LAB 1 : Launching and operating windows containers

Author: Gourav Shah
Publisher: School of Devops
Version: v1.0.1

The objective of this module is to get started with the basic operations for windows containers. In this module you are going to learn how to

- Get started with docker command line client utility.
- How to launch containers and work with those
- What are the common container launch options
- How to define port publishing options to access a web application in a container
- How to troubleshoot running containers
- How to stop and remove containers

Finding information

Open a CMD/Windows PowerShell terminal and run

```
docker

docker version

docker system info
```

Launching and operating your first Container

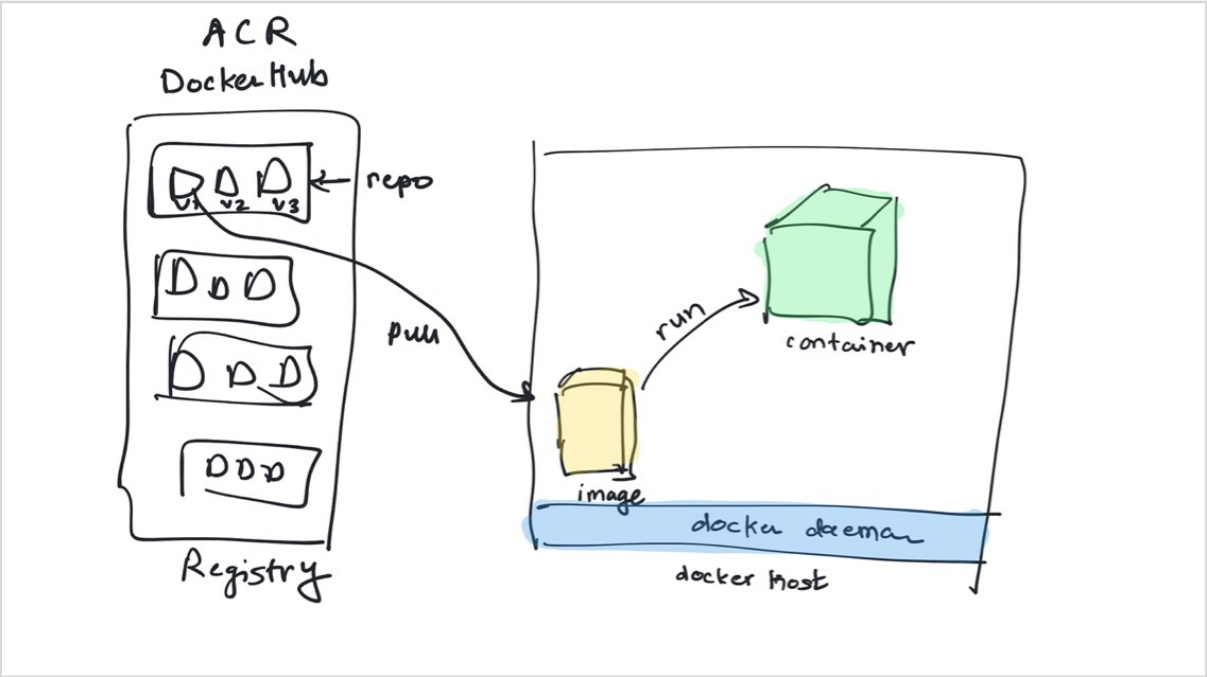
Here is a command sequence to launch your first container. To understand what's happening on the Docker Server side, you could also open another terminal and run the following command.

```
docker system events
```

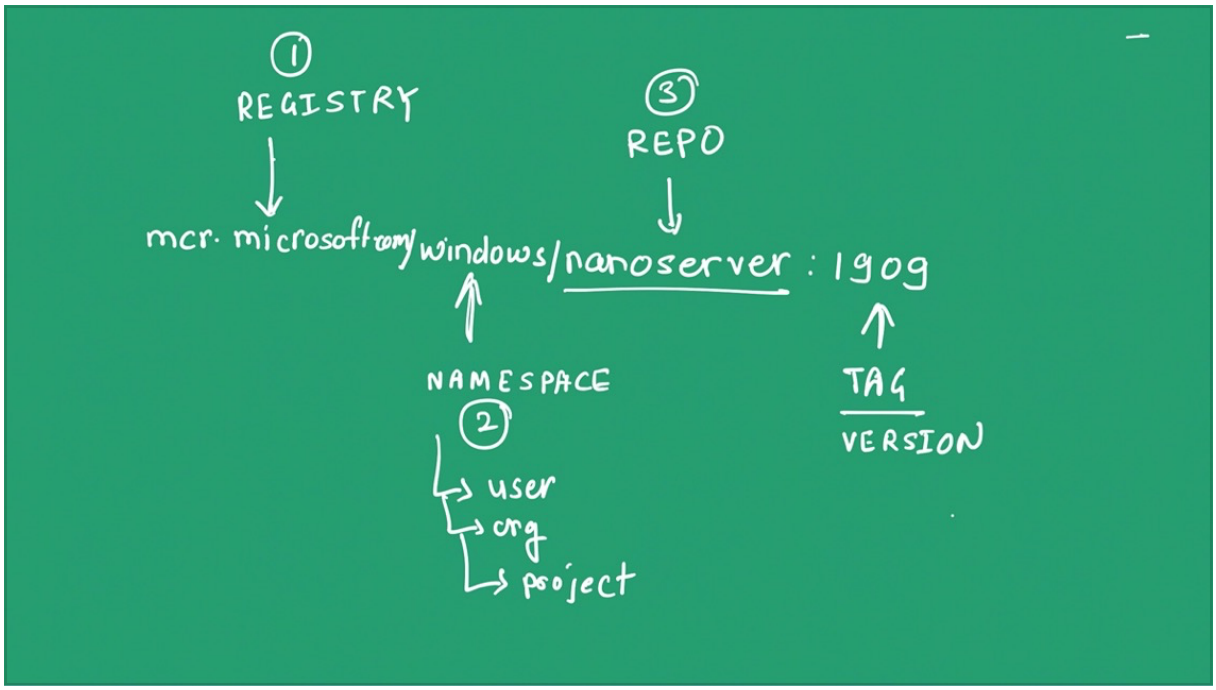
When you launch the above command, you would notice no output, and it would appear as if its hung. However, its just streaming events from docker server and should start showing those events as you start doing some operations such as pulling an image and launching a container next.

