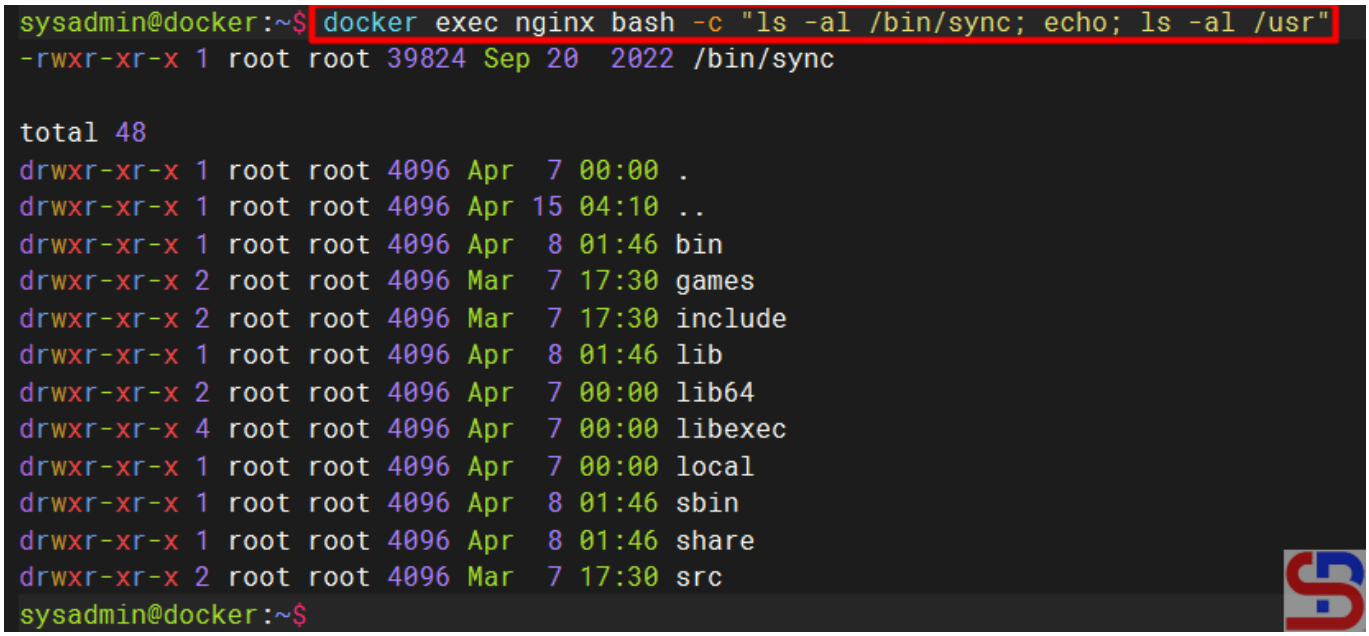
Access into the container

If you want the results of the command in the container to be shown on the server, then use the format below:

```
docker exec container_name/container_id bash -c "your-linux-commands"
```

For example, use the command below:

```
docker exec webapp1 bash -c "ls -al /bin/sync; echo; ls -al /usr"
```



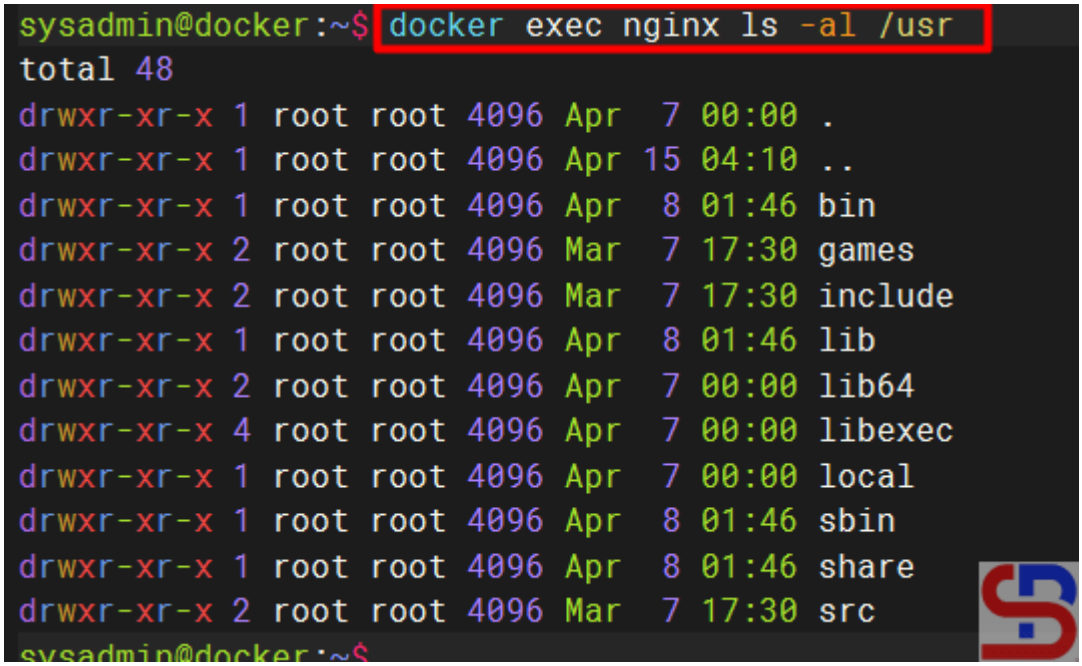
Run some Linux commands in the container

Or you can also use the format below if you only run a command in the container:

```
docker exec container_name/container_id linux-command
```

For example, use the command below:

```
docker exec nginx ls -al /usr
```



Run a command in the container

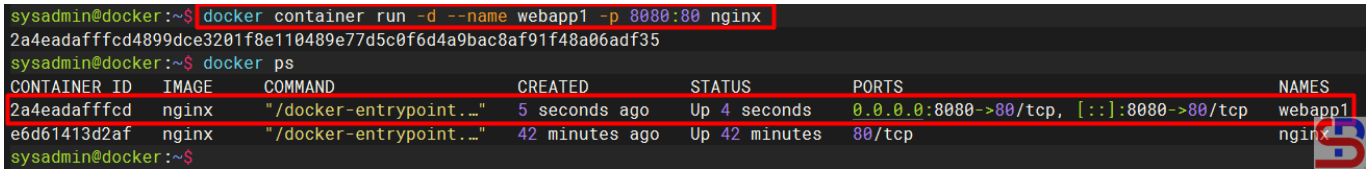
B. Via website

You can access a container through the website, but this method only produces applications that run in the container on the website and do not run commands in the container. Usually, these containers use a web server image such as Apache or Nginx, or applications made by developers. If you want to run these applications and access the application in the browser, use the format below:

```
docker container run -d --name container_name -p port_server:port_container image_name:tag
```

For example, you want to run an nginx container whose application can be seen by using port 8080 on the server, then use the command below:

```
docker container run -d --name webapp1 -p 8080:80 nginx
```

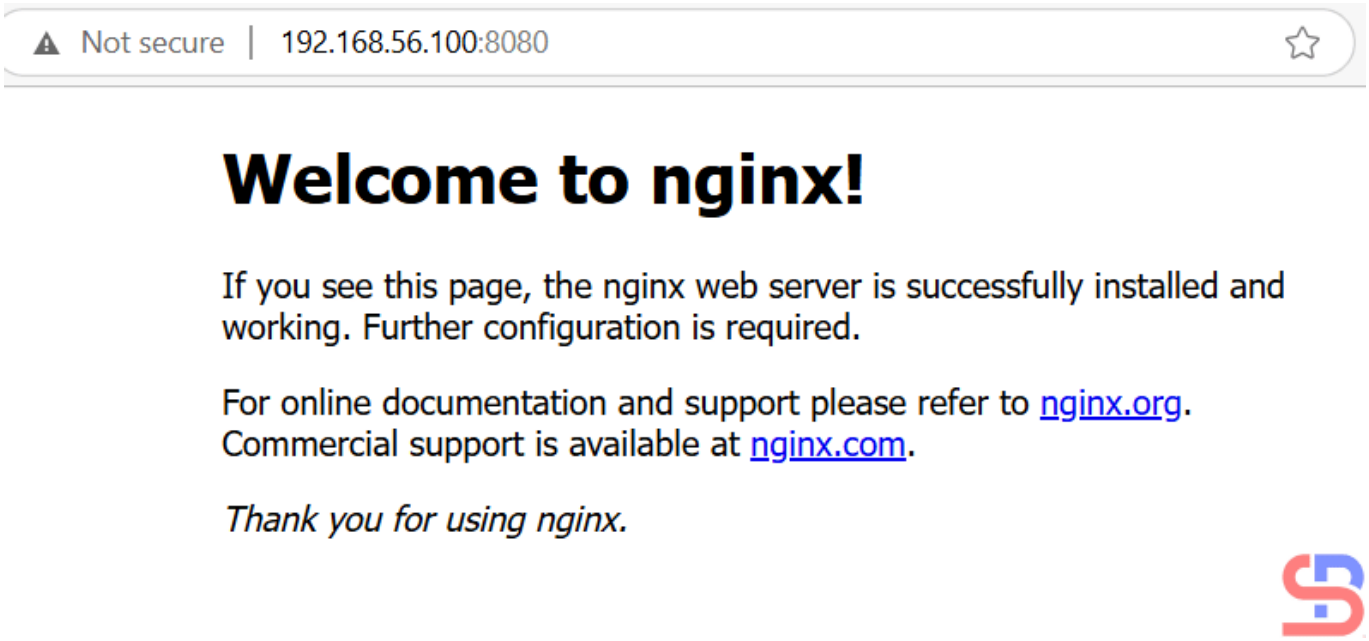


Run the container with the accessed port

Open your browser and type the url below:

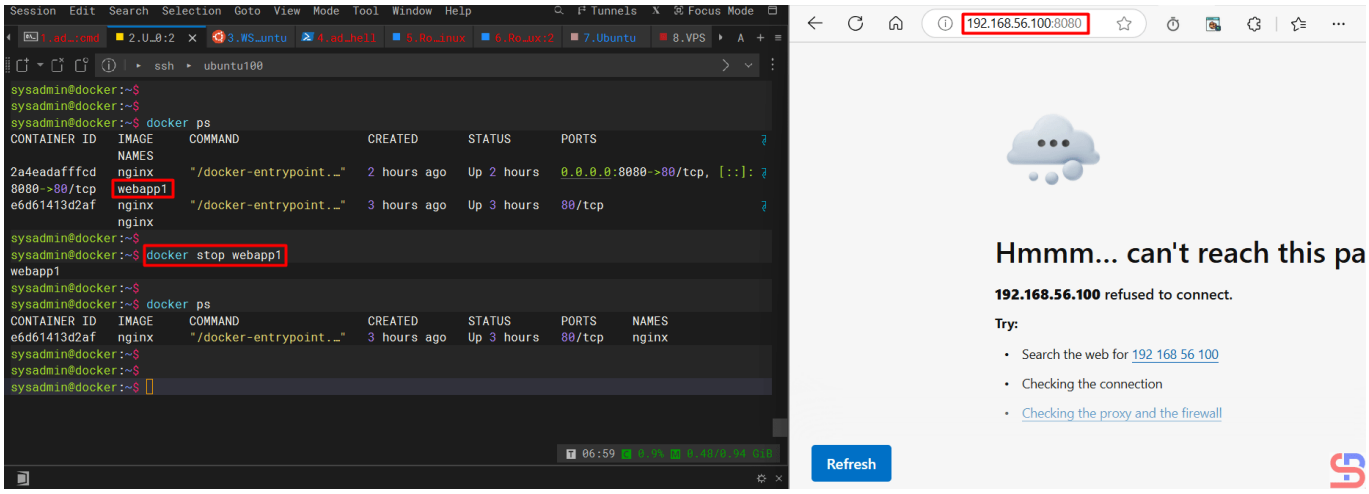
http://your_ip_server:8080

There should be a display as below:



Display the application on the website

If you stop the container, then the application cannot be accessed through a browser as below:



Turn off the container

Note

You can also access a database installed in a container

using the commands in this article.

References

docs.docker.com
spacelift.io
youtube.com

How to Limit Resources in Docker?

written by sysadmin | 21 May 2025

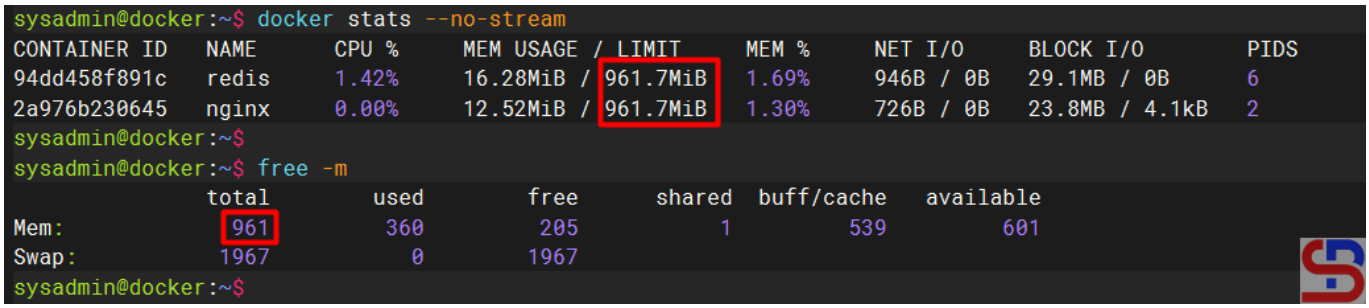
The previous article explained how to monitor an entire container in Docker. In addition to monitoring, you should also limit the resource usage of each container so that server performance remains in good condition.

Problem

How to limit resources in Docker?

Solution

If you run the docker stats command, the command will display the resource container as shown in the image below:



Display stats of the Docker

As you can see in the image above, there are 2 containers

running, and both containers have a memory limit of 941 MB, where the memory size is the size of the memory of the server. This is dangerous because the application in the container can use all the memory or CPU on the server, causing the server to run abnormally. Therefore, you should limit the use of resources in the container.

A. RAM limitation

There are 2 types of memory limitations in Docker:

- The hard limit is a maximum value that cannot be exceeded. When a container exceeds a hard memory limit, Docker takes aggressive actions such as terminating the container so that there will be an OOM or Out Of Memory error in the container. The options used in Docker are **--memory** or **-m**.
- The soft limit is a limit that can be temporarily exceeded. When a soft limit is reached, Docker warns the user but does not take immediate action. The option used is **--memory-reservation**.

If you use swap on the server, you can use the **--memory-swap** option to allocate available swap memory to the container. This swap memory must be larger than the hard limit, usually twice larger than the hard limit. If you want to use a percentage for memory swap, use the **--memory-swappiness** option.

To limit the memory on the new container, use the format below:

```
docker run -d --memory='hard_limit_value' --name docker_name docker-image
```

For example, if you want to limit the memory on a container of 512 MB with a soft limit of 256 MB, then I run the command below:

```
docker run -d --memory='512m' --name nginx nginx
```

```
sysadmin@docker:~$ docker run -d --memory='512m' --name nginx nginx
3302f46fdb23a49fe1d342c8d47ef636f5d4f735318c842537561315b8777c30
sysadmin@docker:~$
sysadmin@docker:~$ docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O         PIDS
3302f46fdb23   nginx     0.00%     2.762MiB / 512MiB     0.54%     586B / 0B     1.3MB / 20.5kB    2
94dd458f891c   redis     1.51%     16.28MiB / 961.7MiB   1.69%     1.16kB / 0B   29.1MB / 0B       6
sysadmin@docker:~$
```

Run Docker with limited memory

You can immediately change the container memory when the container is running using the format below:

```
docker update docker_name --memory='hard_limit_value'
```

For example, I want to change the Redis container memory from 971 MB to 256 MB using the command below:

```
sysadmin@docker:~$ docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O         PIDS
3302f46fdb23   nginx     0.00%     2.762MiB / 512MiB     0.54%     936B / 0B     1.3MB / 20.5kB    2
94dd458f891c   redis     1.27%     16.28MiB / 961.7MiB   1.69%     1.23kB / 0B   29.1MB / 0B       6
sysadmin@docker:~$
sysadmin@docker:~$ docker update redis --memory='256m'
Error response from daemon: Cannot update container 94dd458f891c08d8e0a15eb00878fc83e5c66660d6af8beb0815a3a9040c57f7: Memo
ry limit should be smaller than already set memoryswap limit, update the memoryswap at the same time
sysadmin@docker:~$
```

Error when updating memory

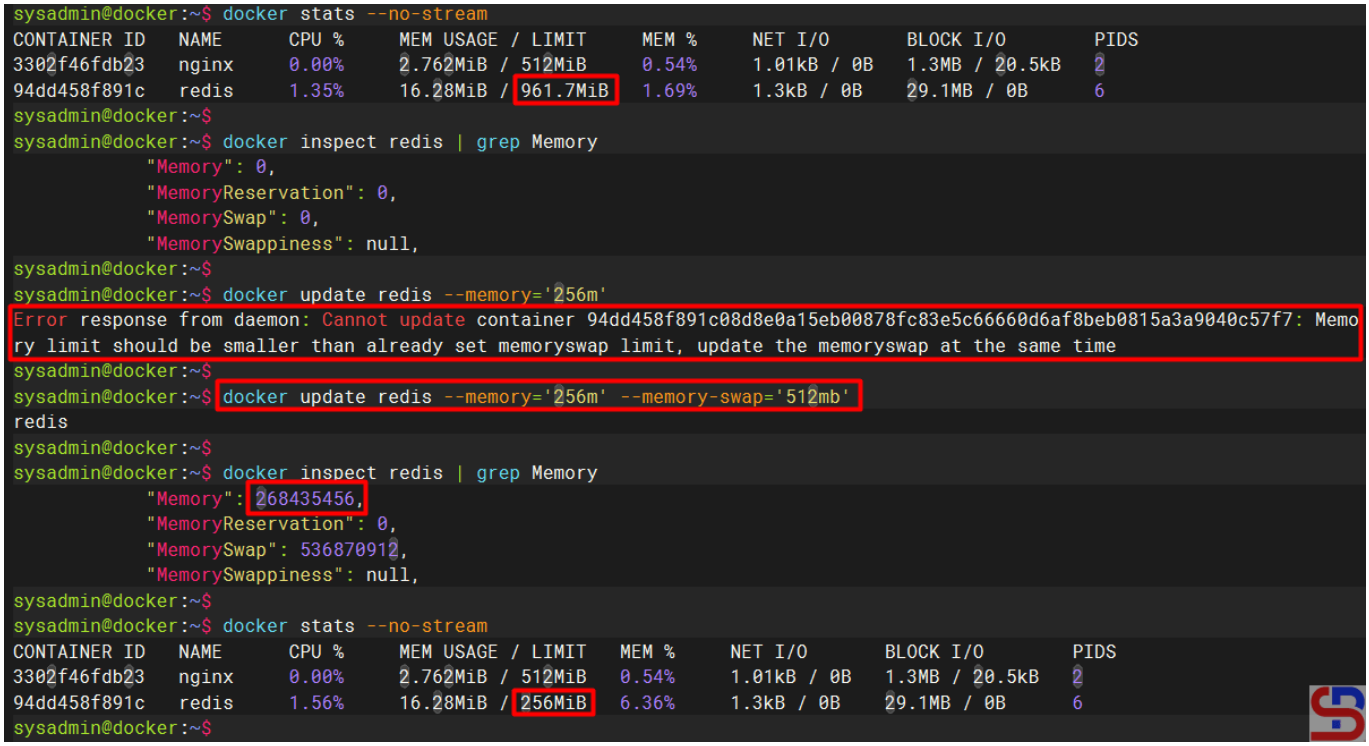
But when running the command, there is an error as below:

```
Error Response from Daemon: Cannot Update Container
94DD458F891C08D8E0A15EB00878FC83E5C6660D6AF8Beb0815A3A9040C57F7: Memory Limit
Should Be Smaller Than Already Set Time
```

