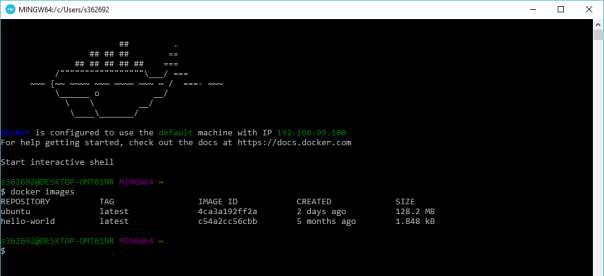
**Docker – container and hosts**

The good thing about the Docker engine is that it is designed to work on various operating systems. We have already seen the installation on Windows and seen all the Docker commands on Linux systems. Now let’s see the various Docker commands on the Windows OS.

**Docker Images**

Let’s run the Docker **images** command on the Windows host.



From here, we can see that we have two images: **ubuntu** and **hello-world**.

**Running a Container**

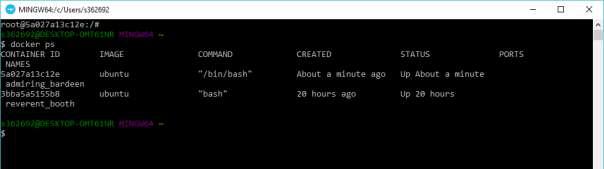
Now let’s run a container in the Windows Docker host.

We can see that by running the container, we can now run the Ubuntu container on a Windows host.

**Listing All Containers**



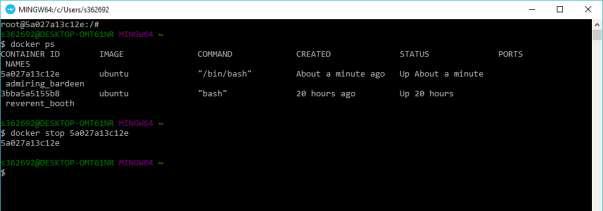
Let’s list all the containers on the Windows host.



**Stopping a Container**



Let’s now stop a running container on the Windows host.



So you can see that the Docker engine is pretty consistent when it comes to different Docker hosts and it works on Windows in the same way it works on Linux.

**Docker – Configuring Docker**

In this chapter, we will look at the different options to configure Docker.

**service docker stop**

This command is used to stop the Docker **daemon** process.

**Syntax**

service docker stop

**Options**

None

**Return Value**

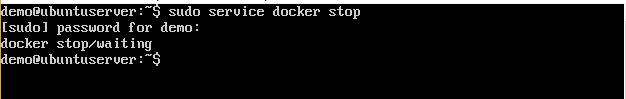
A message showing that the Docker process has stopped.

**Example**

sudo service docker stop

**Output**

When we run the above command, it will produce the following result:



**service docker start**

This command is used to start the Docker daemon process.

**Syntax**

service docker start

**Options**

None

**Return Value**

A message showing that the Docker process has started.

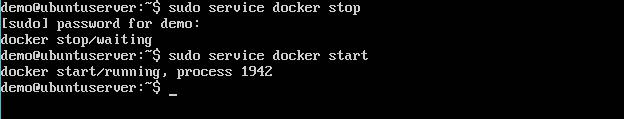
**Example**



sudo service docker start

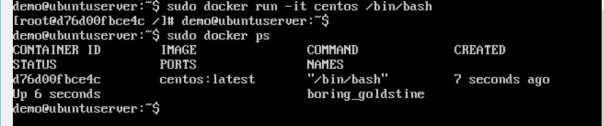
**Output**

When we run the above command, it will produce the following result:



**Docker – Containers and Shells**

By default, when you launch a container, you will also use a **shell command** while launching the container as shown below. This is what we have seen in the earlier chapters when we were working with containers.



In the above screenshot, you can observe that we have issued the following command:

sudo docker run –it centos /bin/bash

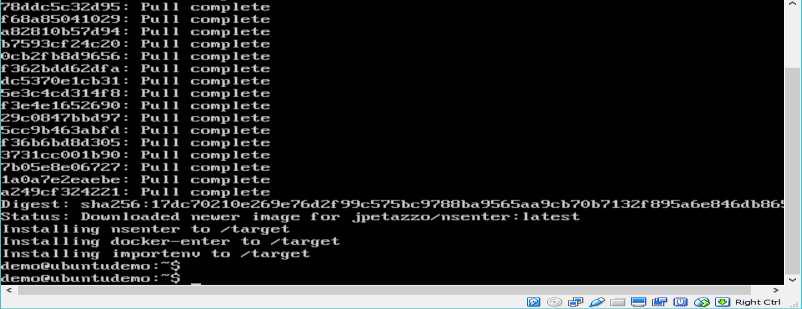
We used this command to create a new container and then used the Ctrl+P+Q command to exit out of the container. It ensures that the container still exists even after we exit from the container.

We can verify that the container still exists with the Docker **ps** command. If we had to exit out of the container directly, then the container itself would be destroyed.

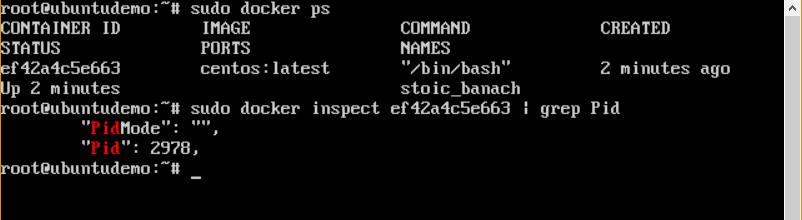
Now there is an easier way to attach to containers and exit them cleanly without the need of destroying them. One way of achieving this is by using the **nsenter** command.

Before we run the **nsenter** command, you need to first install the **nsenter** image. It can be done by using the following command:

docker run --rm -v /usr/local/bin:/target jpetazzo/nsenter



Before we use the **nsenter** command, we need to get the Process ID of the container, because this is required by the **nsenter** command. We can get the Process ID via the Docker **inspect command** and filtering it via the **Pid**



As seen in the above screenshot, we have first used the **docker ps** command to see the running containers. We can see that there is one running container with the ID of ef42a4c5e663.

We then use the Docker **inspect** command to inspect the configuration of this container and then use the **grep** command to just filter the Process ID. And from the output, we can see that the Process ID is 2978.

Now that we have the process ID, we can proceed forward and use the **nsenter** command to attach to the Docker container.

**nsenter**

This method allows one to attach to a container without exiting the container.

**Syntax**

nsenter –m –u –n –p –i –t containerID command

**Options**

* **-u** is used to mention the **Uts namespace**
* **-m** is used to mention the **mount namespace**
* **-n** is used to mention the **network namespace**
* **-p** is used to mention the **process namespace**
* **-i** is to make the container run in interactive mode.
* **-t** is used to connect the I/O streams of the container to the host OS.
* **containerID** –This is the ID of the container.
* **Command** –This is the command to run within the container.

**Return Value**

None

**Example**

sudo nsenter –m –u –n –p –i –t 2978 /bin/bash

**Output**



From the output, we can observe the following points:

* The prompt changes to the **bash shell** directly when we issue the **nsenter** command.
* We then issue the **exit** command. Now normally if you did not use the **nsenter** command, the container would be destroyed. But you would notice that when we run the **nsenter** command, the container is still up and running.