Pulling a docker image from registry

So you want to run a container. Well, in order to do that, just like the case with VMs you need an image available on the host which would run this container. Where does the image come from ? Well, its the container registry (e.g. Docker Hub/ Microsoft Container Registry etc.). Think of a container registry as GitHub for your docker images.



Each image could have up to four fields as the example below,



Lets pull the official Nano Server image created by Microsoft. This is the link Docker Hub to nano server image repo on docker hub. An example image is as follows,

```
mcr.microsoft.com/windows/nanoserver:1809
```

Where, following diagram explains each of these fieds.

To check the existence of this image locally and then to pull it if not present, and then to validate run the following commands.

```
docker image ls

docker image pull  mcr.microsoft.com/windows/nanoserver:1809

docker image ls
```

Launch your first container

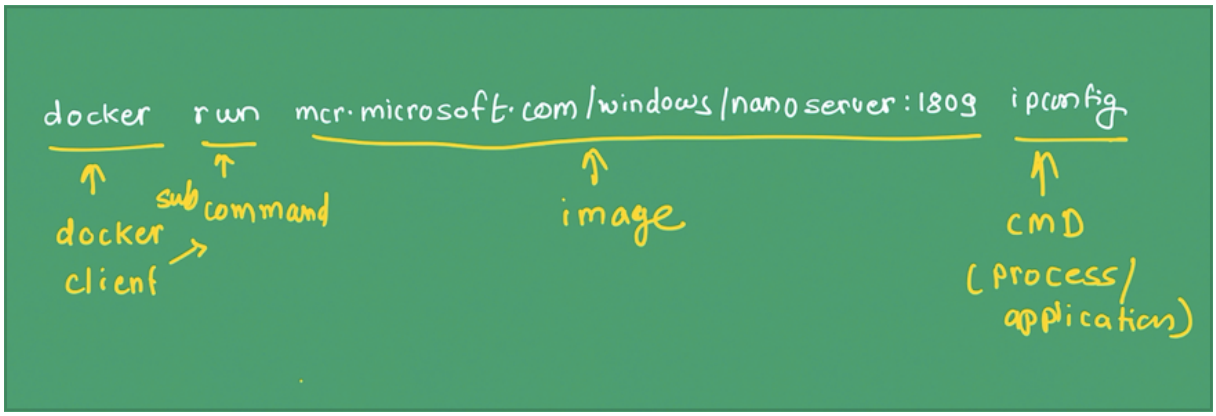
Now, after pulling this image, launch container as

```
docker container run mcr.microsoft.com/windows/nanoserver:1809 ipconfig
```

or the legacy way,

```
docker run mcr.microsoft.com/windows/nanoserver:1809 ipconfig
```

Whats the difference between `docker container run` and `docker run`, well, both are essentially the same commands. Some time after 1.13 version of docker, they introduced new categories of commands, and docker run became docker container run. Its still backwards compatible. In fact, folks are so used to the previous versions of the commands, that they continue to use only those many a times. So essentialy both the commands are same, and feel free to use whoever you prefer. I am going to go with the second command here.




```
docker run mcr.microsoft.com/windows/nanoserver:1809 ipconfig
```

When you launch the container, observer two things,

- The output on the terminal
- Events streamed from the other terminal where you had launched `docker system events`