To overcome the error, update the memory container swap, whose value must be greater than the memory value. Therefore, use the command below to increase the memory of a container:

```
docker update redis --memory='256m' --memory-swap='512mb'
```

And the container memory value should have changed as shown below:



Update the memory in a container

B. CPU limitation

Before you limit the CPU in the container, you need to know how many CPU cores there are on your server by using the command below:

```
nproc
```

To limit the CPU used in the container, use the **--cpus** option to determine how many CPU cores can be used in a container. If your server has two CPUs and you set --cpus=’1.5′, the container is guaranteed at most one and a half of the CPUs and this is the equivalent of setting --cpu-period=’100000′ and --cpu-quota=’150000′ (cpu-period is used with cpu-quota to configure the CPU scheduler with defaults to 100000 microseconds and cpu-quota is used with cpu-period to configure the CPU scheduler). Use the format below to limit the CPU in the container:

```
docker run -d --cpus='cpu_value' --name docker_name docker_image:tag
```

You can see the details of the CPU used by a container by using the format below:

```
docker inspect nginx | grep -e Cpu -e cpu
```

For example, I want to limit the CPU to 1, so I run the command below:

```
docker run -d --cpus='1' --name nginx nginx
```

```
sysadmin@docker:~$ docker run -d --cpus='1' --name nginx nginx
87ef6df710dcbf459a6f53db4ddb26445d97809e7c0b8174a0bb79c344a19054
sysadmin@docker:~$
sysadmin@docker:~$ docker stats --no-stream
CONTAINER ID   NAME      CPU %     MEM USAGE / LIMIT    MEM %     NET I/O     BLOCK I/O         PIDS
87ef6df710dc   nginx     0.00%     11.5MiB / 961.4MiB   1.20%     806B / 0B   9.56MB / 12.3kB   3
sysadmin@docker:~$
sysadmin@docker:~$ docker inspect nginx | grep -i cpus
            "CpuShares": 0,
            "NanoCpus": 1000000000,
            "CpusetCpus": "",
            "CpusetMems": "",
sysadmin@docker:~$
```

Run a container with a limited CPU

Warning

You have to be careful in determining the CPU limitations of a container because if you limit the CPU too much (for example, giving 0.5 CPU to the container even though the container can run using 1 CPU) or give a CPU limitation that is greater than the CPU the server uses (for example The server CPU only has 1 CPU but you give the container 2 CPUs) then the container will immediately turn off automatically.

You can use the **--cpu-shares** option for a container to control the share of CPU cycles available, whose default value is 1024. This option is like a soft limit option on memory, so if you run the command below:

```
docker run -d --cpu-shares='2048' --name webapp1 nginx
```

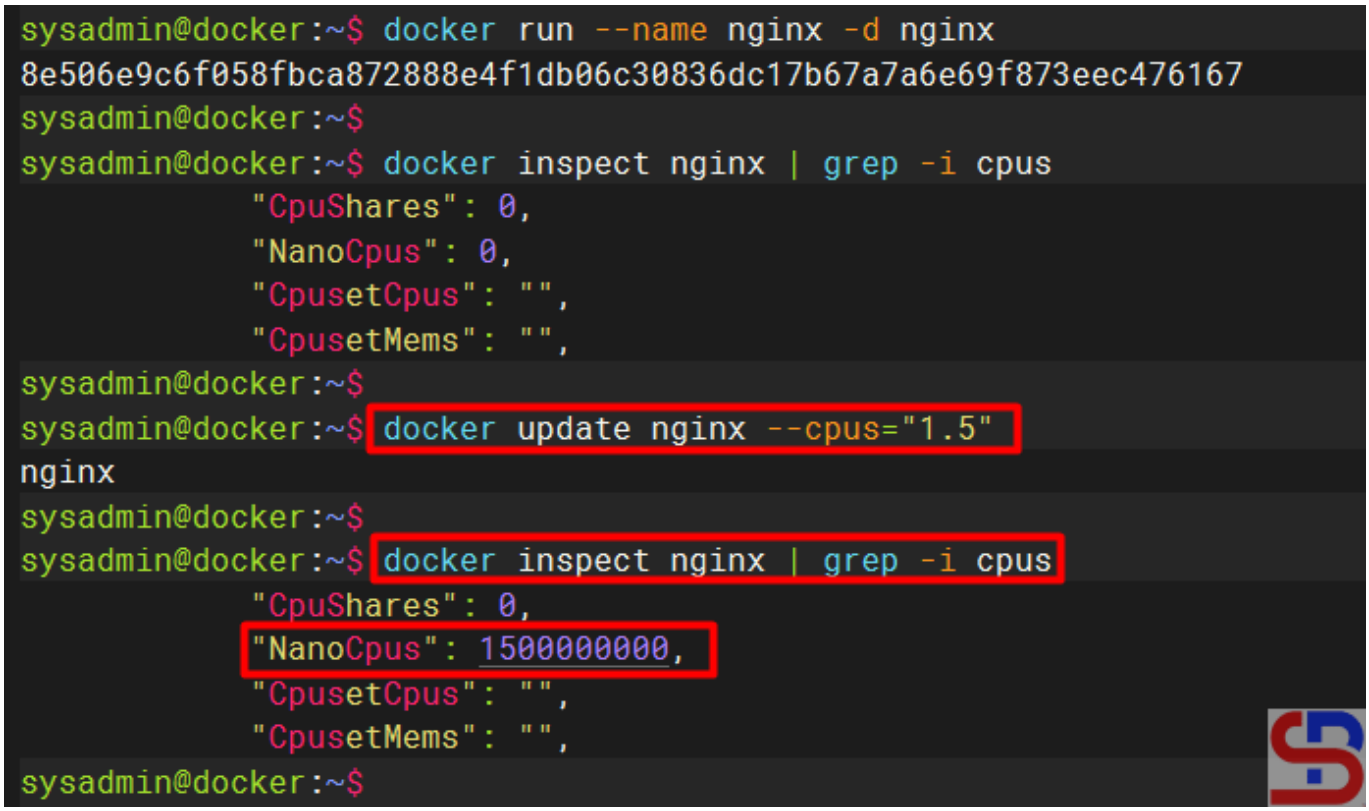
If there is more than 1 container on a host and CPU cycles are constrained, then the webapp1 container will receive 2x more CPU than the other containers. You can update the CPU

in a running container using the format:

```
docker update docker_name --cpus='cpu_value'
```

For example, I have 2 CPUs on the server, and initially, I use all the CPUs in the webapp1 container. Then, I wanted to update the CPU on the container to 1.5, so I ran the command below:

```
docker update webapp1 --cpus='1.5' nginx
```



Update the CPU in a container

C. HDD limitation

As of this writing (April 2025), the HDD limitation can only be used for **btrfs**, **overlay2**, **windowsfilter**, and **zfs** storage drivers. For the overlay2 storage driver, the size option is only available if the backing filesystem is **xfs** and mounted with the **pquota** mount option. Type the command below to see the Docker settings on the server:

```
docker info | grep -e 'Filesystem' -e 'Support' -e 'Storage'
```

Run the command above, and here is the result:

```
sysadmin@docker:~$ docker info | grep -e 'Filesystem' -e 'Support' -e 'Storage'
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
sysadmin@docker:~$
```

Check the filesystem

To limit the hard disk in a container, follow the format below:

```
docker run -d --name nginx --storage-opt size=1g nginx
```

If I want to limit my hard disk in a container, I run the command below:

```
docker run -d --name nginx --storage-opt size=1g nginx
```

But, I have an error like the image below:

```
docker: Error response from daemon: –storage-opt is supported only for overlay pver xfs with ‘pquota’ mount option
```

```
sysadmin@docker:~$ docker run -d --name redis --storage-opt size=1g redis
docker: Error response from daemon: --storage-opt is supported only for overlay over xfs with 'pquota' mount option.
See 'docker run --help'.
sysadmin@docker:~$
```

Error when limiting HDD for a container

From the image above, I can’t limit the HDD to the container because my Backing Filesystem in my server still uses extfs, not xfs. So, if I want to limit my HDD in my containers, I have to change my Backing Filesystem in my server. So I made an experiment by turning off Docker and deleting the Docker folder using the command below:

```
sudo systemctl stop docker
sudo rm -rf /var/lib/docker && sudo mkdir /var/lib/docker
```

Then I inserted an additional hard disk into the existing server, and then I converted the file system to xfs using the commands below:

```
sudo mkfs.xfs /dev/sdb1
sudo mount /dev/sdb1 /var/lib/docker
```

```
sysadmin@docker:~$ sudo mount /dev/sdb1 /var/lib/docker/
sysadmin@docker:~$
sysadmin@docker:~$ df -T
Filesystem                         Type  1K-blocks    Used Available Use% Mounted on
tmpfs                              tmpfs     98448    1112     97336   2% /run
/dev/mapper/ubuntu--vg-ubuntu--lv  ext4   10218772 6110044   3568056  64% /
tmpfs                              tmpfs    492220       0    492220   0% /dev/shm
tmpfs                              tmpfs      5120       0      5120   0% /run/lock
/dev/sda2                          ext4    1768056   96560   1563364   6% /boot
tmpfs                              tmpfs     98444      12     98432   1% /run/user/1000
/dev/sdb1                          xfs     2030592   71980   1958612   4% /var/lib/docker
sysadmin@docker:~$
```

Create xfs filesystem

And I added to the **/etc/fstab** file the script below:

```
echo '/dev/sdb1 /var/lib/docker xfs defaults,quota,prjquota,pquota,gquota 0
0' | sudo tee -a /etc/fstab
```

I rebooted the server to test whether the fstab file settings were correct. After that, I tried to create a container by limiting the container's hard disk to 1GB using the command below:

```
docker run -d --name nginx --storage-opt size=1g nginx
```

```
sysadmin@docker:~$ docker info | grep -e 'Filesystem' -e 'Support' -e 'Storage'
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
sysadmin@docker:~$
sysadmin@docker:~$ docker run -d --name nginx --storage-opt size=1g nginx
8d609d92bcc7d2b6be5d0c3f888ad6f7d6135f05466ded4761cb6a6941c59a17
sysadmin@docker:~$
sysadmin@docker:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
8d609d92bcc7   nginx     "/docker-entrypoint.…"   10 seconds ago   Up 9 seconds    80/tcp    nginx
sysadmin@docker:~$
```

Limiting HDD in a container

Note

You can combine more than one restriction above by using one command. For example, if you want to limit memory, CPU, and HDD in a container, then you can run the command below:

```
docker run -d --name webapp1 -m 512m --cpus=1.5 --storage-opt size=1g nginx
```

As far as I know, Sysadmin only limits the use of RAM and CPU in Docker, but rarely limits HDD in Docker. If you want to limit resources in Docker, you must discuss it with the developers so that there are no problems with the application in the future.

References

phoenixnap.com
baeldung.com
docs.docker.com
nodramadevops.com
stackoverflow.com
hands-on.cloud
blogs.perficient.com
blog.devops.dev
youtube.com
reddit.com

How to Monitor Containers in Docker?

written by sysadmin | 21 May 2025

After you run containers in Docker on your server or your Docker Host, you should monitor all existing containers to find out the performance of each container.

Problem

How to monitor containers in Docker?

Solution

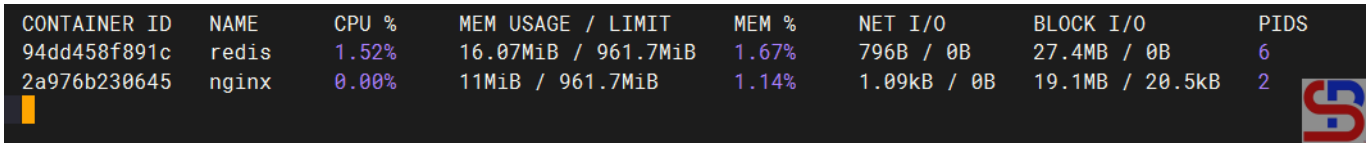
There are 2 methods for container monitors in Docker:

A. Via CLI

In CLI, to monitor all containers in Docker, you can use the command:

```
docker stats
```

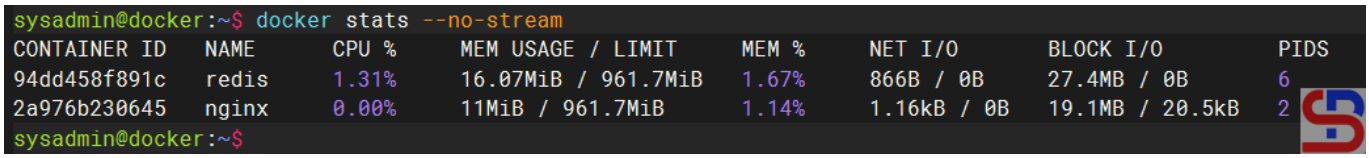
You will see the display as below:



Using the docker stats command

From the image above, you can see that the command displays the results in streaming, and to exit from the command above, press **Ctrl+Z** or **Ctrl-C**. If you don't want to display the results in streaming mode, then use the command below:

```
docker stats --no-stream
```



Display monitor containers in Docker without stream

B. Via Website

If you want to monitor Docker via a website, you can use the Portainer tool. Portainer is a tool for managing containers through a browser that can support Docker host, Docker Swarm, Nomad, and Kubernetes. It has 2 components, namely Portainer Server, which is used to manage containers, networks, and environments, and Portainer Agent is the component installed on another Docker system to enable

communication with the server. Portainer has 2 editions, namely Portainer Business Edition or PBE and Portainer Community Edition or PCE, where both editions at the time of this writing (April 2025) have version 2.27.3. This article will discuss how to install Portainer Community Edition. Here are the steps:

1. Create Docker Volume

Type the command below to create a new volume in Docker:

```
docker volume create portainer_data
```

2. Install Portainer

Type the command below to install the latest version of Portainer:

```
docker run -d \
-p 8000:8000 \
-p 9443:9443 \
--name portainer \
--restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data portainer/portainer-ce
```

```
sysadmin@docker:~$ docker run -d \
-p 8000:8000 \
-p 9443:9443 \
--name portainer \
--restart=always \
-v /var/run/docker.sock:/var/run/docker.sock \
-v portainer_data:/data portainer/portainer-ce
Unable to find image 'portainer/portainer-ce:latest' locally
latest: Pulling from portainer/portainer-ce
e2e06b27b87e: Pull complete
1fed1531b45b: Pull complete
04de093ad5ed: Pull complete
86a7cce72d42: Pull complete
e09df2601140: Pull complete
eae3ebf29ea8: Pull complete
c12aa3fbd31a: Pull complete
f111bda3f9a6: Pull complete
81021110ed01: Pull complete
4f4fb700ef54: Pull complete
Digest: sha256:7f10a26bfdda3fc58295ea09b860117ecd86a642d66fb94ce1f27a4c221d4649
Status: Downloaded newer image for portainer/portainer-ce:latest
12496e61ee8addcff1a3a18ff95ade6802c951622fe6e4a6e2b23a030d6bb082
sysadmin@docker:~$
```

Install portainer

3. Check the Portainer

The following command can be used to determine whether Portainer is operating or not:

docker ps

```
sysadmin@docker:~$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS
                                                                                 NAMES
12496e61ee8a   portainer/portainer-ce   "/portainer"             4 minutes ago    Up 4 minutes    0.0.0.0:8000->8000/tcp,
:::8000->8000/tcp, 0.0.0.0:9443->9443/tcp, :::9443->9443/tcp, 9000/tcp   portainer
94dd458f891c   redis                  "docker-entrypoint.s…"   29 minutes ago   Up 29 minutes   6379/tcp
                                                                                 redis
2a976b230645   nginx                  "/docker-entrypoint.…"   31 minutes ago   Up 31 minutes   80/tcp
                                                                                 nginx
sysadmin@docker:~$
```

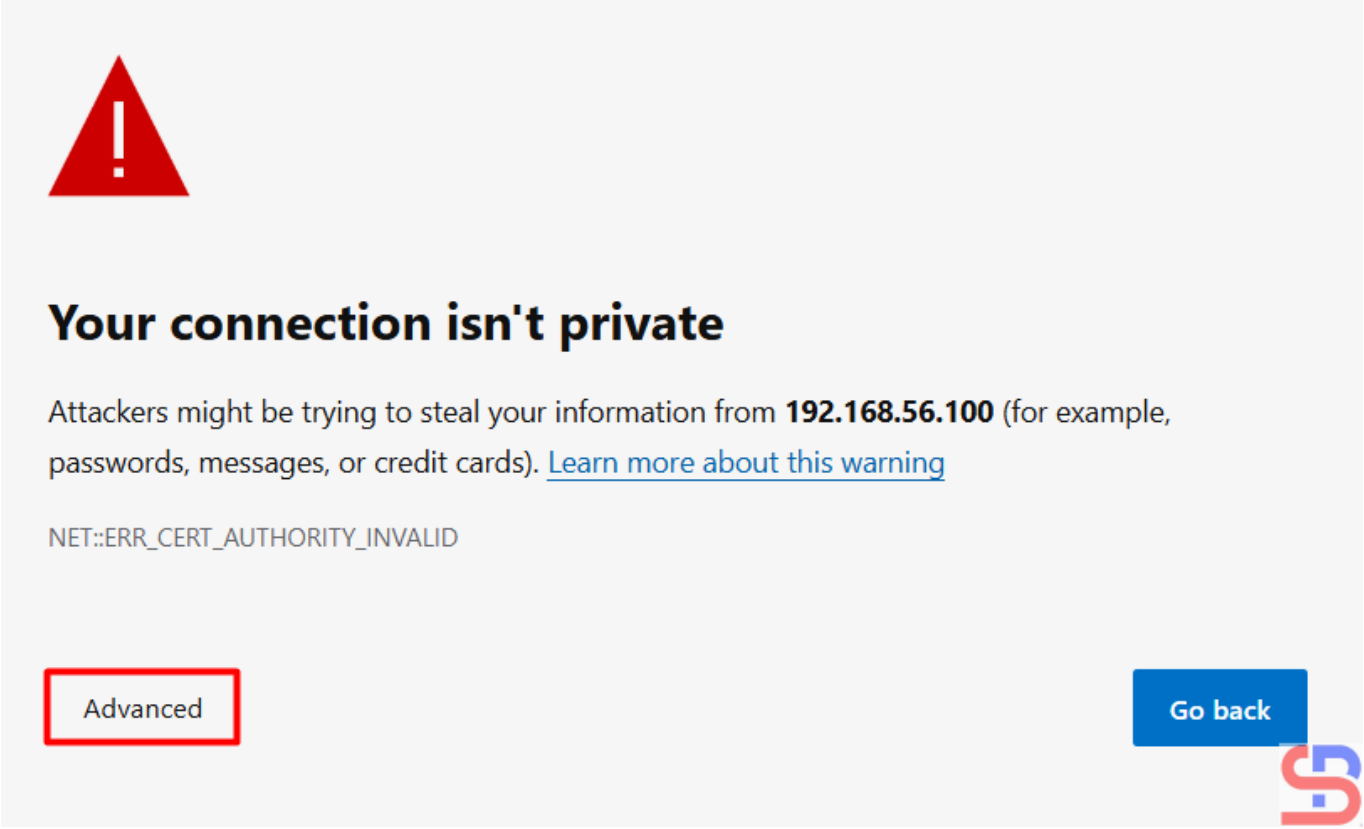
Check the container

4. Access the Portainer

After that, open your browser and type:

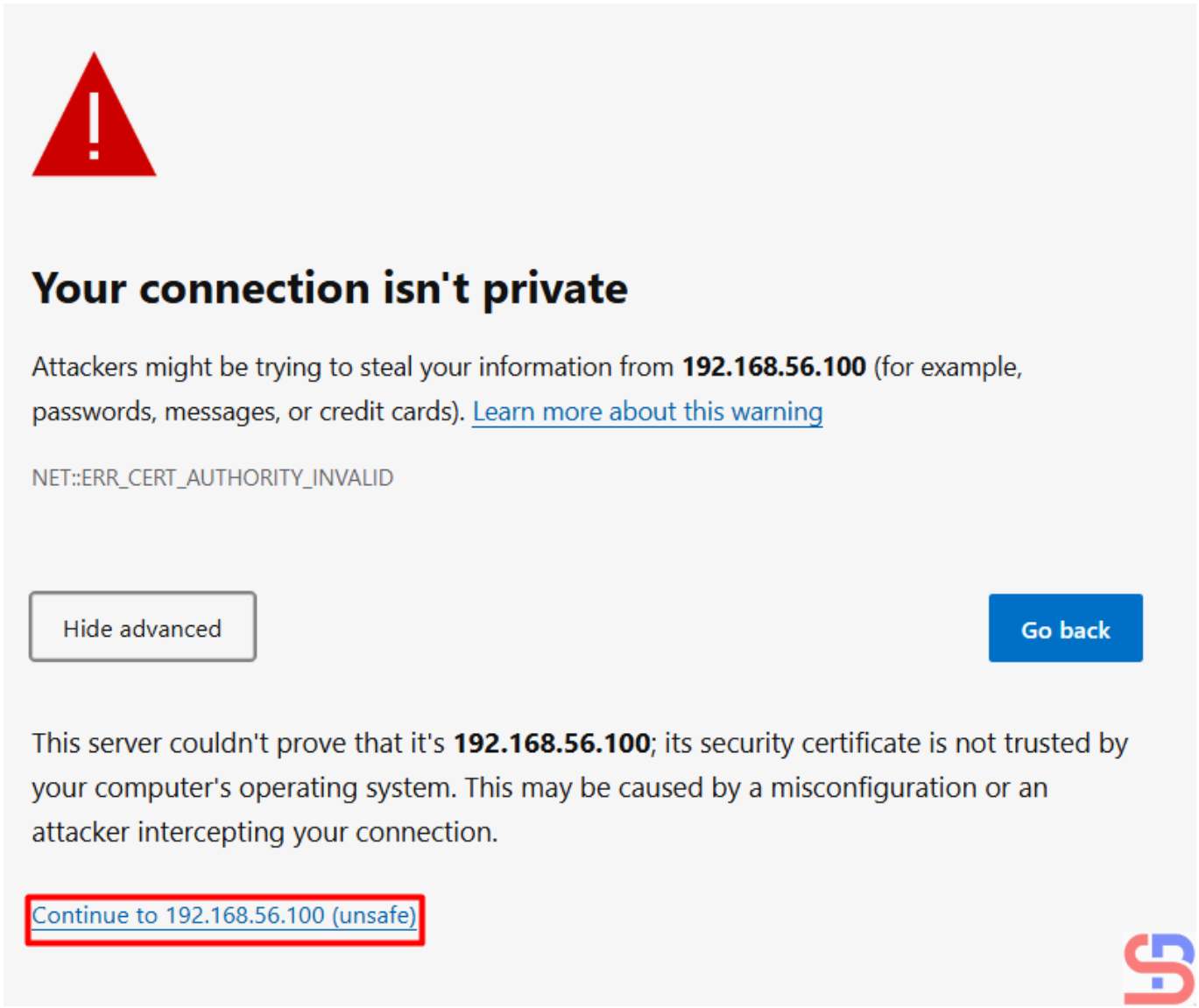
https://your_IP_server:9443

There will be an image like below:



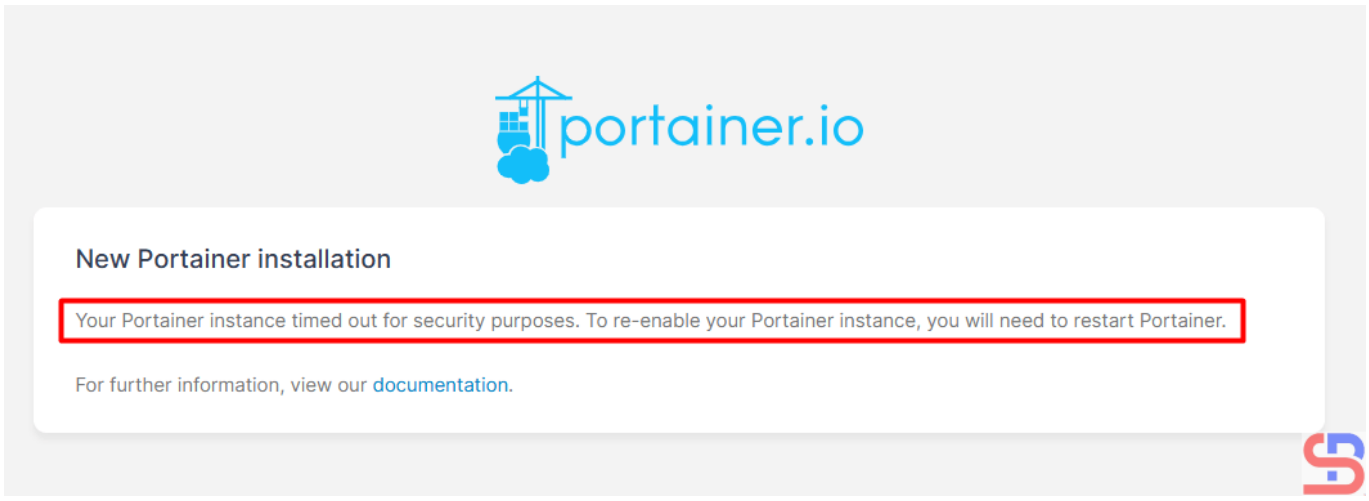
Click Advanced

A picture similar to the one below will appear when you click the **Advanced** button:



Click the unsafe link

Click the **unsafe** link in your browser, and then there will be an image like below:

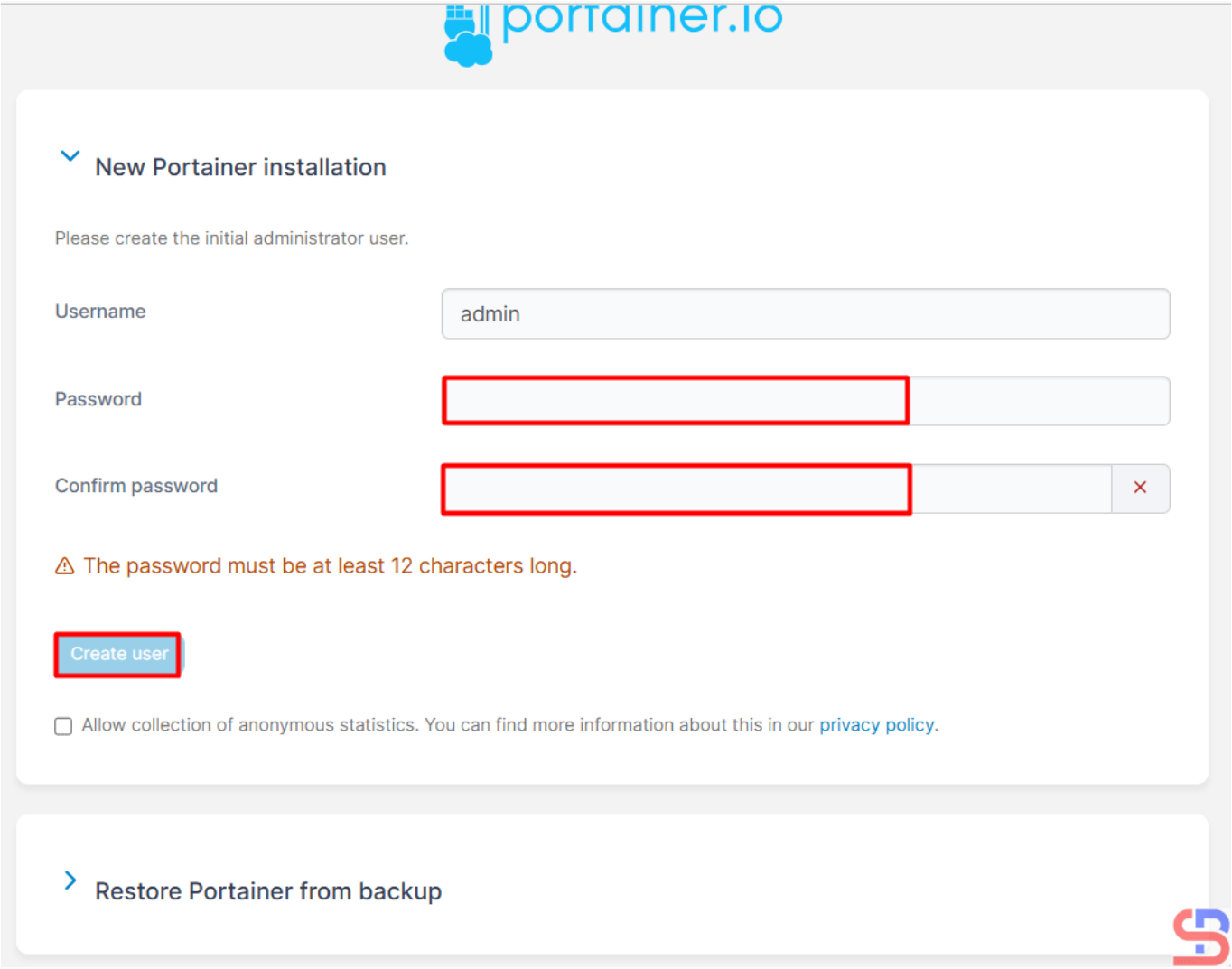


Restart Portainer

If you have an error like the picture above, then restart Portainer by running the command below:

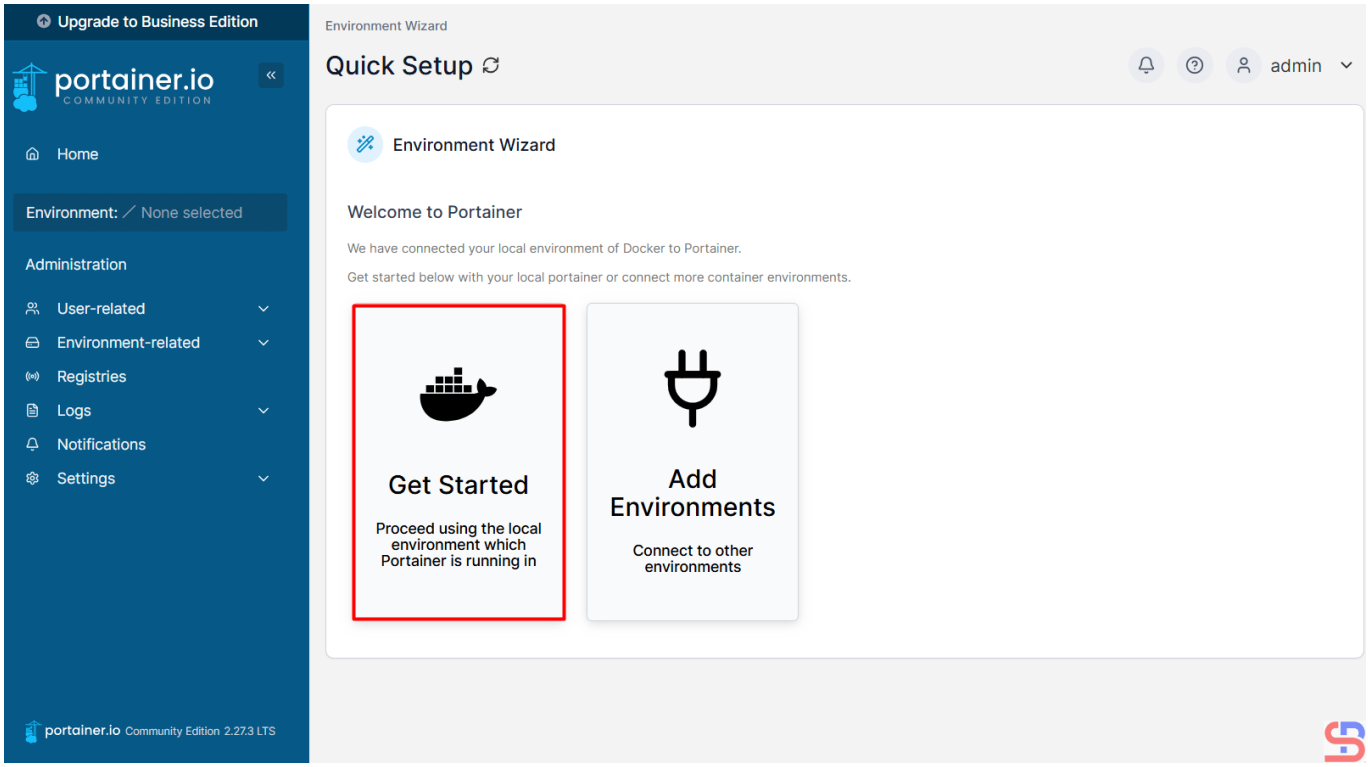
```
docker restart portainer
```

Enter the desired name and password, then click the **Create user** button, and you will see an image below:



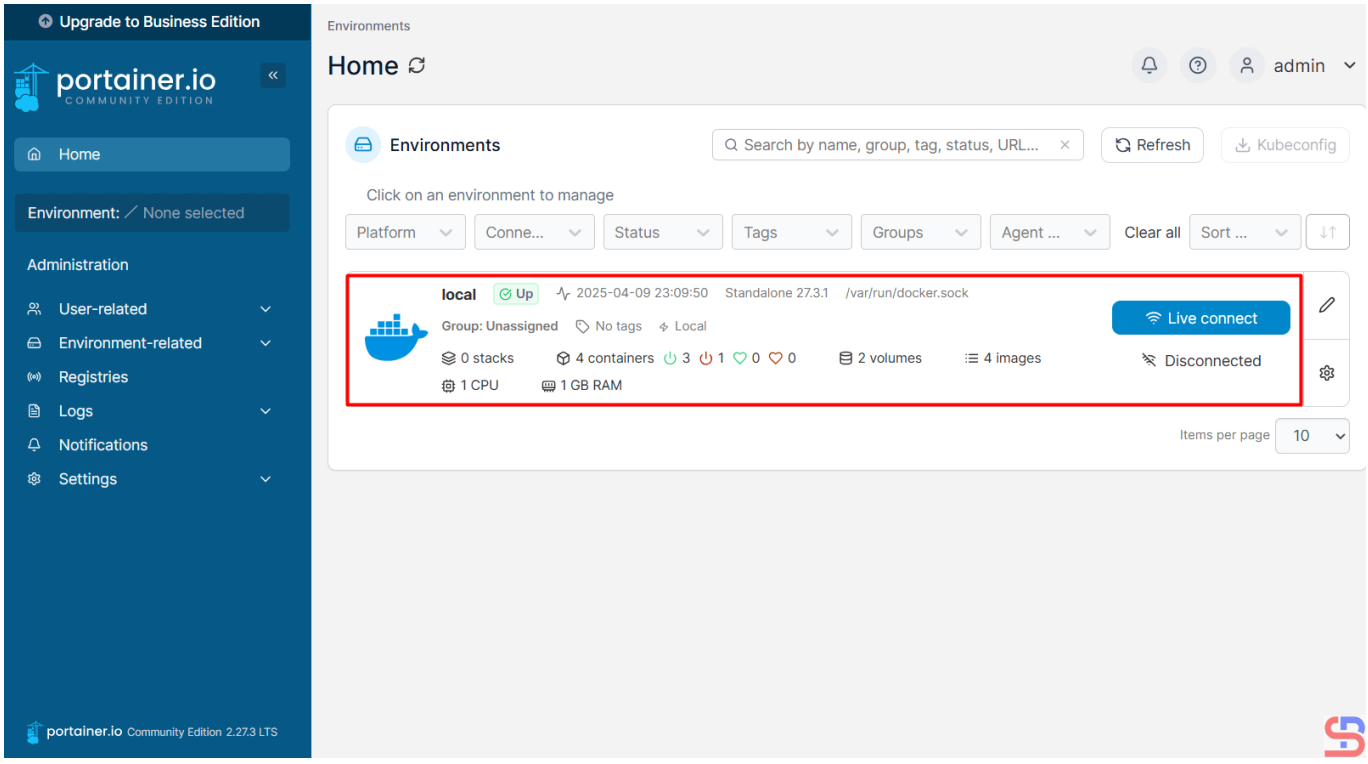
Write the username and password

The Portainer dashboard will appear. Click the **Get Started** box like in the above image, and there will be an image below:



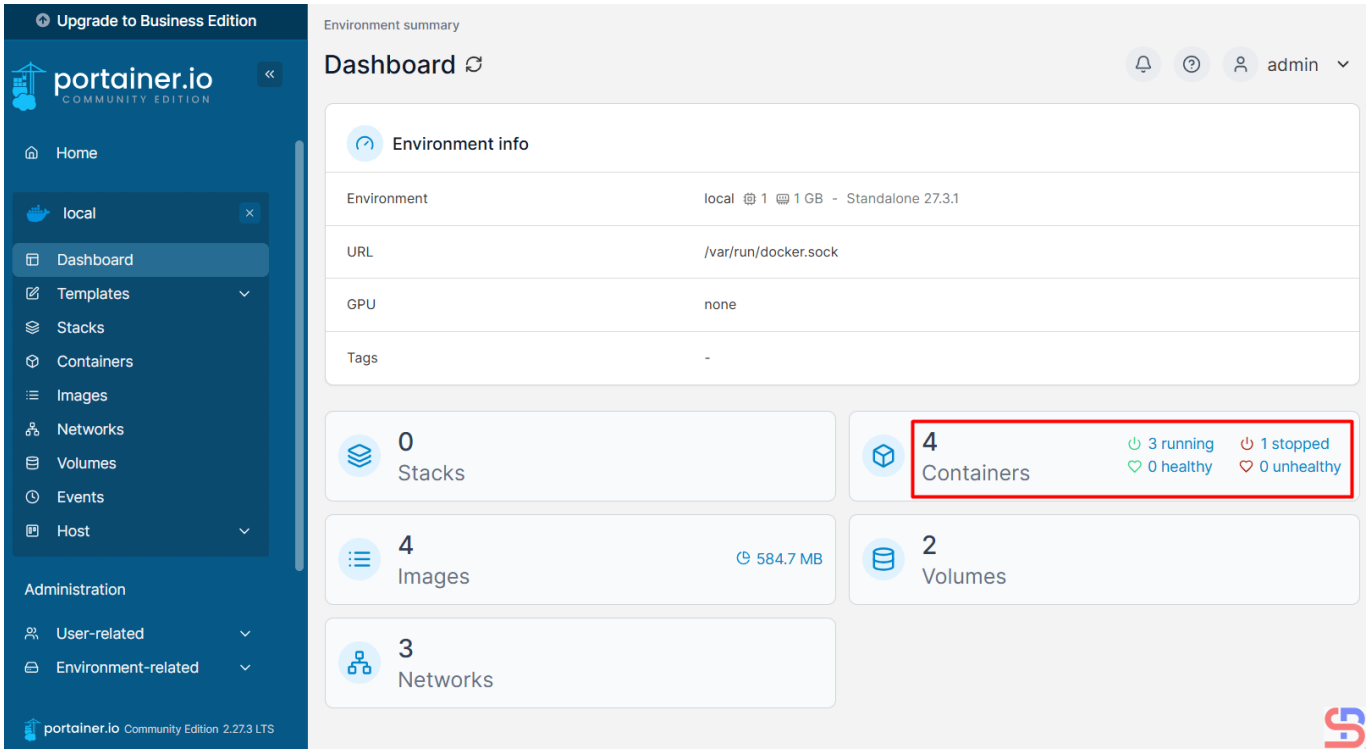
Click the Get Started box

There will be an image below:



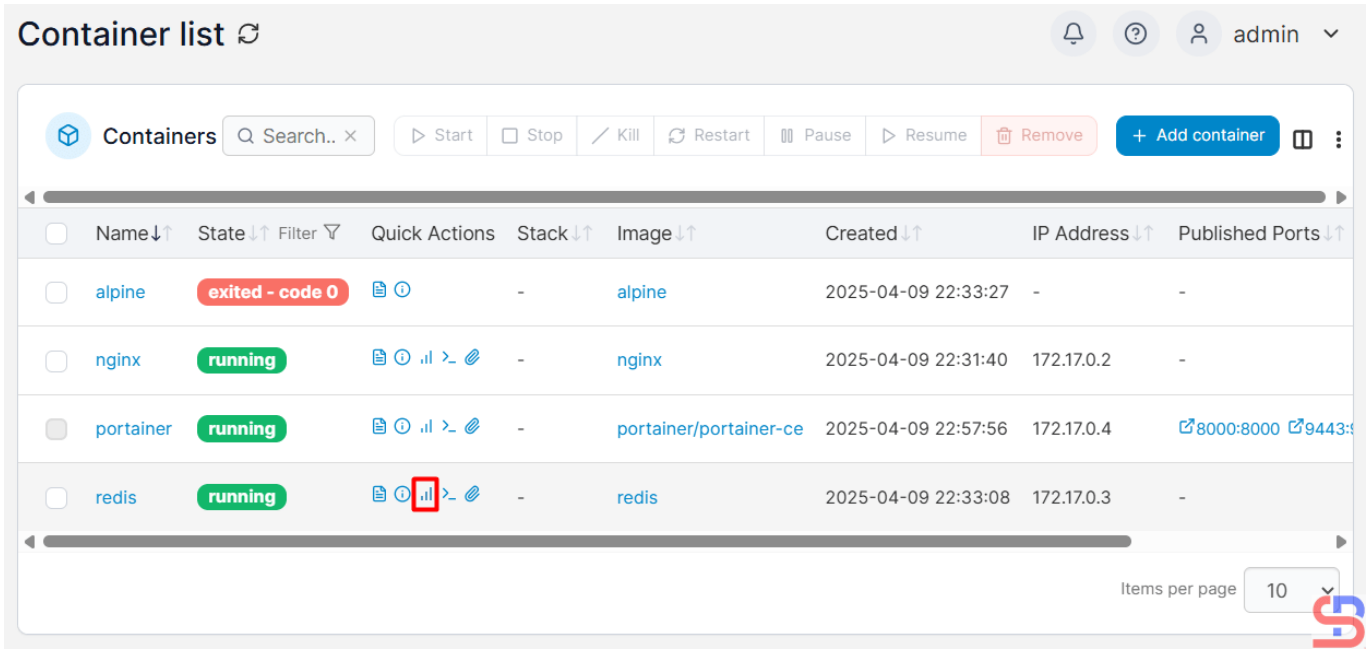
The Portainer dashboard

Click on the red box, and there will be an image below:



The information about the container(s) in Docker

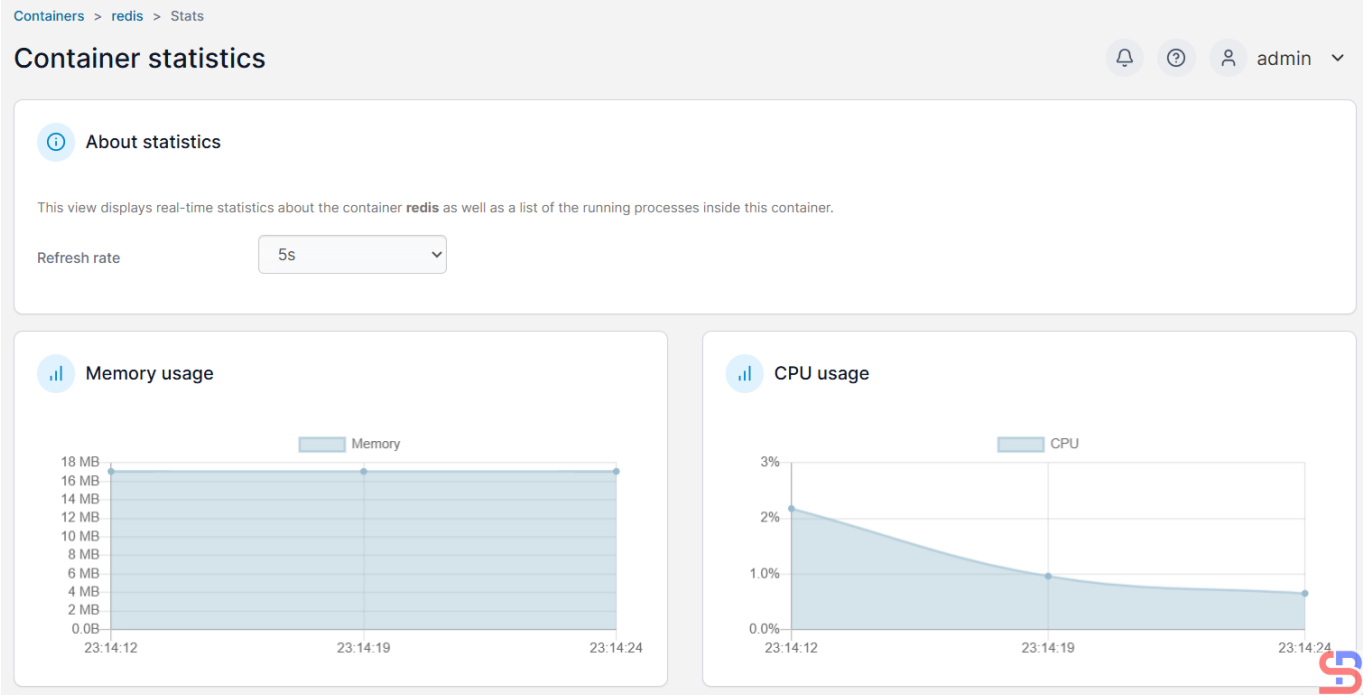
I have 4 containers in my server, but I want to detail each container, so I click in the red box, and there will be an image below:



The detailed information of each container

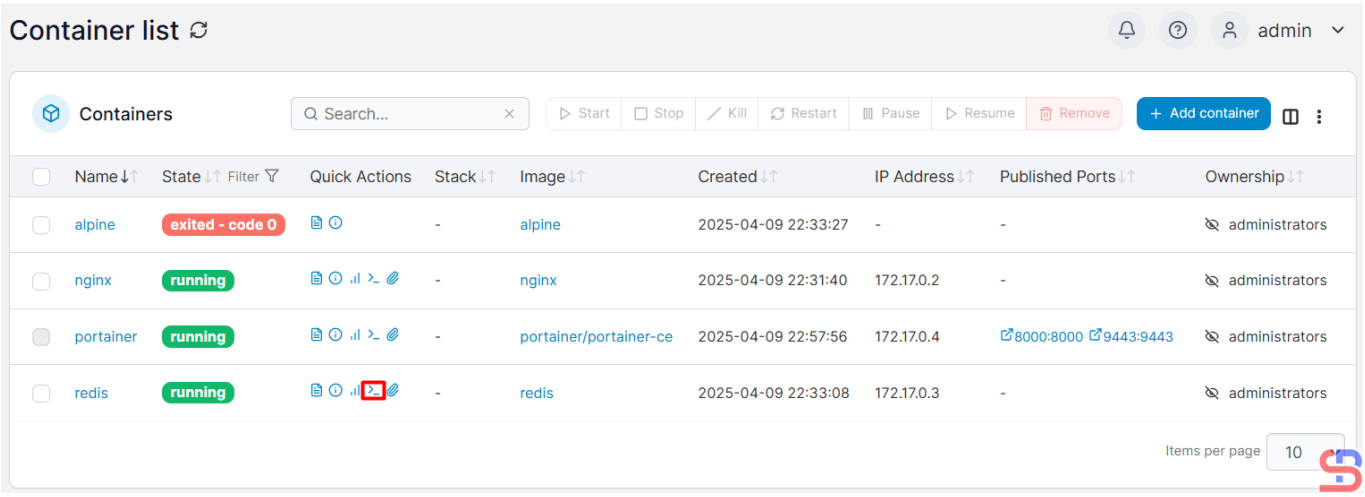
If I want to display the resource of the Redis instance, click the icon in the red box, and there will be a display

below:



The resource is displayed in a container

If I want to access a container, I click the icon like in the red box:

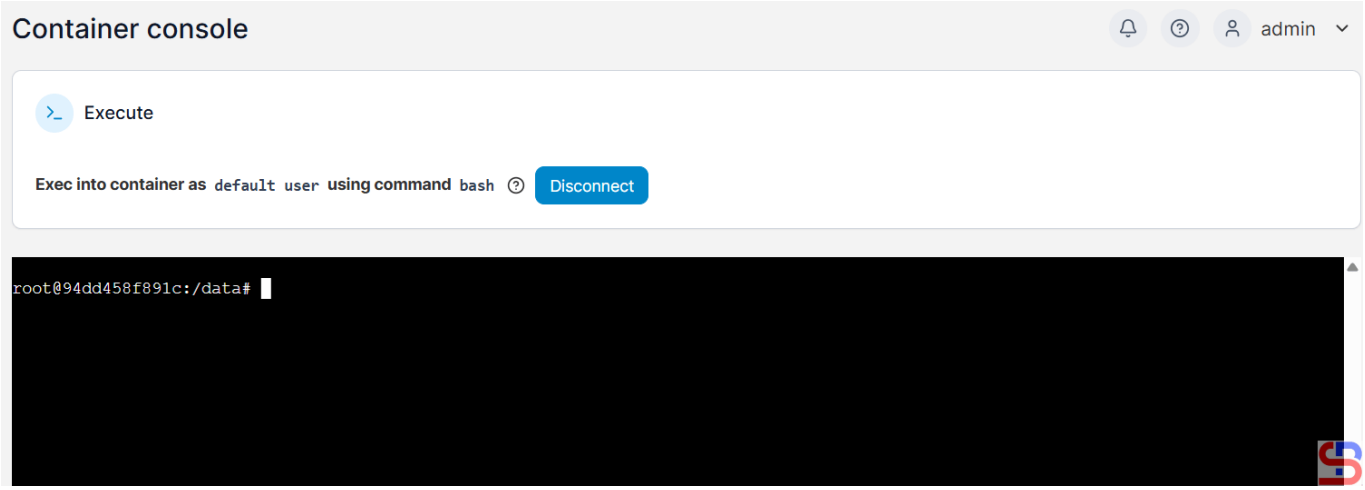


Click the icon to access the container

There will be a display like in the image below:

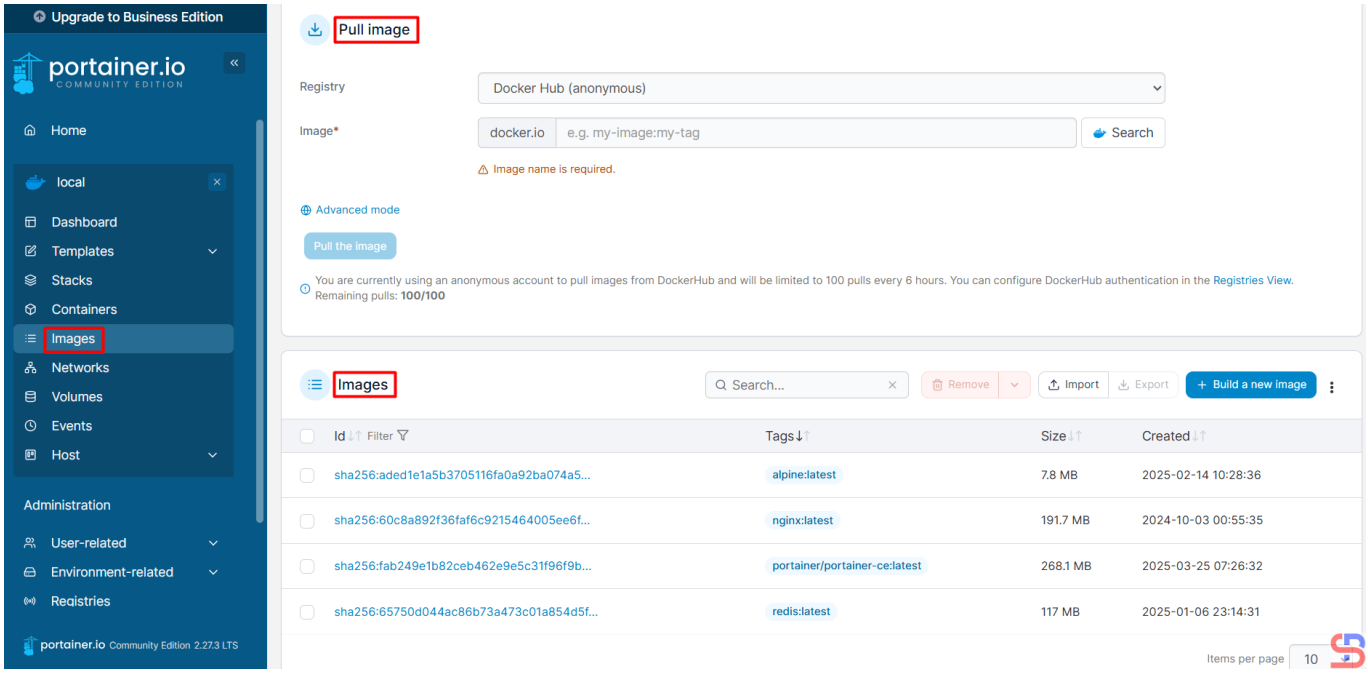
Click the Connect button after you choose the options

Select the command used in the container and select the desired user. After that, click the **Connect** button, and there will image like in the image below:



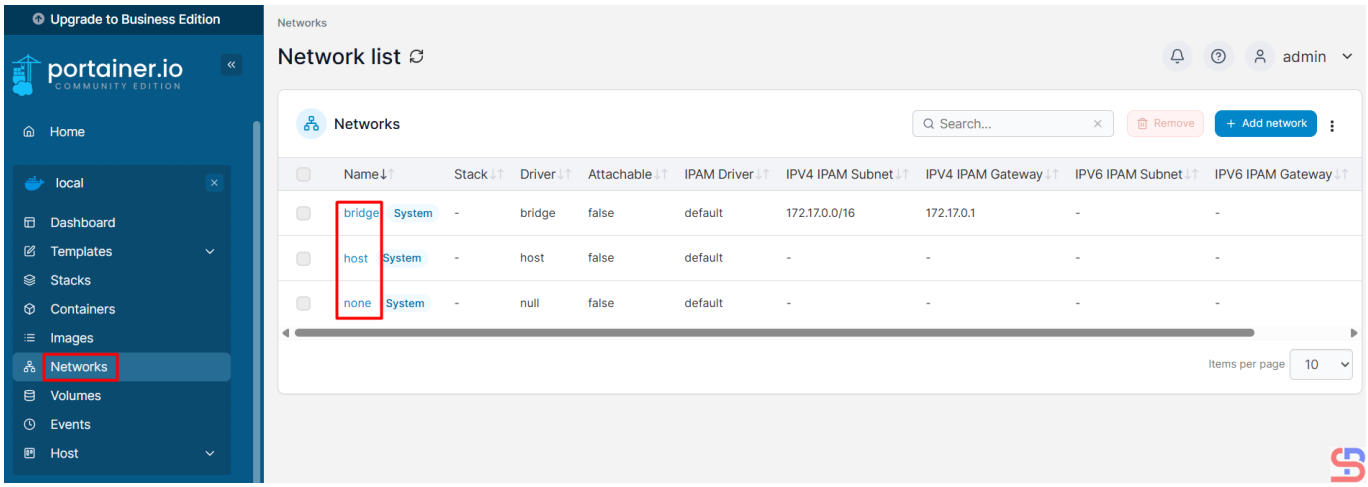
Access to the container

You can access the inside of the container and give the Linux command from your browser to the container. From this tool, you can see the images in your Docker when you click Images, like in the image below:



Display the Images

You can display the [Volume](#) in Docker after you click the Volumes, like in the image below:



Display the Volumes

Note

If you want to monitor Docker on another server using Portainer, you have to install the agent using the command below:

```
curl -L https://downloads.portainer.io/agent-stack.yml -o agent-stack.yml && docker stack deploy --compose-file=agent-stack.yml portainer-agent
```

References

youtube.dimas-maryanto.com
docs.portainer.io
phoenixnap.com
musaamin.web.id
letscloud.io

How to Back up and Restore Docker Image(s)?

written by sysadmin | 21 May 2025

By default, if you want to create a Docker container on your server, you can download the required image directly using the **docker pull** command. But sometimes, there are some cases where you cannot download the image directly from the internet, and you have to back up the existing image and then restore the image to a server.

Problem

How to back up and restore Docker image(s)?

Solution

I have a server that, due to security issues, cannot be connected to the internet, while on the server, many applications run using Docker. Because it cannot directly download the Docker image, I have to download the Docker image on a server that is connected to the internet, and then the image will be installed on this server.