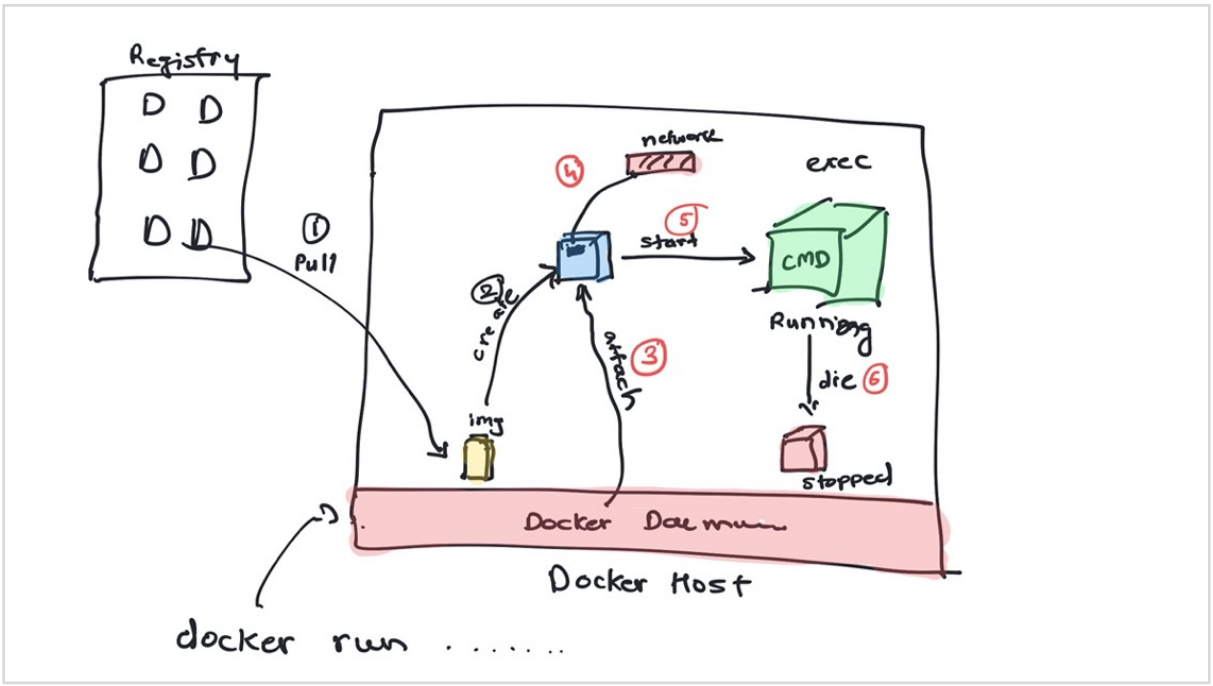
Following is the sample output of the command run above.

```
 docker container run mcr.microsoft.com/windows/nanoserver:1809
Microsoft Windows [Version 10.0.17763.1098]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\>
```

And following are the list of events on the server side

- container create
- container attach
- network connect
- container start
- container exec_create
- container exec_start: cmd.exe
- container die
- network disconnect



Run the following command to check if the container you started is running

```
docker ps
```

Above command is to list the currently running container. If you do not find the container you started in this list, check the status of it by listing the last run container using `-l` flag.

```
docker ps -l
```

[sample output]

```
 docker ps -l
CONTAINER ID        IMAGE                                       COMMAND
CREATED             STATUS                      PORTS               NAMES
9568fdc29d49        mcr.microsoft.com/windows/nanoserver:1809   "c:\\windows\
\system32…"   13 minutes ago      Exited (0) 12 minutes ago
charming_antonelli
```

it shows as exited.

What just happened ?

Lets now analyse what happened here after you launched the container

- Docker tries to look up for the image locally. If it does not find it, it will try to download it from the registry. In our case, the image was present, so this step was skipped.
- Docker then created the container i.e. all the components required to run a process in a contained environment.
- Docker service attached itself to this container so that it can monitor the process, manage logs etc.
- Docker then asked the kernel to setup network configurations for the container. This step creates the virtual network interface for the container, sets up the IP address etc.

- The docker started running the container .
- Docker then launches the actual command or the application that you desire .e.g. ipconfig in this example. The application so start can also be configured implicitly in the image metadata, so that you do not have to provide at the launch time.
- Since this example had a one off command e.g. ipconfig instead of a long running process e.g. IIS, as soon as the process exits, docker thinks its job is done and it stops the container.
- Once the container stops, the network is also disconnected from the container. Do remember, network is a separate entity than the container itself.

Listing Containers

Lets now launch a few more containers with different commands

```
docker run mcr.microsoft.com/windows/nanoserver:1809 hostname

docker run mcr.microsoft.com/windows/nanoserver:1809 ping

docker run mcr.microsoft.com/windows/nanoserver:1809 net accounts
```

Now try to list containers using the following flags to docker ps command

- -l : only one last container
- -n 3 : last two containers. You could try different integers instead of 3
- -a : all containers

eg.

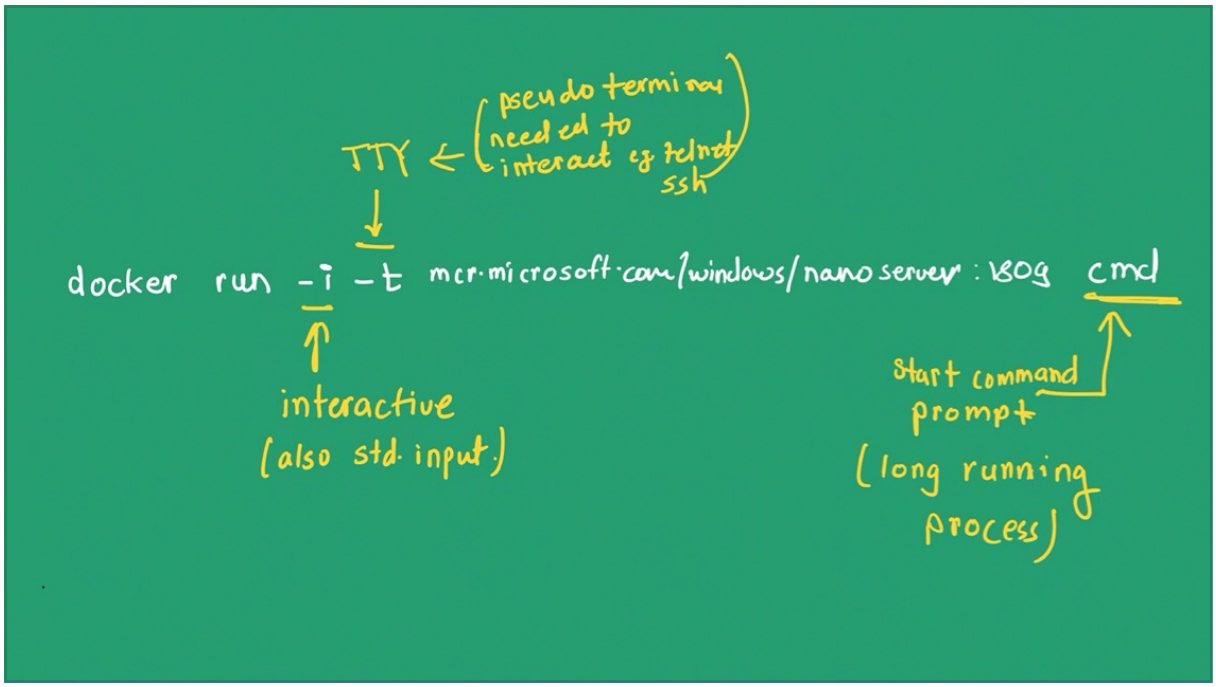
```
docker ps
docker ps –l
docker ps –n 3
docker ps –a
```

Using Common Container Runtime Options

In the previous section you launched and run a few containers, however without any flags to `docker run` command, and with one off commands. In this one, you would learn some common options to use with docker run commands, run containers for long, and observe how it works.

Launching an Interactive Container

The first two options you would use would be -i and -t. Also this time you would launch a long running command e..g cmd




```
docker run -it mcr.microsoft.com/windows/nanoserver:1809 cmd
```

When you launch the above command, you would notice the following,

- This time not only the container is launched, but also you land inside the container with a command prompt. Finally, you could look and feel the container now ! Yes !
- The container is not stopped, but rather in a running state. This is because you launched it with `cmd` this time. Until you keep running that command prompt and do not exit, it will keep running.

Exercise : While you are inside the container, you may want to run some windows commands and observe how containers work. Type `exit` and enter to come out of the container. Exiting from the command prompt will stop the container.

Now, try launching a container with **Windows Server Core** image. While you launch it as below, observe the docker system events as well to see whats different this time (e.g. you should see an event for image being pulled if you have not used this image before).

```
docker container run --rm -it mcr.microsoft.com/windows/servercore:ltsc2019
powershell
```

where,

- mcr.microsoft.com/windows/servercore:ltsc2019 : is the image for windows server core, LTSC (Long Term Servicing Channel) version.
- poweshell : launch powershell instead of cmd as before, as windows server core comes with psh.
- --rm : this option tells docker to delete the container once stopped (on exit from powershell). This is a useful option to clean up one off containers created just for testing.

You could try running some commands such as below inside this container

```
ipconfig
hostname
ls c:\
Get-Process
Get-WindowsFeature
```

Again, Exiting the powershell should stop and remove this container. Validate by running docker ps -a command.

Running a container in the background/detached mode

The most desired way of running a container would be to launch an application with a container, and keep running it in the background. This could be achieved using `--detach` option as follows . This time I would be launching a sample application

```
docker run -idt --name sample --rm mcr.microsoft.com/dotnet/core/
samples:aspnetapp
```