A. Backup Docker image

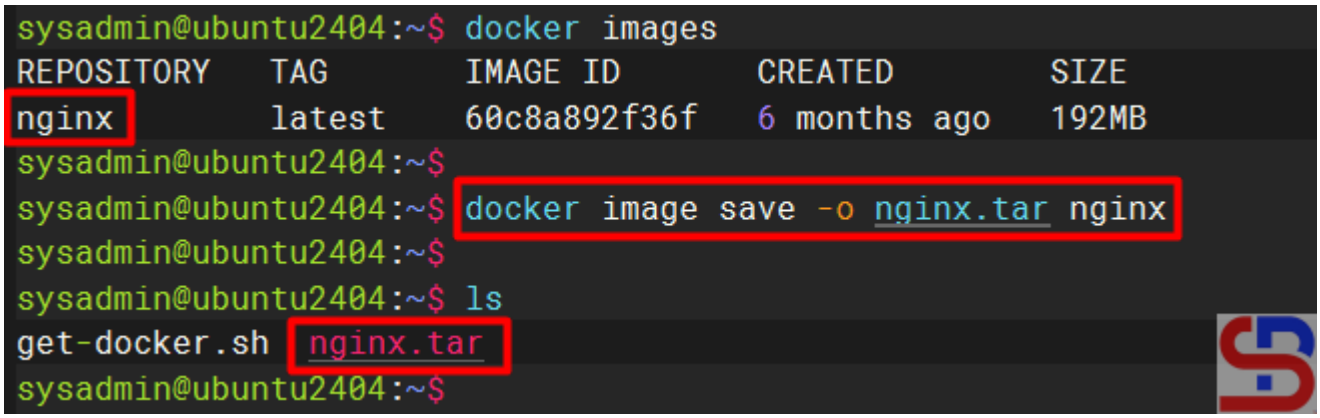
To back up a Docker image, use the format below:

```
docker image save -o image_name.tar image_name:tag
```

For example, if you want to back up the nginx image, then use the command below:

```
docker image save -o nginx.tar nginx
```

Wait until the process is complete, and if it is finished, the backup image file will be formed as shown below:



Back up one Docker image

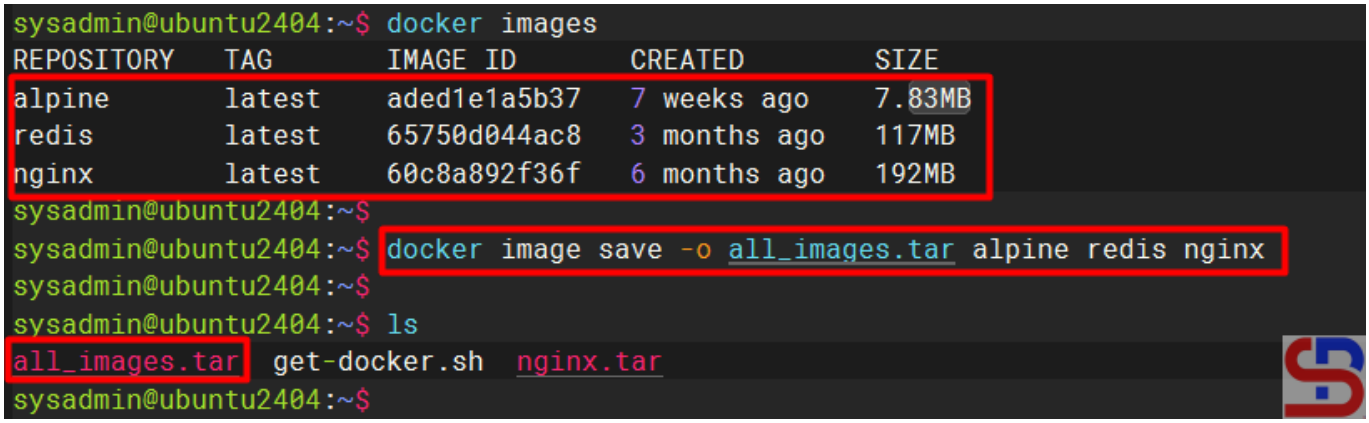
But use the format below if you want to back up more than one image:

```
docker image save -o image1_name.tar image2_name:tag ...
```

I have more than one Docker image on my server, so I want to back up all the images so I can run the images on my other server, which is not connected to the internet. Then use the command below to back up the docker image of more than one Docker image:

```
docker image save -o all_images.tar alpine redis nginx
```

And the backup image file should be formed according to the image below:



Backup more than one docker image

B. Restore Image Docker

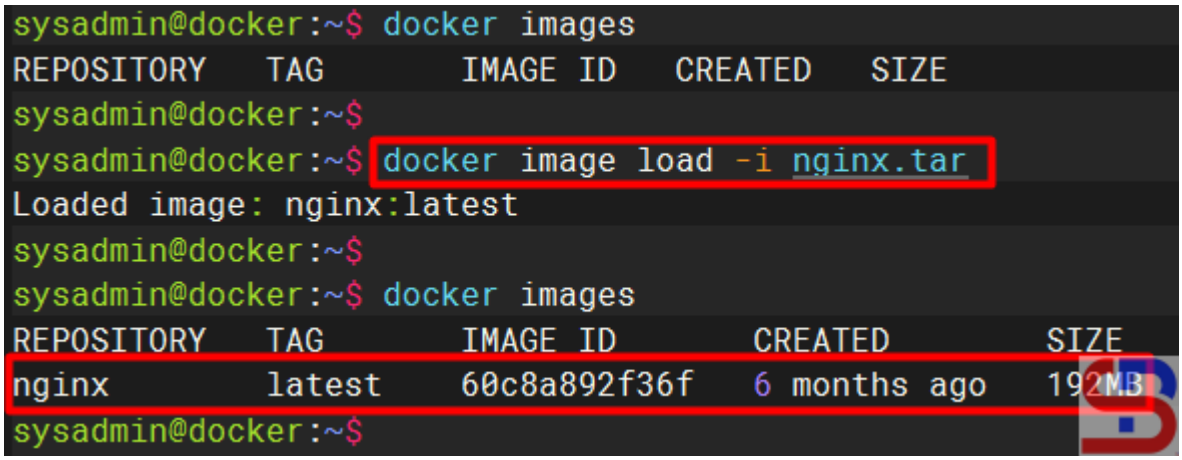
After you back up the Docker image, move your Docker image to the desired server. To restore the Docker image, use the format below:

```
docker image load -i image_name.tar
```

So I restored the Docker image backup file using the command below:

```
docker image load -i nginx.tar
```

Then the Nginx image will be restored on the server as shown below:

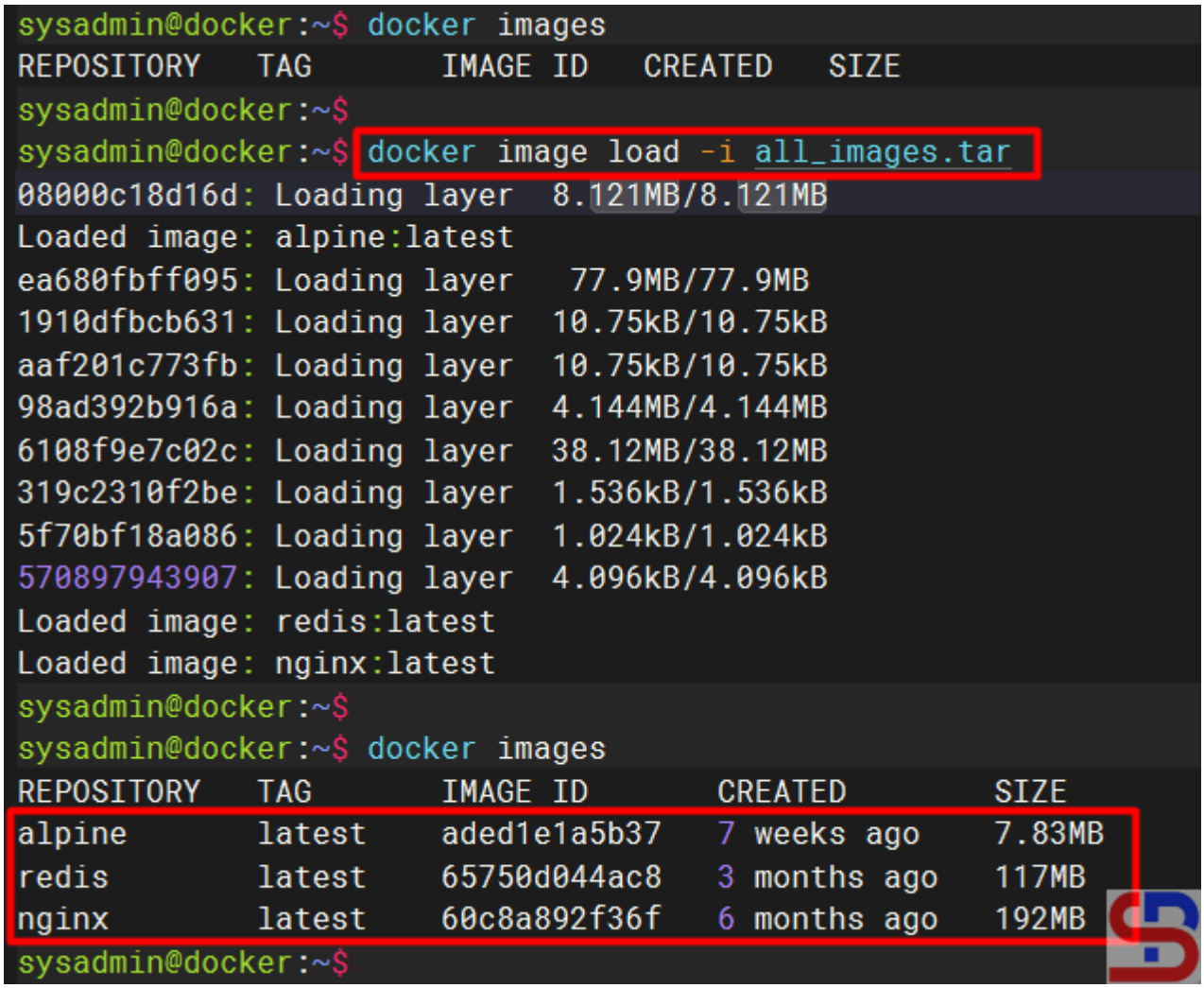


Restore one Docker image

With the same command, you can also restore more than one image using the command below:

```
docker image load -i all_images.tar
```

Then all Docker images will be restored on that server, like in the image below:



Restore more than one Docker image

Note

You can use the command below to back up the Docker image using the format below:

```
docker save image_name | gzip -c > image_name.tgz
```

So if you want to back up the Nginx image, use the command

below:

```
docker save nginx | gzip -c > nginx.tgz
```

The advantage of using this command is that the size of the backup file is much smaller than using the previous command, as shown in the image below:

```
sysadmin@docker:~$ ls -lh
total 496M
-rw------- 1 sysadmin sysadmin 310M Apr  9 14:11 all_images.tar
-rw-rw-r-- 1 sysadmin sysadmin  22K Nov 24 10:03 get-docker.sh
-rw------- 1 sysadmin sysadmin 187M Apr  9 09:53 nginx.tar
sysadmin@docker:~$
sysadmin@docker:~$ docker save nginx | gzip -c > nginx.tgz
sysadmin@docker:~$ docker save nginx redis alpine | gzip -c > all_images.tgz
sysadmin@docker:~$
sysadmin@docker:~$ ls -lh
total 677M
-rw------- 1 sysadmin sysadmin 310M Apr  9 14:11 all_images.tar
-rw-rw-r-- 1 sysadmin sysadmin 114M Apr  9 14:44 all_images.tgz
-rw-rw-r-- 1 sysadmin sysadmin  22K Nov 24 10:03 get-docker.sh
-rw------- 1 sysadmin sysadmin 187M Apr  9 09:53 nginx.tar
-rw-rw-r-- 1 sysadmin sysadmin  68M Apr  9 14:34 nginx.tgz
sysadmin@docker:~$
```

Back up the Docker image using another command

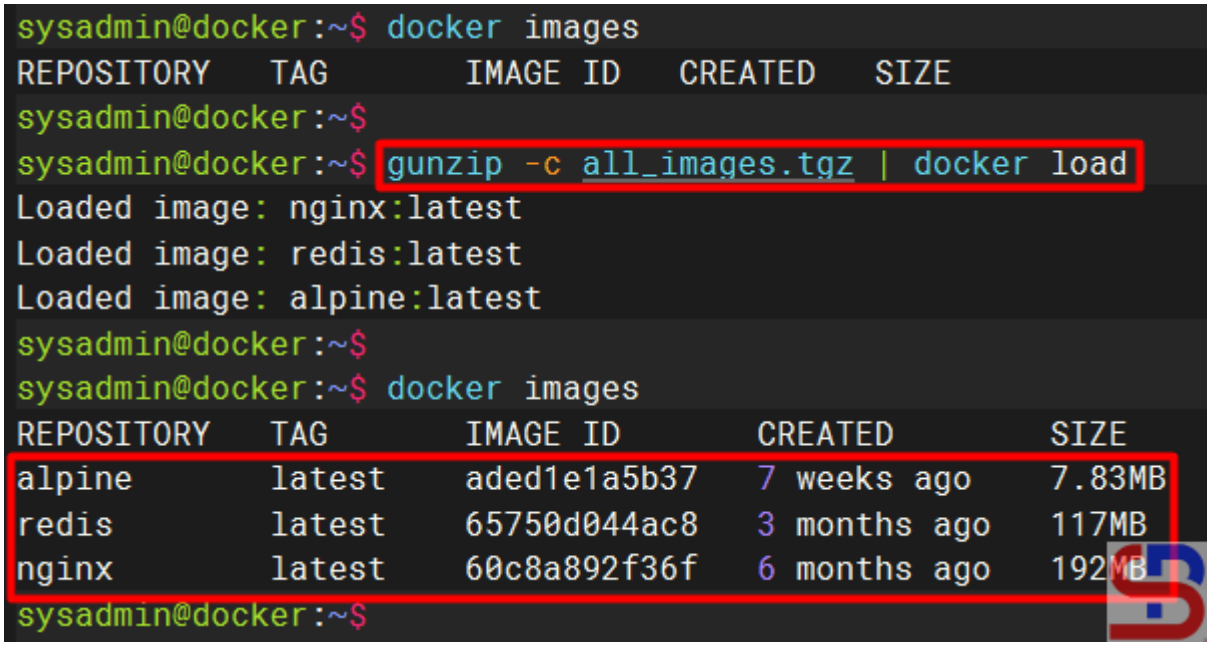
To restore, use the format below:

```
gunzip -c filename.tgz | docker load
```

So if you want to restore more than one image, use the command below:

```
gunzip -c all_images.tgz | docker load
```

And the Docker image will be restored on the server.



Restore the backup Docker image using another command

References

youtube.dimas-maryanto.com
youtube.com
docs.docker.com
stackoverflow.com

How to Manage a Container in Docker?

written by sysadmin | 21 May 2025

The previous article explained how to install Docker on Linux. This article will explain how to manage a container in Docker.

Problem

How to manage a container in Docker?

Solution

To manage a container in Docker, you have to remember basic Docker commands. And here are the basic Docker commands:

1. Search for container images

To run containers in Docker, we need a Docker image. A Docker image is an immutable (unchangeable) file that contains the source code, libraries, dependencies, tools, and other files needed for an application to run. The place to store Docker images is known as the Docker registry, which by default uses the Docker Hub located at **hub.docker.com**. If you are looking for a container image in Docker, use the format below:

```
docker search container_name
```

For example, if you want to find an nginx image, then use the command below:

```
docker search nginx
```

```
sysadmin@ubuntu2404:~$ docker search nginx
NAME                                DESCRIPTION                                     STARS     OFFICIAL
nginx                               Official build of Nginx.                        20651     [OK]
nginx/nginx-ingress                 NGINX and  NGINX Plus Ingress Controllers fo…  100
nginx/nginx-prometheus-exporter     NGINX Prometheus Exporter for NGINX and NGIN…  48
nginx/unit                          This repository is retired, use the Docker o…  65
nginx/nginx-ingress-operator        NGINX Ingress Operator for NGINX and NGINX P…  2
nginx/nginx-quic-qns                NGINX QUIC interop                              1
nginx/unit-preview                  Unit preview features                           0
nginx/nginxaas-loadbalancer-kubernetes                                              0
bitnami/nginx                       Bitnami container image for NGINX               196
ubuntu/nginx                        Nginx, a high-performance reverse proxy & we…  127
bitnamicharts/nginx                 Bitnami Helm chart for NGINX Open Source        0
rancher/nginx                                                                       2
kasmweb/nginx                       An Nginx image based off nginx:alpine and in…  8
linuxserver/nginx                   An Nginx container, brought to you by LinuxS…  227
redash/nginx                        Pre-configured nginx to proxy linked contain…  3
dtagdevsec/nginx                    T-Pot Nginx                                     0
vmware/nginx                                                                        2
paketobuildpacks/nginx                                                              0
chainguard/nginx                    Build, ship and run secure software with Cha…  4
gluufederation/nginx                 A customized NGINX image containing a consu…  1
intel/nginx                                                                         0
droidwiki/nginx                                                                     0
circleci/nginx                      This image is for internal use                  2
corpusops/nginx                     https://github.com/corpusops/docker-images/     1
antrea/nginx                        Nginx server used for Antrea e2e testing        0
sysadmin@ubuntu2404:~$
```

Searching the nginx image

2. Download the Docker image

To download the Docker image, use the following format:

```
docker image pull image_name:tag_version
```

where the tag_version is the version of the image, and if you don’t write the tag, it is considered that you want to install the latest version of the image. For example, if you want to download the newest version of the nginx image, use the command:

```
docker image pull nginx
```

```
sysadmin@ubuntu2404:~$ docker image pull nginx
Using default tag: latest
latest: Pulling from library/nginx
7cf63256a31a: Pull complete
bf9acace214a: Pull complete
513c3649bb14: Pull complete
d014f92d532d: Pull complete
9dd21ad5a4a6: Pull complete
943ea0f0c2e4: Pull complete
103f50cb3e9f: Pull complete
Digest: sha256:9d6b58feebd2dbd3c56ab5853333d627cc6e281011cfd6050fa4bcf2072c9496
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
sysadmin@ubuntu2404:~$
```

Download the nginx image

But if you want to download nginx with a certain version, for example, version 1.27.2, then use the command:

```
docker image pull nginx:1.27.2
```

```
sysadmin@ubuntu2404:~$ docker pull nginx:1.27.2
1.27.2: Pulling from library/nginx
2d429b9e73a6: Pull complete
9b1039c85176: Pull complete
9ad567d3b8a2: Pull complete
773c63cd62e4: Pull complete
1d2712910bdf: Pull complete
4b0adc47c460: Pull complete
171eebbdf235: Pull complete
Digest: sha256:bc5eac5eafc581aeda3008b4b1f07ebba230de2f27d47767129a6a905c84f470
Status: Downloaded newer image for nginx:1.27.2
docker.io/library/nginx:1.27.2
sysadmin@ubuntu2404:~$
```

Download the nginx with a certain version

3. List the Docker image(s)

To display the Docker image that you have downloaded, use the command below:

```
docker image ls
```

```
sysadmin@ubuntu2404:~$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    b52e0b094bc0   5 weeks ago   192MB
sysadmin@ubuntu2404:~$
```