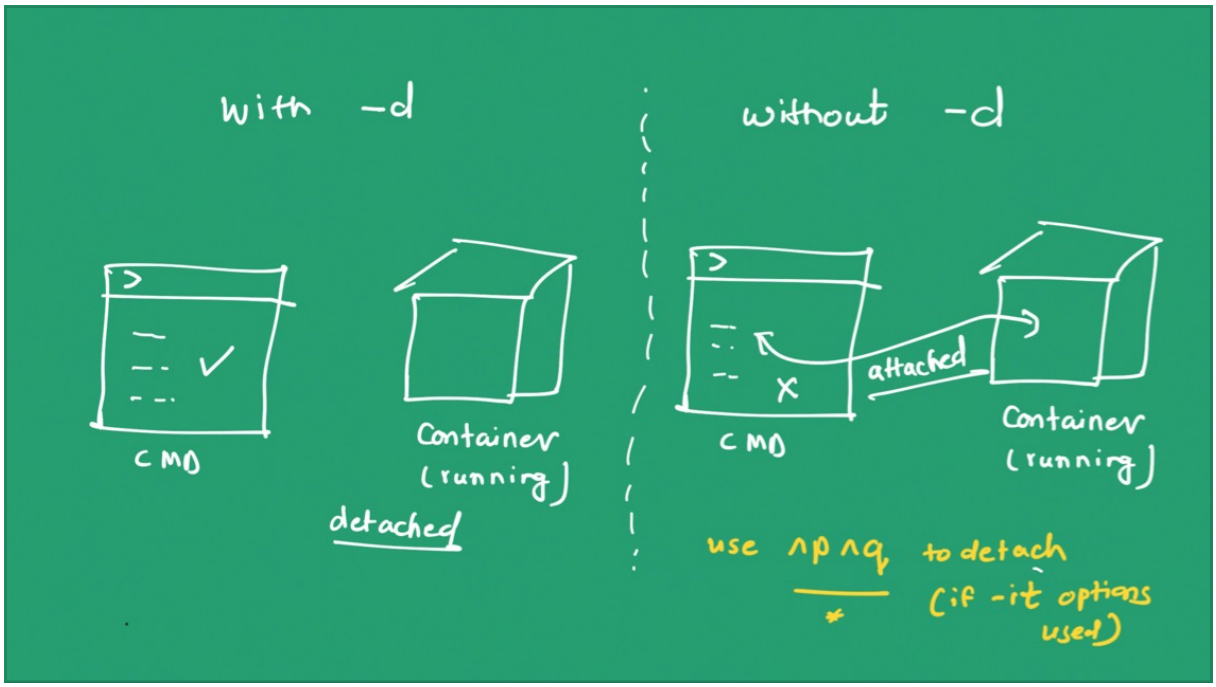
Where,

- –name : this option provides name to the container instead of the automatically generated random name that docker has assigned to it
- mcr.microsoft.com/dotnet/core/samples:aspnetapp : an image with ASP .NET sample application built in

This time you should notice that the container is not only launched but also is running in the background, allowing you to proceed with the next commands/tasks. If you want to see the difference, try to run another container without -d option.

```
docker run -it --rm mcr.microsoft.com/dotnet/core/samples:aspnetapp
```

A container with the same image is launched without -d now. You should see that you are attached to the process running inside the container now. If you were using -d instead, this container would be running in the background with detached option.



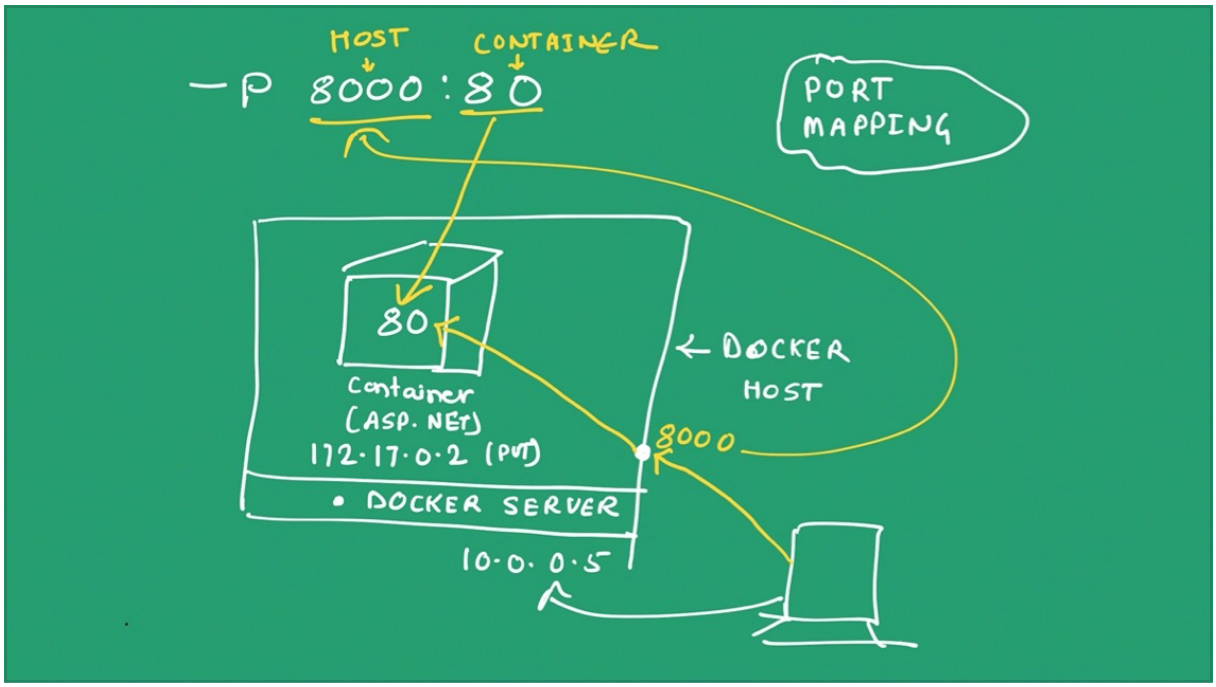
If launched without -d , remember to use `^p ^q` as escape sequence to detach from the container's process. This will only work if `-it` options were used to launch the container. Use `^c` in other cases which may kill the container.

Best Practice : When launching containers use -idt options. Out of this -d is a must. -it is optional but recommended as you could detach/attach from the process/command/ application running inside the container.