List the Docker images

Or you can use the command below:

```
docker images
```

```
sysadmin@ubuntu2404:~$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
nginx        latest    60c8a892f36f   6 months ago   192MB
sysadmin@ubuntu2404:~$
```

List the Docker images

4. Create a container

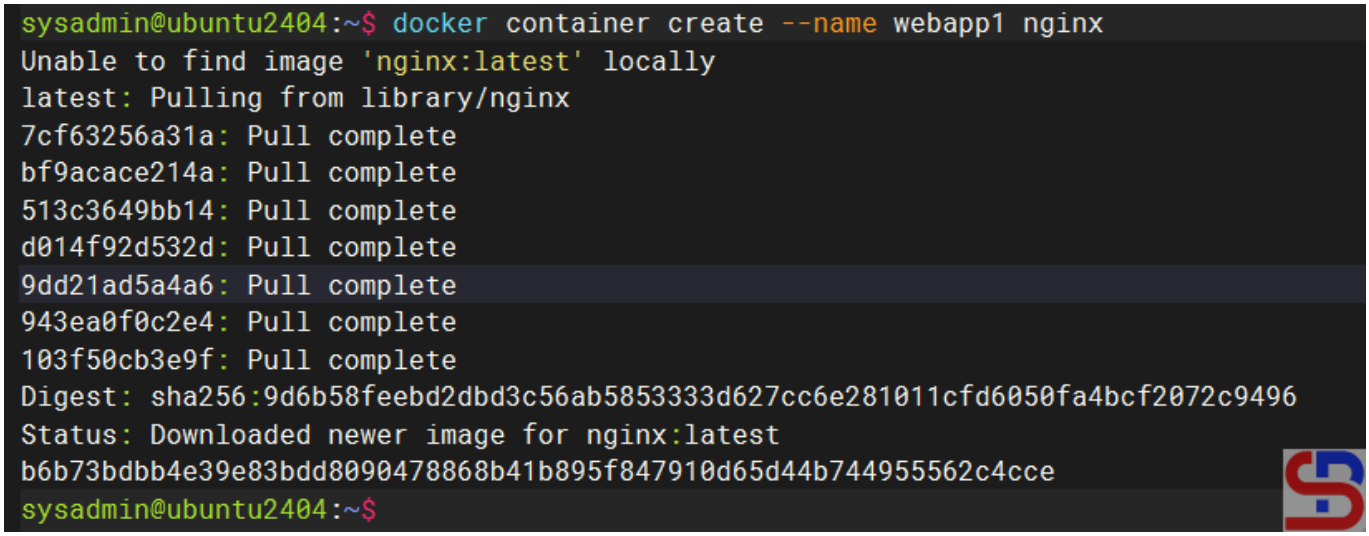
You can create the container using the format:

```
docker container create –name container_name image_name:tag
```

For example, if you want to create a container with the name webapp1, which contains the nginx application, then use the command below:

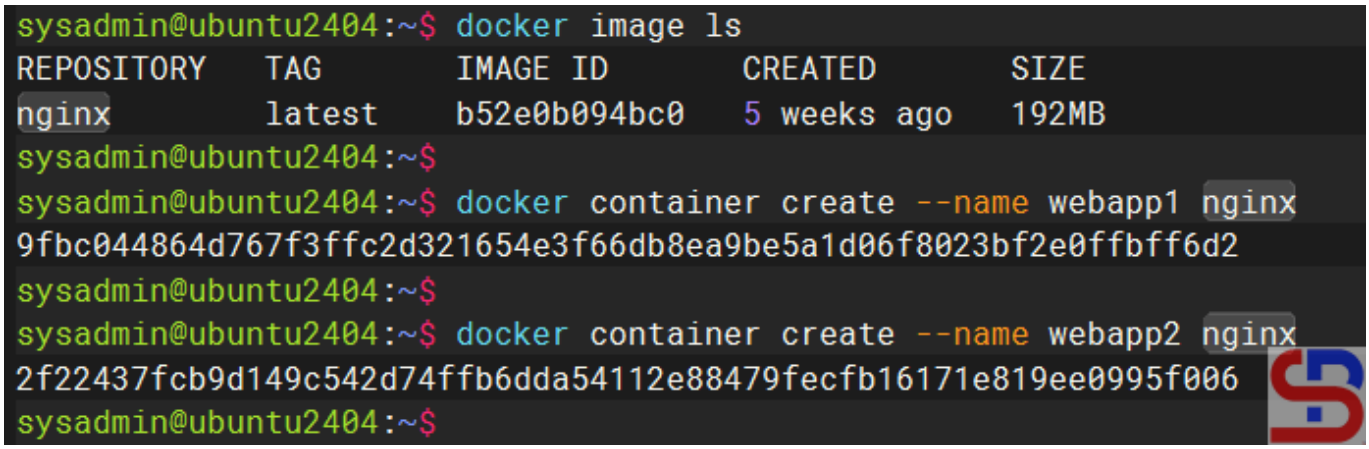
```
docker container create --name webapp1 nginx
```

When you use this command, Docker will first check whether the nginx image is on the server. If the image is not on the server, then Docker will download the nginx image, and after that, it will create an nginx container, and the image will remain on your server, as shown in the image below:



Create the container

You can use an image to create multiple containers as long as the container names are different, as in the image below:



Create the containers with 1 image

5. List the status of the container(s)

To display the container status, you can use the command:

```
docker ps
```

```
sysadmin@ubuntu2404:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
sysadmin@ubuntu2404:~$
```

List the status of running Docker

Maybe you are confused about why there is no container status displayed, even though you have made 2 containers before. Remember that the **docker ps** command only displays the ongoing container status. While the 2 containers you made had not been running, you just made a container. If you want to display all container statuses, use the command below:

```
docker ps -a
```

```
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS    PORTS     NAMES
2f22437fcb9d   nginx     "/docker-entrypoint.…"   6 minutes ago   Created             webapp2
9fbc044864d7   nginx     "/docker-entrypoint.…"   6 minutes ago   Created             webapp1
sysadmin@ubuntu2404:~$
```

List all container statuses in Docker

6. Turn on the container

To turn on a container, you can use the format:

```
docker container start container_id/container_name
```

Usually, I use container_name instead of container_id because it’s easier to memorize, so I run the following command:

```
docker container start webapp1
```

```
sysadmin@ubuntu2404:~$ docker container start webapp1
webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
b6b73bdbb4e3   nginx     "/docker-entrypoint.…"   31 minutes ago   Up 5 seconds    80/tcp    webapp1
sysadmin@ubuntu2404:~$
```

Turn on the container

7. Pause the container

You can pause a container with the following format:

```
docker container pause container_id/container_name
```

So, you can use the command below to pause the container:

```
docker container pause webapp1
```

```
sysadmin@ubuntu2404:~$ docker container pause webapp1
webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                     PORTS     NAMES
b6b73bdbb4e3   nginx     "/docker-entrypoint.…"   31 minutes ago   Up 53 seconds (Paused)     80/tcp    webapp1
sysadmin@ubuntu2404:~$
```

Pause the container

To resume the container, use the following format:

```
docker containers unpause container_id/container_name
```

You can use the command below to resume the container:

```
docker container unpause webapp1
```

```
sysadmin@ubuntu2404:~$ docker container unpause webapp1
webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS              PORTS     NAMES
b6b73bdbb4e3   nginx     "/docker-entrypoint.…"   32 minutes ago   Up About a minute   80/tcp    webapp1
sysadmin@ubuntu2404:~$
```

Resume the container

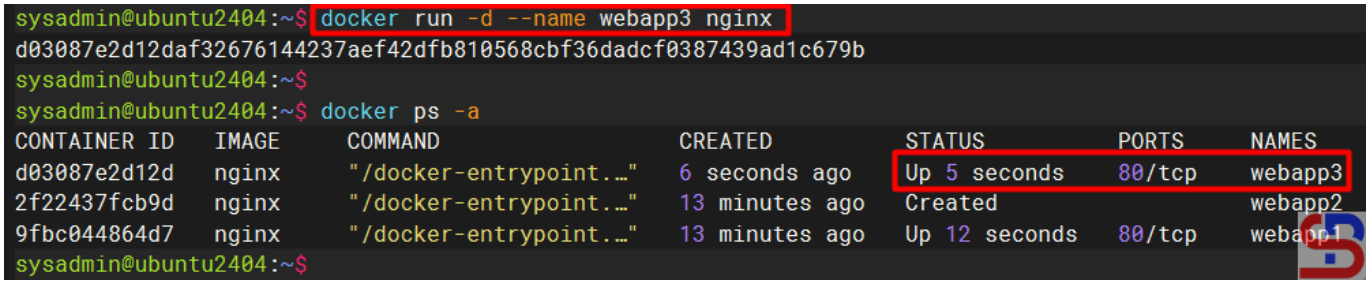
8. Run a container with a single command

As explained above, if you want to run a container, you have to download the image first, create a container, and turn on the container (see numbers 2, 4, and 6). There is a command that can summarize the three commands above, using the format below:

```
docker run -d --name container_id/container_name image_name:tag
```

where the **-d option** is to run the container in the background. So if you want to run a container with the name webapp2, which contains the nginx application, then use the command below:

```
docker run -d --name webapp3 nginx
```

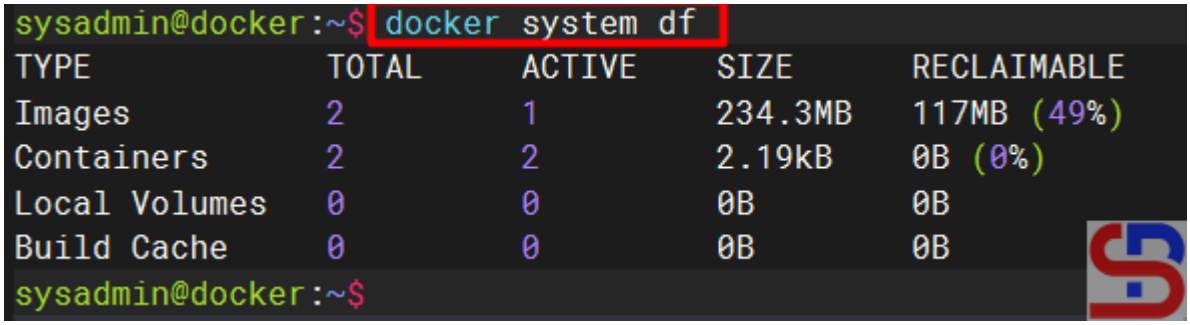


Run the container

9. Display the size of Docker

To display how large Docker is installed on your server, use the command below:

```
docker system df
```



Display the size of Docker

To display a Docker size in detail, use the command below:

```
docker system df -v
```

```
sysadmin@docker:~$ docker system df -v
Images space usage:

REPOSITORY   TAG       IMAGE ID       CREATED        SIZE      SHARED SIZE   UNIQUE SIZE   CONTAINERS
nginx        latest    4cad75abc83d   2 months ago   192MB     74.83MB       117.2MB       2
redis        latest    65750d044ac8   3 months ago   117MB     74.83MB       42.21MB       0

Containers space usage:

CONTAINER ID   IMAGE     COMMAND                  LOCAL VOLUMES   SIZE      CREATED          STATUS          NAMES
d8982eca0840   nginx     "/docker-entrypoint.…"   0               1.09kB    4 minutes ago    Up 4 minutes    webapp1
8d609d92bcc7   nginx     "/docker-entrypoint.…"   0               1.09kB    11 minutes ago   Up 11 minutes   nginx

Local Volumes space usage:

VOLUME NAME   LINKS     SIZE

Build cache usage: 0B

CACHE ID   CACHE TYPE   SIZE      CREATED   LAST USED   USAGE     SHARED
sysadmin@docker:~$
```

Display the size of the Docker

10. Display logs

To display logs of the running container to check something, follow the format below:

```
docker container logs container_id/container_name
```

So, run the command below to check the logs of your container:

```
docker container logs webapp1
```

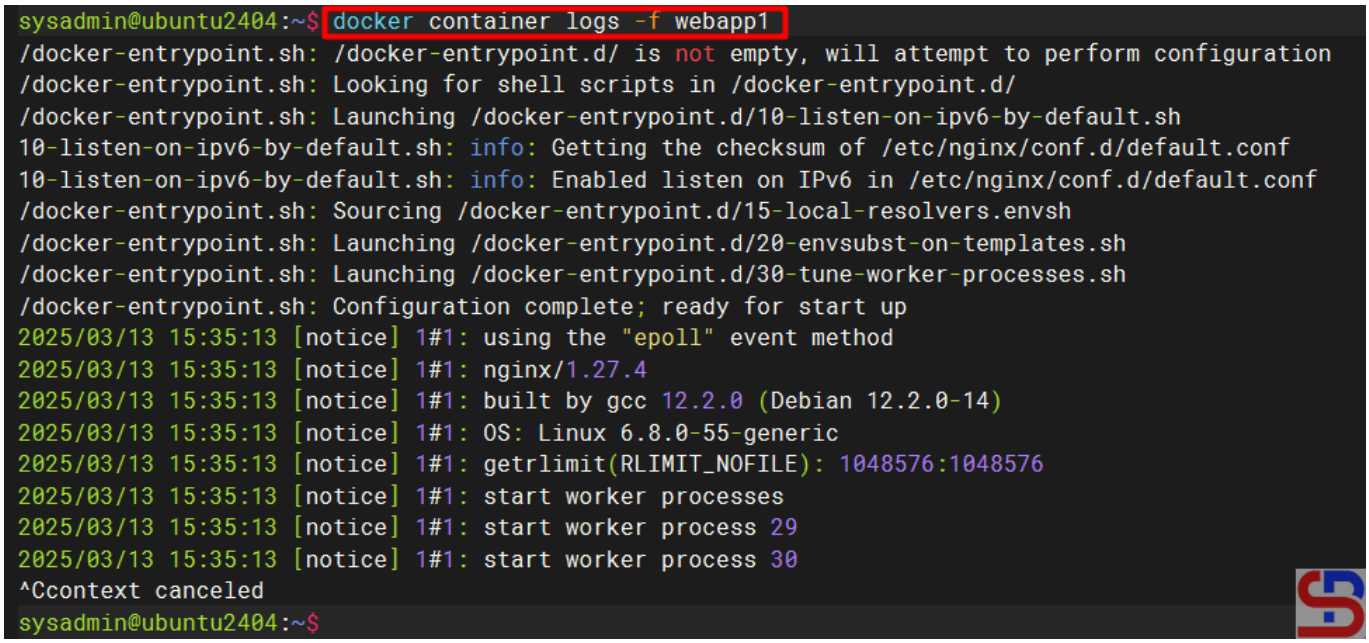
```
sysadmin@ubuntu2404:~$ docker container logs webapp1
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/03/13 15:35:13 [notice] 1#1: using the "epoll" event method
2025/03/13 15:35:13 [notice] 1#1: nginx/1.27.4
2025/03/13 15:35:13 [notice] 1#1: built by gcc 12.2.0 (Debian 12.2.0-14)
2025/03/13 15:35:13 [notice] 1#1: OS: Linux 6.8.0-55-generic
2025/03/13 15:35:13 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/03/13 15:35:13 [notice] 1#1: start worker processes
2025/03/13 15:35:13 [notice] 1#1: start worker process 29
2025/03/13 15:35:13 [notice] 1#1: start worker process 30
sysadmin@ubuntu2404:~$
```

Display logs in the container

If you want to display real-time logs of the container, give an option **-f** like in the below command:

```
docker container logs -f webapp1
```

Press **Ctrl-C** to exit the log.



Display real-time logs in the container

11. Inspect the container

To display detailed information about a container, use the following format:

```
docker inspect container_name/container_id
```

So, if you want to see the detailed information about the container that you created before, use the command below:

```
docker inspect webapp1
```

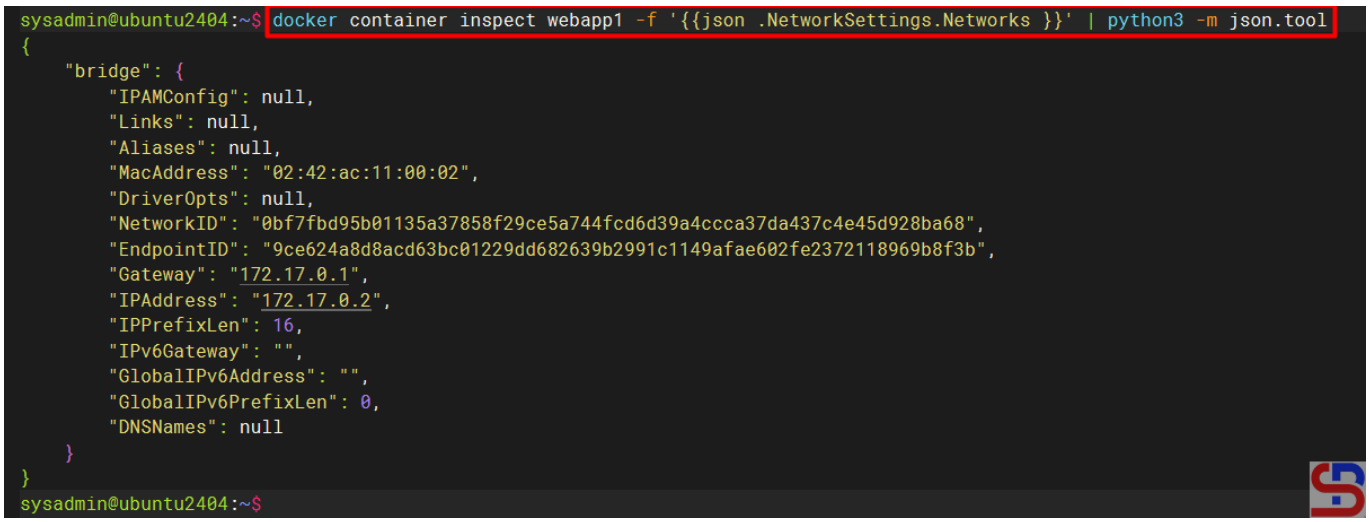
Inspect the container

If you only want to display specific items when running the inspect command, use the following format:

```
docker container inspect container_name/container_id -f '{{json .the item_you_want_to_display<.sub_item> }}' | python -m json.tool
```

So if you want to display only the network section when using the Docker inspect command, use the command below:

```
docker container inspect webapp1 -f '{{json .NetworkSettings.Networks }}' | python3 -m json.tool
```



Inspect the network of the container only

12. Stop the container(s)

To stop the container, use the format below:

```
docker container stop container_id/container_name
```

For example, if I want to stop my container, then use the command below:

```
docker container stop webapp1
```

```
sysadmin@ubuntu2404:~$ docker container stop webapp1
webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
d03087e2d12d   nginx     "/docker-entrypoint.…"   4 minutes ago    Up 4 minutes                80/tcp    webapp3
2f22437fcb9d   nginx     "/docker-entrypoint.…"   18 minutes ago   Created                               webapp2
9fbc044864d7   nginx     "/docker-entrypoint.…"   18 minutes ago   Exited (0) 3 seconds ago              webapp1
sysadmin@ubuntu2404:~$
```