Accessing Web Apps with Port Mapping

You have learnt all useful common options to launch containers with. You in fact launched a sample ASP .Net application, which is a web application. Even the container is running, the question is how to access the web app ? Well, in order to do that, you need learn about one more option i.e. publishing ports / port mapping.

```
docker run -idt -p 8000:80  --name webapp --rm mcr.microsoft.com/dotnet/core/
samples:aspnetapp
```



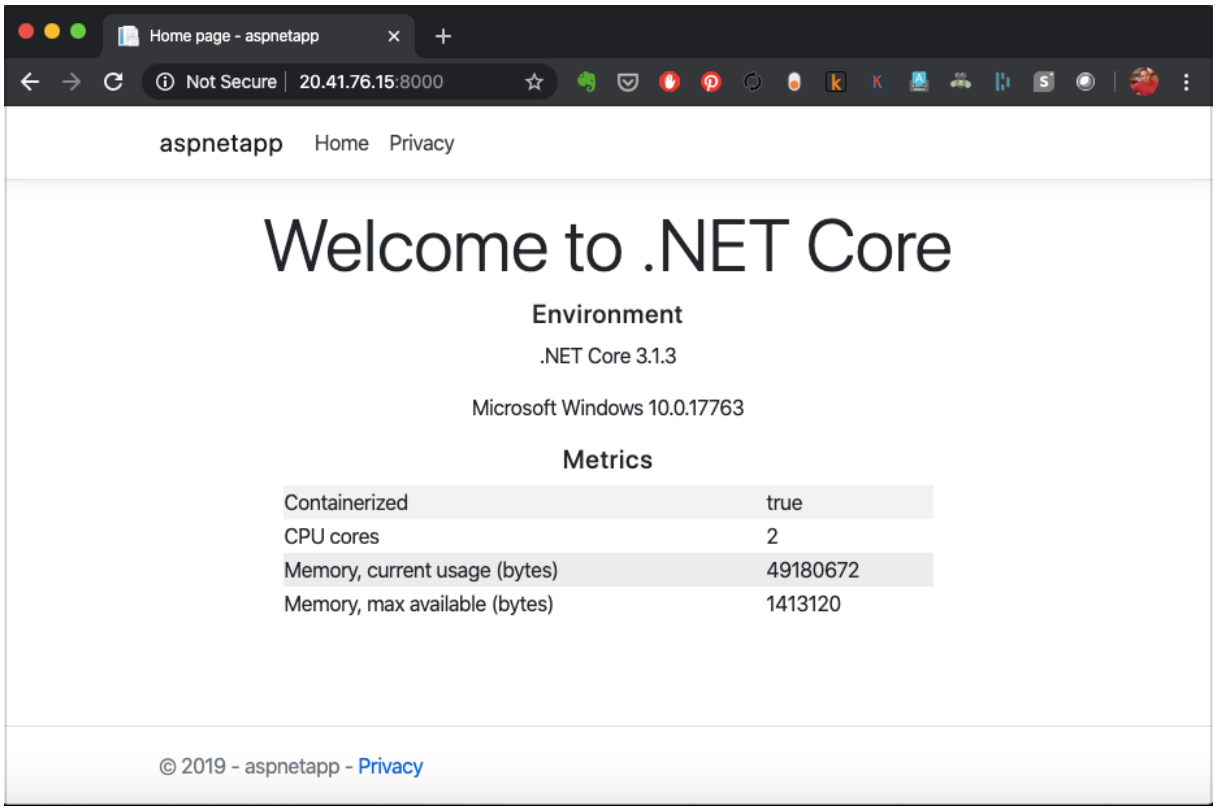
where,

- -p 8000:80 - This option maps Docker Host's 8000 port to container's 80 port, where the actual web application is running.

This allows you to access the application using host IP and host port as `http://hostip:hostport`

- If you are using Docker Desktop replace `hostip` with localhost and use this URL `http://localhost:8000`
- If you are using a remote server, use the IP Address or hostname of it to access the application e.g. for my azure VM its 'http://20.41.76.15:8000'
- Ensure that port 8000 is open from the firewall configurations if using a remote host/ cloud server.

e.g.



Apart from using -p option, you could also use the following

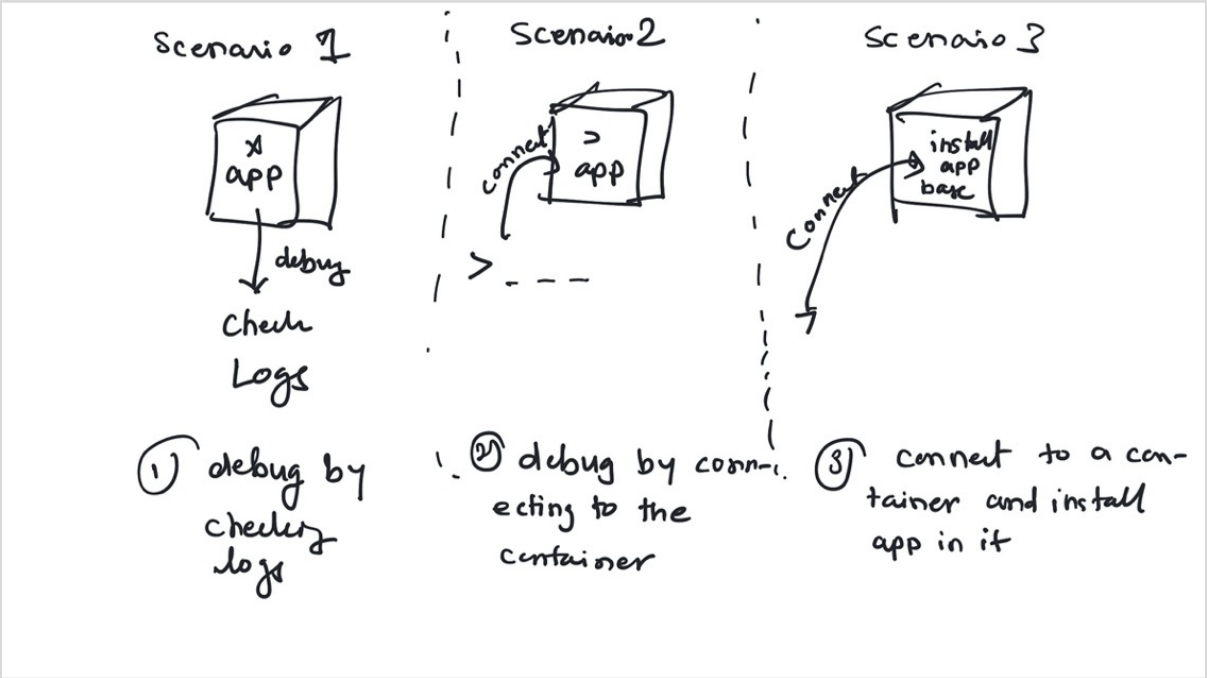
- Using only `-P` (P in capitals) option would automatically pick up the exposed port defined in container image and map it to a host port starting with 32768 it will automatically increment this number for future port mappings.
- Using options such as `-p 80` will pick up the container port 80 and map it to host port starting with 32768 it will automatically increment this number for future port

Troubleshooting Windows Containers

You have launched applications in a container. Now what ? Most of the times, you may have to do some debugging or troubleshooting on a running contianer. How do you achieve that ? Well there may be two operations involved while debugging/modifying container

- You may want to get the logs from the application
- You may want to connect to the container (e.g. open a shell inside it) to manage and debug further.

Example Scenarios



- **Scenario 1**: You have launched a ASP .NET application, which is misbehaving. You want to debug it and the first thing that you may want to do is get the applicaiton logs.
- **Scenario 2**: You have an application running inside a container. Logs don't tell you much about what its doing. You may want to connect to this container, open a shell and debug further.
- **Scenario 3**: You have a container running Windows Server Core. You want to modify it by installing IIS service on it.

Scenario 1 : Checking Application Logs

Lets look at scenario 1. If you want to check application logs you could use either of the following commands,

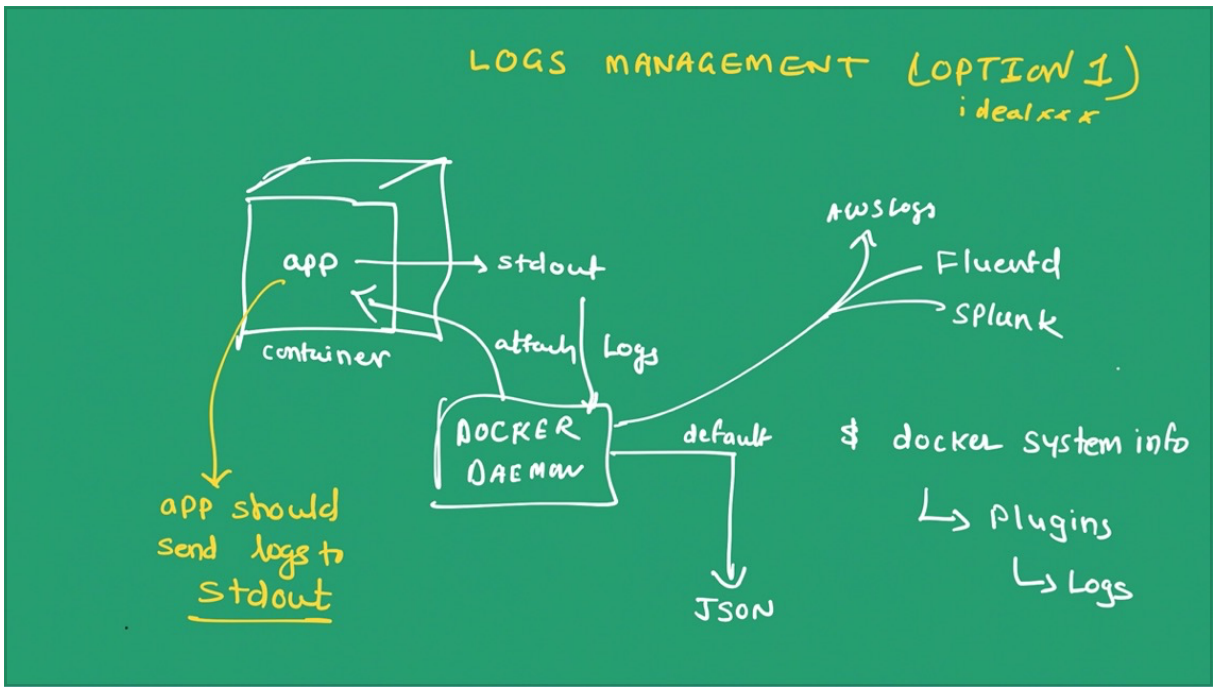
assuming your container name is webapp

```
docker ps
[note container's name/id ]
docker logs webapp

docker logs -f webapp
```