Stop the container

You can stop all the containers running with the below command:

```
docker stop webapp1 webapp2
```

```
sysadmin@ubuntu2404:~$ docker container stop webapp1 webapp3
webapp1
webapp3
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
d03087e2d12d   nginx     "/docker-entrypoint.…"   6 minutes ago    Exited (0) 3 seconds ago              webapp3
2f22437fcb9d   nginx     "/docker-entrypoint.…"   20 minutes ago   Created                               webapp2
9fbc044864d7   nginx     "/docker-entrypoint.…"   20 minutes ago   Exited (0) 3 seconds ago              webapp1
sysadmin@ubuntu2404:~$
```

Stop more than one container

Or use the below command to stop all the running containers:

```
docker kill $(docker ps -q)
```

```
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
d03087e2d12d   nginx     "/docker-entrypoint.…"   9 minutes ago    Up 4 seconds    80/tcp    webapp3
2f22437fcb9d   nginx     "/docker-entrypoint.…"   23 minutes ago   Created                   webapp2
9fbc044864d7   nginx     "/docker-entrypoint.…"   23 minutes ago   Up 4 seconds    80/tcp    webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker kill $(docker ps -q)
d03087e2d12d
9fbc044864d7
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                       PORTS     NAMES
d03087e2d12d   nginx     "/docker-entrypoint.…"   9 minutes ago    Exited (137) 3 seconds ago             webapp3
2f22437fcb9d   nginx     "/docker-entrypoint.…"   23 minutes ago   Created                                webapp2
9fbc044864d7   nginx     "/docker-entrypoint.…"   23 minutes ago   Exited (137) 3 seconds ago             webapp1
sysadmin@ubuntu2404:~$
```

Stop all running containers

13. Remove the container(s)

Before you remove the container, you have to **stop the container first**. To delete a container that's already turned off, use the format below:

```
docker container rm container_id/container_name
```

Run the command below to remove the container:

```
docker container rm webapp1
```

```
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
d03087e2d12d   nginx     "/docker-entrypoint.…"   11 minutes ago   Exited (137) 2 minutes ago            webapp3
2f22437fcb9d   nginx     "/docker-entrypoint.…"   25 minutes ago   Created                               webapp2
9fbc044864d7   nginx     "/docker-entrypoint.…"   25 minutes ago   Exited (137) 2 minutes ago            webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker container rm webapp1
webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
d03087e2d12d   nginx     "/docker-entrypoint.…"   12 minutes ago   Exited (137) 2 minutes ago            webapp3
2f22437fcb9d   nginx     "/docker-entrypoint.…"   25 minutes ago   Created                               webapp2
sysadmin@ubuntu2404:~$
```

Delete the container

By default, you **can't remove a container if the container is still running**. You can use the command below to delete the container even if the container is still running, but it is not recommended:

```
docker container rm -f webapp2
```

```
sysadmin@ubuntu2404:~$ docker container start webapp2
webapp2
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker container rm webapp2
Error response from daemon: cannot remove container "/webapp2": container is running: stop the container before removing or force remove
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker container rm -f webapp2
webapp2
sysadmin@ubuntu2404:~$
```

Force delete the running container

If you have a lot of containers that are no longer used and you don't want to delete them one by one, you can use the command below to delete all the unused containers:

```
docker rm $(docker ps -a -q)
```

```
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS     NAMES
ba79cff3e0bf   nginx     "/docker-entrypoint.…"   21 seconds ago   Exited (0) 3 seconds ago              webapp2
2edbd2c033d8   nginx     "/docker-entrypoint.…"   26 seconds ago   Exited (0) 3 seconds ago              webapp1
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker rm $(docker ps -a -q)
ba79cff3e0bf
2edbd2c033d8
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
sysadmin@ubuntu2404:~$
```

Delete all the stop containers

You can also use the command below to delete all the stop containers:

```
docker container prune
```

```
sysadmin@docker:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS                      PORTS     NAMES
2fd039880269   mysql     "docker-entrypoint.s…"   5 hours ago     Exited (137) 2 hours ago              db-mysql
2a4eadafffcd   nginx     "/docker-entrypoint.…"   28 hours ago    Exited (0) 2 hours ago                webapp1
e6d61413d2af   nginx     "/docker-entrypoint.…"   29 hours ago    Exited (0) 2 hours ago                nginx
sysadmin@docker:~$
sysadmin@docker:~$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
2fd039880269153802f303435bf9a197fd1aefed5b96c5df0fe2a8e291266cb3
2a4eadafffcd4899dce3201f8e110489e77d5c0f6d4a9bac8af91f48a06adf35
e6d61413d2afa872ee797369775b0bbaf39d433b29c13c03538383f3bc228888

Total reclaimed space: 22.47MB
sysadmin@docker:~$
sysadmin@docker:~$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
sysadmin@docker:~$
```

Delete the stop containers using the prune command

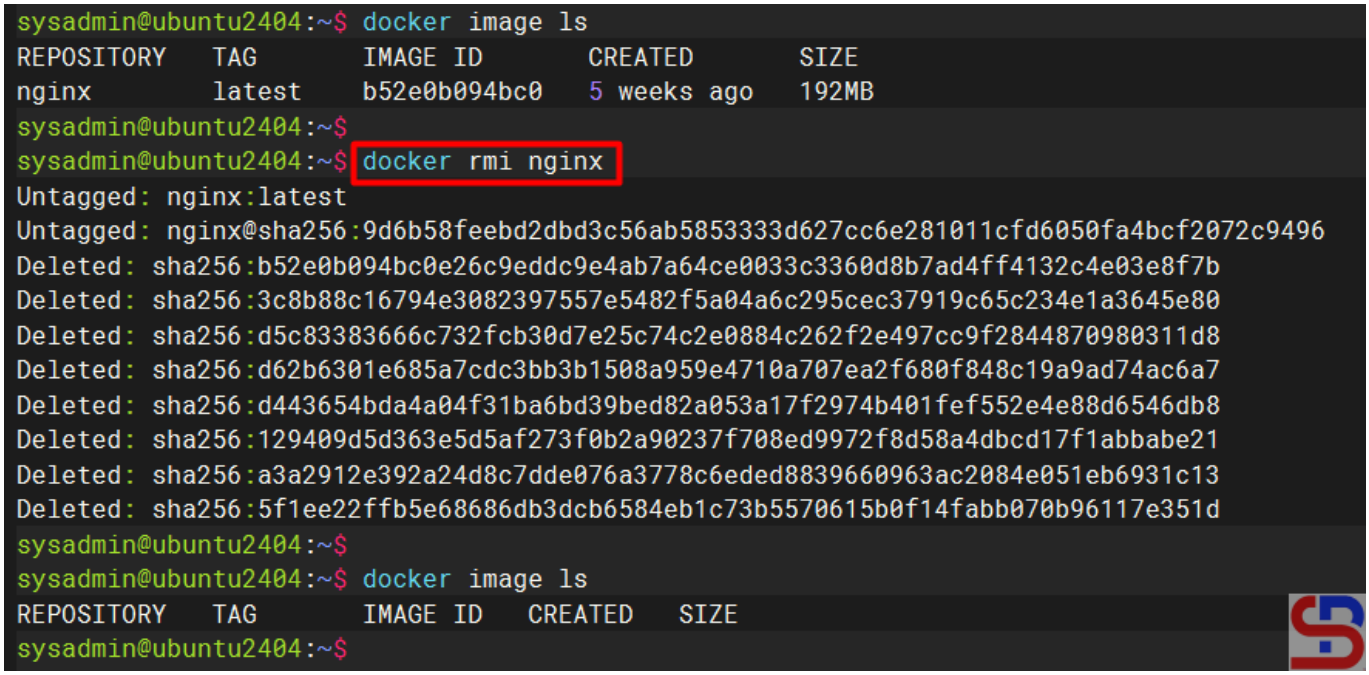
14. Delete the image(s)

To delete the Docker image that you have already downloaded, use the format below:

```
docker image rm image_name
```

Run the image below if you want to delete the nginx image:

```
docker image rm nginx
```



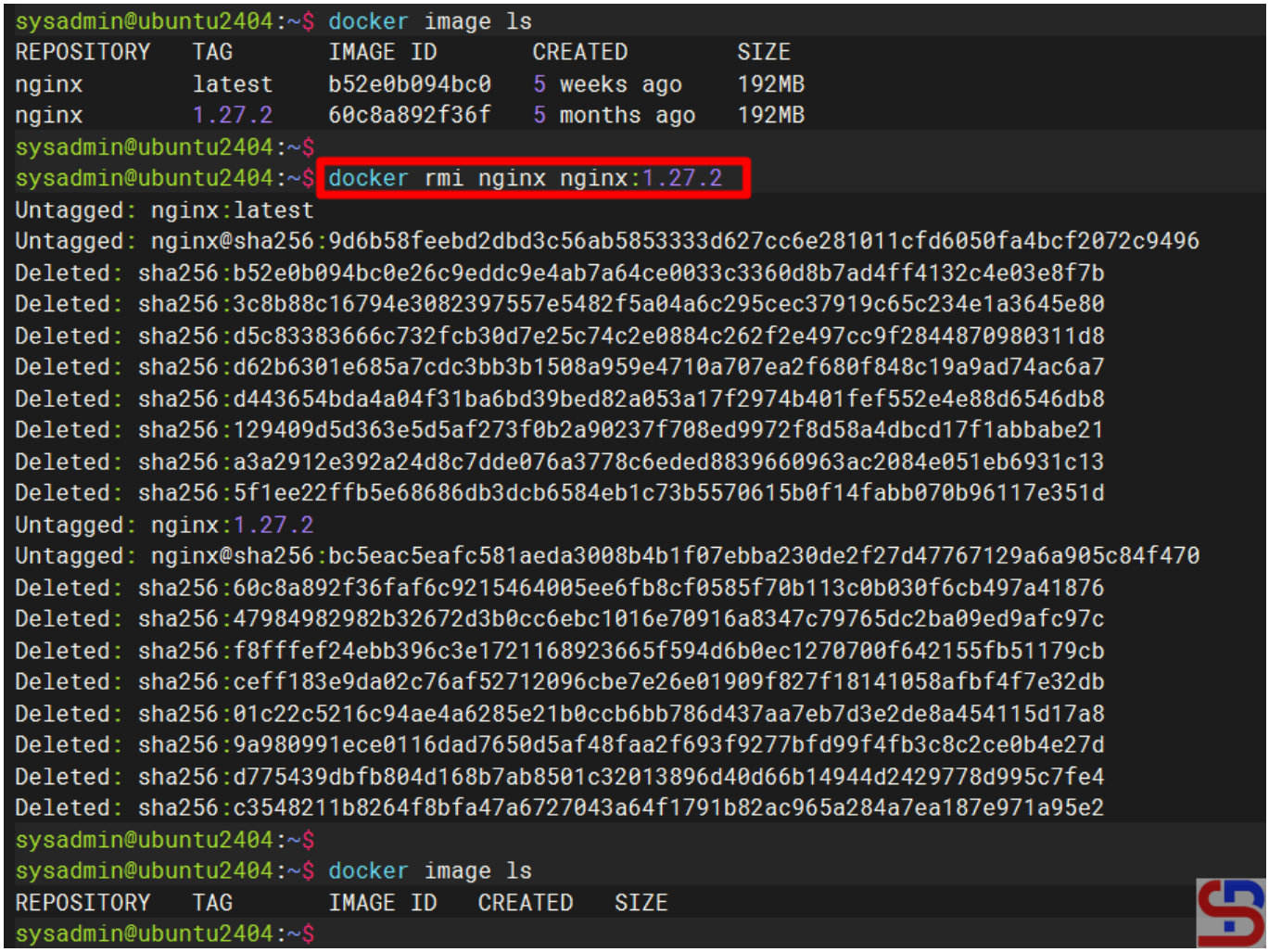
Delete the image

However, you must know that you **can't delete the Docker image if the image is still running in the container**. So you must remove the container first before you delete the image. If you want to delete multiple Docker images, use the following format:

```
docker image rm image_name1 image_name2 ...
```

So if you want to delete the nginx image and nginx:1.27.2 at once, then use the command below:

```
docker image rm nginx nginx:1.27.2
```



Delete more than one image

And if you want to delete all the images, you can use the command below:

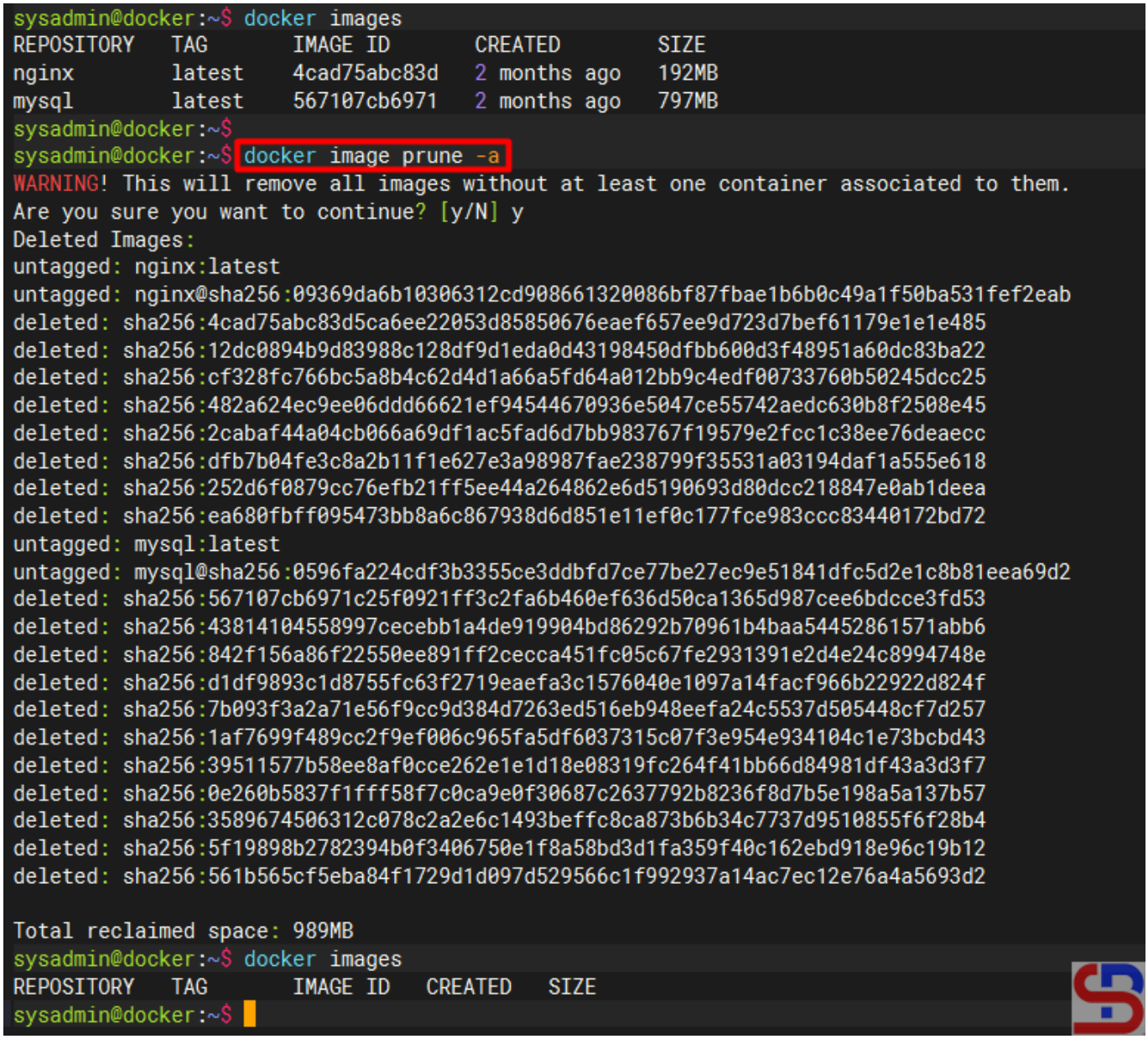
```
docker rmi $(docker images -a -q)
```

```
sysadmin@ubuntu2404:~$ docker image ls
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
nginx        latest    b52e0b094bc0   5 weeks ago    192MB
nginx        1.27.2    60c8a892f36f   5 months ago   192MB
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker rmi $(docker images -a -q)
Untagged: nginx:latest
Untagged: nginx@sha256:9d6b58feebd2dbd3c56ab5853333d627cc6e281011cfd6050fa4bcf2072c9496
Deleted: sha256:b52e0b094bc0e26c9eddc9e4ab7a64ce0033c3360d8b7ad4ff4132c4e03e8f7b
Deleted: sha256:3c8b88c16794e3082397557e5482f5a04a6c295cec37919c65c234e1a3645e80
Deleted: sha256:d5c83383666c732fcb30d7e25c74c2e0884c262f2e497cc9f2844870980311d8
Deleted: sha256:d62b6301e685a7cdc3bb3b1508a959e4710a707ea2f680f848c19a9ad74ac6a7
Deleted: sha256:d443654bda4a04f31ba6bd39bed82a053a17f2974b401fef552e4e88d6546db8
Deleted: sha256:129409d5d363e5d5af273f0b2a90237f708ed9972f8d58a4dbcd17f1abbabe21
Deleted: sha256:a3a2912e392a24d8c7dde076a3778c6eded8839660963ac2084e051eb6931c13
Deleted: sha256:5f1ee22ffb5e68686db3dcb6584eb1c73b5570615b0f14fabb070b96117e351d
Untagged: nginx:1.27.2
Untagged: nginx@sha256:bc5eac5eafc581aeda3008b4b1f07ebba230de2f27d47767129a6a905c84f470
Deleted: sha256:60c8a892f36faf6c9215464005ee6fb8cf0585f70b113c0b030f6cb497a41876
Deleted: sha256:47984982982b32672d3b0cc6ebc1016e70916a8347c79765dc2ba09ed9afc97c
Deleted: sha256:f8fffef24ebb396c3e1721168923665f594d6b0ec1270700f642155fb51179cb
Deleted: sha256:ceff183e9da02c76af52712096cbe7e26e01909f827f18141058afbf4f7e32db
Deleted: sha256:01c22c5216c94ae4a6285e21b0ccb6bb786d437aa7eb7d3e2de8a454115d17a8
Deleted: sha256:9a980991ece0116dad7650d5af48faa2f693f9277bfd99f4fb3c8c2ce0b4e27d
Deleted: sha256:d775439dbfb804d168b7ab8501c32013896d40d66b14944d2429778d995c7fe4
Deleted: sha256:c3548211b8264f8bfa47a6727043a64f1791b82ac965a284a7ea187e971a95e2
sysadmin@ubuntu2404:~$
sysadmin@ubuntu2404:~$ docker image ls
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE
sysadmin@ubuntu2404:~$
```

Delete all the images

Or, you can use the command below to remove the unused images:

```
docker image prune -a
```



Delete all the unused images using the prune command

Note

If you forget or don’t know what command to use in Docker, use the following command:

```
docker --help
```

```
sysadmin@ubuntu2404:~$ docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Common Commands:
  run         Create and run a new container from an image
  exec        Execute a command in a running container
  ps          List containers
  build       Build an image from a Dockerfile
  pull        Download an image from a registry
  push        Upload an image to a registry
  images      List images
  login       Authenticate to a registry
  logout      Log out from a registry
  search      Search Docker Hub for images
  version     Show the Docker version information
  info        Display system-wide information

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx
  compose*    Docker Compose
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
```

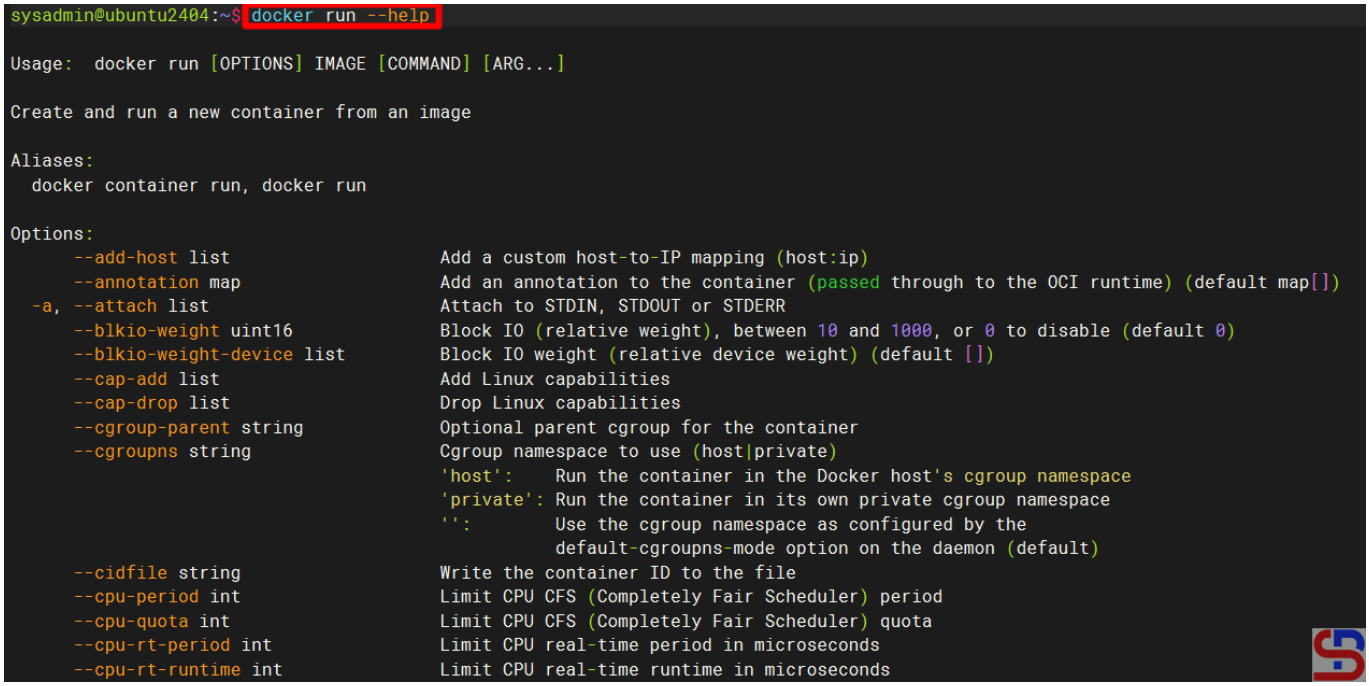
Using the docker help command

After that, if you want to know the options in the Docker command, then use the following format:

```
docker command --help
```

For example, if you want to know the options of the run command in Docker, then type the command below:

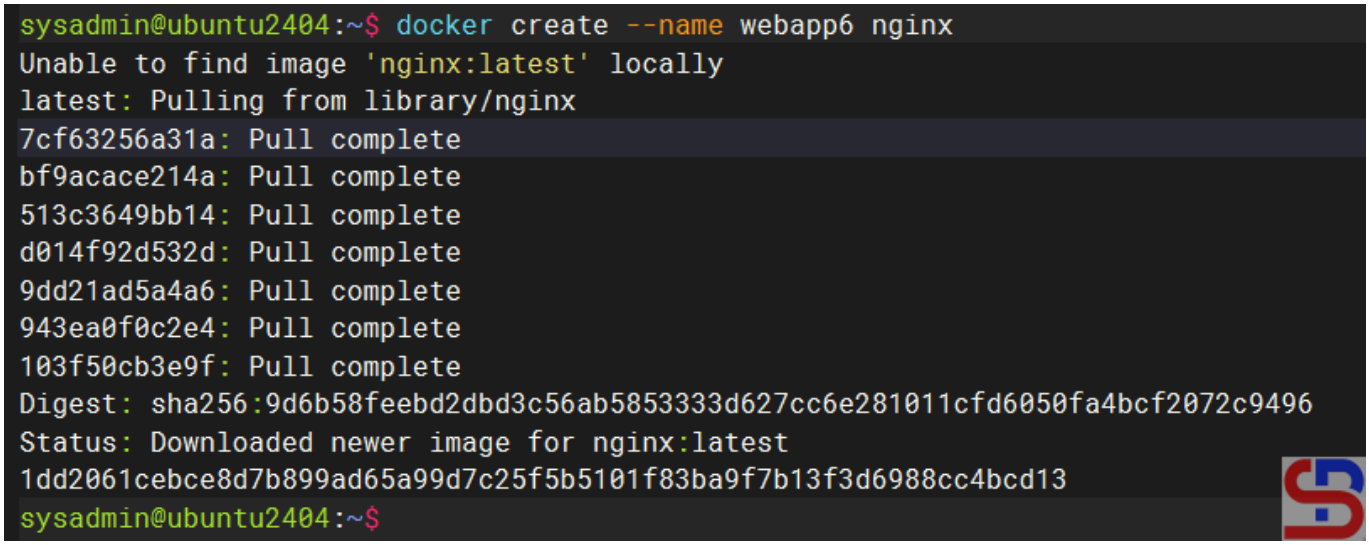
```
docker run --help
```



Using the docker run help command

You can shorten all the **Docker container** commands to just the **docker** command to shorten the typing time. For example, if you want to create a container, you can use the command:

```
docker create --name webapp6 nginx
```



Using the docker create command

References

geeksforgeeks.org
mygreatlearning.com
youtube.com
youtube.dimas-maryanto.com

How to Install Docker on the Linux Server?

written by sysadmin | 21 May 2025

A Docker is a platform for developing, shipping, and running container applications. Docker is like installing a virtual machine application on your laptop or server, whether it’s VirtualBox, VMWare, or Xen, so you can test various operating systems or applications on it without putting your laptop or server in danger.

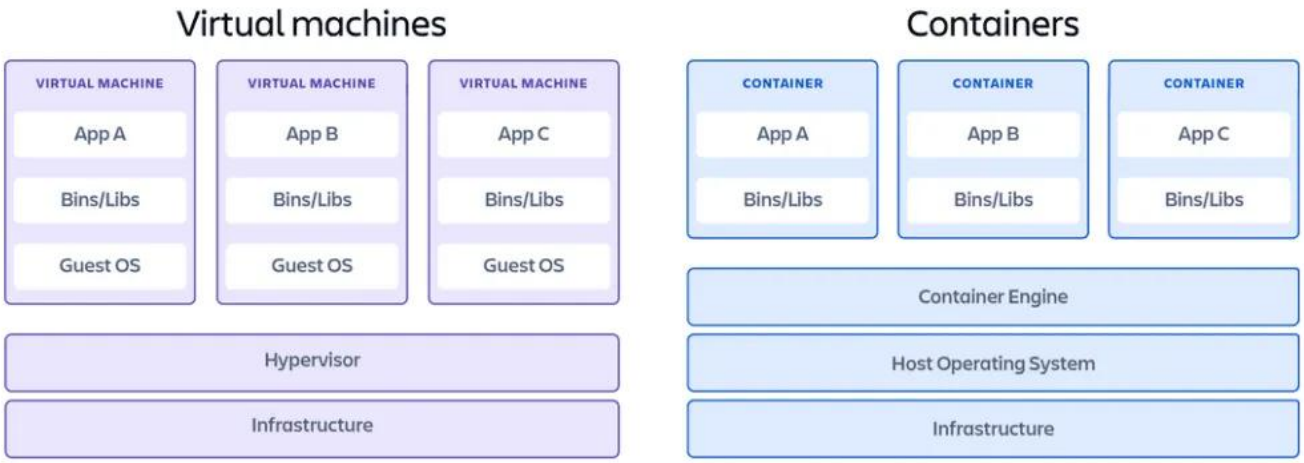
Problem

How to install Docker on the Linux server?

Solution

A. Docker summary

The key differentiator between containers and virtual machines is that virtual machines virtualize an entire machine down to the hardware layers, and containers only virtualize software layers above the operating system level. Take a look at the image below to make the difference between virtual machines and Docker clearer:



Comparison of Docker and Virtual Machine Architecture (image credit from atlassian.com)

The table below shows the comparison between virtual machines and Docker:

Comparison Item	Docker Container	VM
Isolation level	Low	High
Time required for startup	Seconds	Minutes
Image size	Several megabytes	Hundreds of megabytes to several gigabytes
Running performance (compared with bare metal servers)	Performance loss: < 2%	Performance loss: about 15%
Image portability	Not related to the platform	Related to the platform
Density	100 to 1000 on a single machine	10 to 100 on a single machine
Security	1. When the privilege of a user in a container is escalated from a common user to the **root** user, the user gains root permissions of the host machine. 2. Hardware isolation is not implemented, so containers are vulnerable to attacks.	1. The root permissions of a VM tenant are isolated from those of the host machine. 2. Hardware isolation is implemented to prevent VM escape and data exchange.

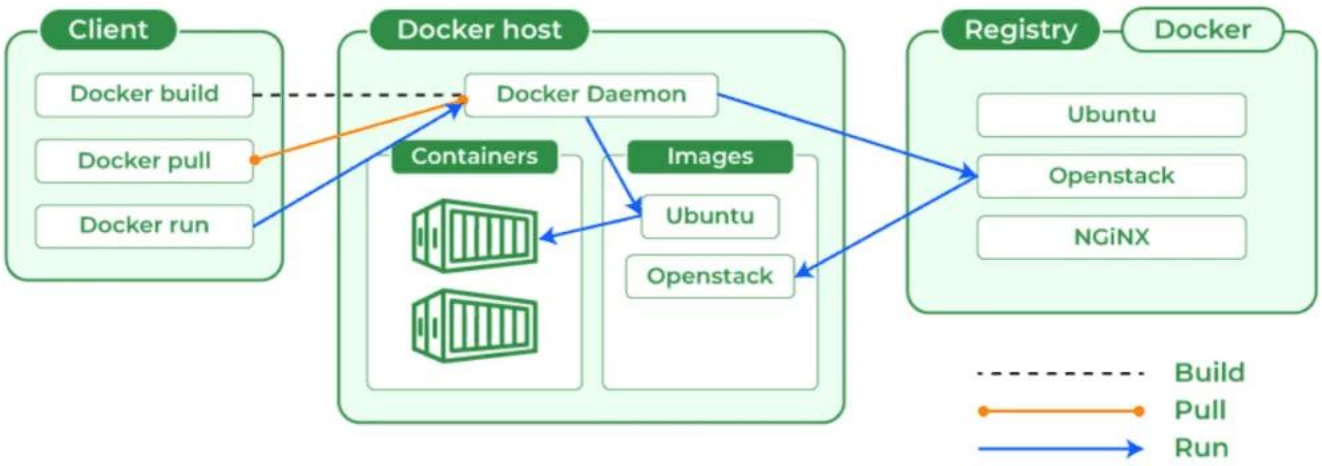
Comparison between Docker and virtual machine (Image credit from huawei.com)

Below is a brief explanation of the terms in Docker:

- □□□□□□ □□□□: Downloads images from Docker Hub if not locally available
- □□□□□□ □□□□□: Creates a local image using a Dockerfile, enabling custom images

- □□□□□□ □□□□:Uploads images to Docker Hub, allowing sharing
- □□□□□□ □□□: Takes an image to run a container, useful for starting web servers or other applications
- □□□□□□ □□□□□: Read-only templates forming the base of containers, including all application dependencies
- □□□□□□ □□□□□□: Interacts with Docker, sending instructions to the Docker daemon to execute tasks
- □□□□□□ **Daemon**: Handles all requests, including building, running, and distributing containers
- □□□□□□ □□□□□□□□□□: Include everything needed to run an application, such as code, libraries, and configurations
- □□□□□□ □□□□□□□□: Stores Docker images, with Docker Hub as a public registry and the option to create private ones

The image below is a picture of how the Docker works:



How Docker works (Image credit from geeksforgeeks.org)

B. Install Docker

In general, use the command below to install Docker on Linux:

```
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

But after you execute the commands above, there is an error like this when you install it in RockyLinux:

ERROR: Unsupported distribution ‘rocky’

You have to install Docker manually using these commands:

```
sudo dnf config-manager --add-repo
https://download.docker.com/linux/rhel/docker-ce.repo
sudo dnf -y install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

Or when you install Docker in OpenSUSE, you get an error like this:

ERROR: Unsupported distribution ‘opensuse-leap’

Use the commands below to install Docker in OpenSUSE:

```
sudo zypper install -y docker docker-compose docker-compose-switch
```

C. After installing Docker

Use the following command to run Docker:

```
sudo systemctl restart docker
sudo systemctl enable docker
```

To see the version of Docker you installed, use the command below:

```
docker info
```

```
sysadmin@ubuntu2404:~$ docker info
Client: Docker Engine - Community
 Version:    27.5.1
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc.)
    Version:  v0.20.0
    Path:     /usr/libexec/docker/cli-plugins/docker-buildx
  compose: Docker Compose (Docker Inc.)
    Version:  v2.32.4
    Path:     /usr/libexec/docker/cli-plugins/docker-compose

Server:
ERROR: permission denied while trying to connect to the Docker daemon socket at u
nix:///var/run/docker.sock: Get "http://%2Fvar%2Frun%2Fdocker.sock/v1.47/info": d
ial unix /var/run/docker.sock: connect: permission denied
errors pretty printing info
sysadmin@ubuntu2404:~$
```

Running the docker info command

D. Test the application in Docker

After that, to see whether Docker is running well on the server, use the command below to run the hello-world container on your server:

```
docker run hello-world
```

If you have got the error like the image below:

```
sysadmin@ubuntu2404:~$ docker run hello-world
docker: permission denied while trying to connect to the Docker daemon socket at
unix:///var/run/docker.sock: Head "http://%2Fvar%2Frun%2Fdocker.sock/_ping": dial
 unix /var/run/docker.sock: connect: permission denied.
See 'docker run --help'.
sysadmin@ubuntu2404:~$
```

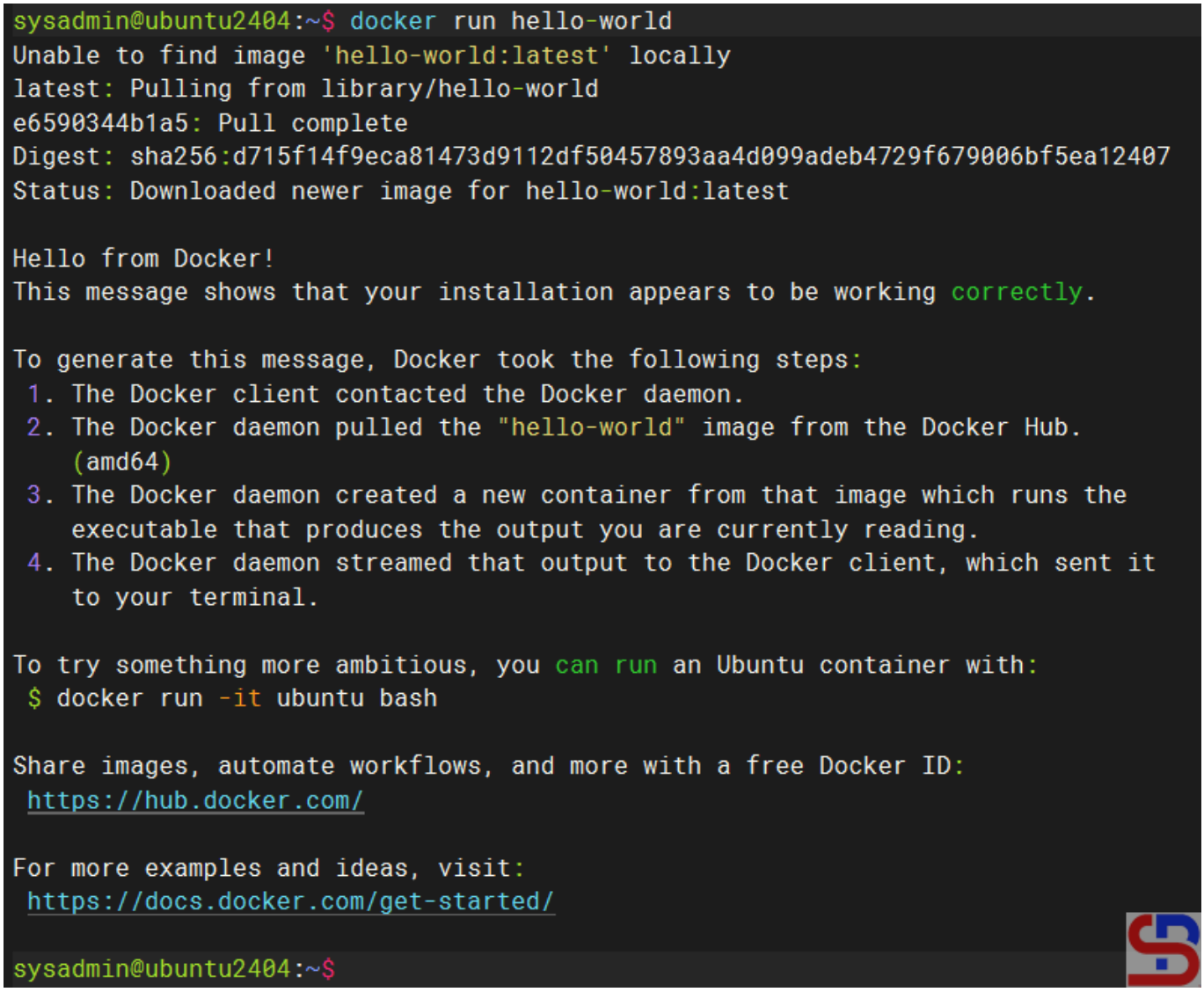
Error when running the docker run command

```
ERROR: permission denied while trying to connect to the Docker daemon socket
at unix:///var/run/docker.sock: Get
“http://%2Fvar%2Frun%2Fdocker.sock/v1.47/info”: dial unix
/var/run/docker.sock: connect: permission denied
```

Then you have to run the following command:

```
sudo usermod -aG docker $USER
```

Log out of your server and log in again. After that, you should be able to run the Docker commands like in the image below:



Test the Docker run command

That way, your Docker application is ready to use.

Note

You don’t have to use Docker to create and run containers, but you can use other applications such as podman, buildah, cri-o, etc. However, Docker is the most popular at the

moment. Also, the terms and workings of various container applications are almost the same, so if you understand the terms and workings of Docker, then you will also understand the terms and workings of other container applications. To learn about basic commands in Docker, go to this page.

References

phoenixnap.com
youtube.com
docs.docker.com
geeksforgeeks.org
atlassian.com
qa.com
info.support.huawei.com
simform.com
linkedin.com
docs.rockylinux.org
en.opensuse.org