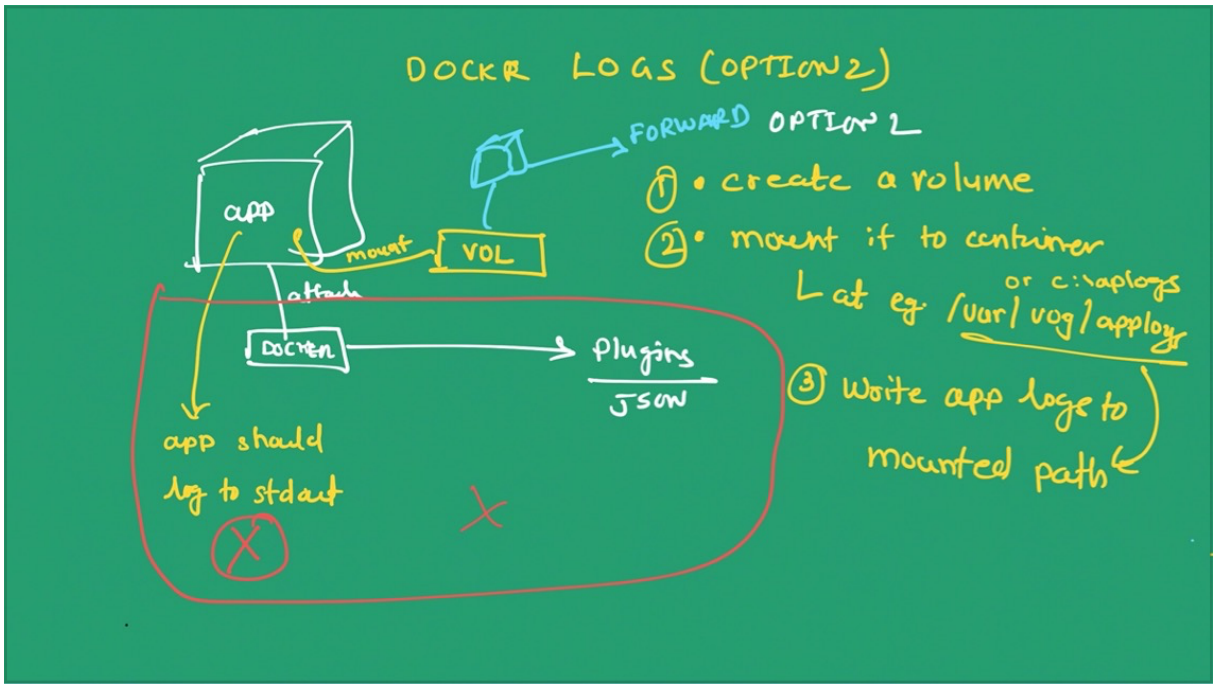
```
[this command will follow the logs and will show you as they update. Use ^c to stop following. ]
```

The above command does show the logs for the ASP .NET app running inside the container . Well, the question now is how does docker know about these application logs ? Answer to that is depicted in the following diagram.



- Docker attaches to the application's standard out.
- Application writes logs to standard out e.g. sample ASP .NET app above does this.
- Docker daemon then forwards the logs based on the logs plugin used. Default is a json-file. However docker is capable of forwarding logs to remote systems using plugins such as awslogs, gcplogs, syslog, plunk.
- This integration available out of box. You could check the log plugins available using `docker system info` command's plugins-> Logs section.
- This also needs application to log to standard out.
- You could read more about docker's logging drivers here Configure logging drivers | Docker Documentation.

What if your application can not be modified to log to stdio (standard output) , or what if it generates multiple log streams and writes it to multiple files. In such cases, you could use the following option.



Where,

- You mount a volume to the container. This needs to be done while launching the container. Volume can be local or remote.
- Mount it at the path where application write the logs e.g. /var/log/applog or e.g. C:\applog
- Ensure that your application then writes the logs only to that path.
- Optionally, from the volume, you could forward the logs to remote systems using forwarders.

Scenario 2 : Get inside the Container's shell

Just checking the logs is not enough. You may want to get inside a command prompt or a powershell of the container. Alternately, you may want to just execute a one off command against the running container. Well you could do either using `exec` command that docker offers.

```
docker ps
[note container name/id]

[to run a  one off  command]
docker exec webapp ipconfig
```

```
[for getting a shell access inside the container]
docker exec -it webapp cmd
```

The most commonly used command is the last one with `-it` options and `cmd` as the command. You could have used `powershell` instead of cmd had your container image supported powershell. The one with windows server core comes with one.

Once you open a command prompt above, you could execute any command that is available as part of that containers' image.

Scenario 3 : Installing IIS inside a Container

This is just an extension of scenario 2. Launch a container with Windows Sever Core this time,

```
docker container run -idt -p 80:80 --name myweb --rm -it mcr.microsoft.com/
windows/servercore:ltsc2019 powershell
```

Now, login to this container using exec and powershell and follow the steps to enable and start IIS web service.

```
[get inside the running container with a powershell]
docker exec -it myweb powershell

[check if IIS server is enabled]
Get-WindowsFeature

[enable IIS service if not alreday]
Add-WindowsFeature Web-Server

[validate if IIS is enabled now]
Get-WindowsFeature
```

```
[setup ServiceMonitor to manage IIS service]
Invoke-WebRequest -UseBasicParsing -Uri “https://
dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/ServiceMonitor.exe”
-OutFile “C:\ServiceMonitor.exe”

[time to roll. finally launch IIS service]
C:\ServiceMonitor.exe w3svc
```

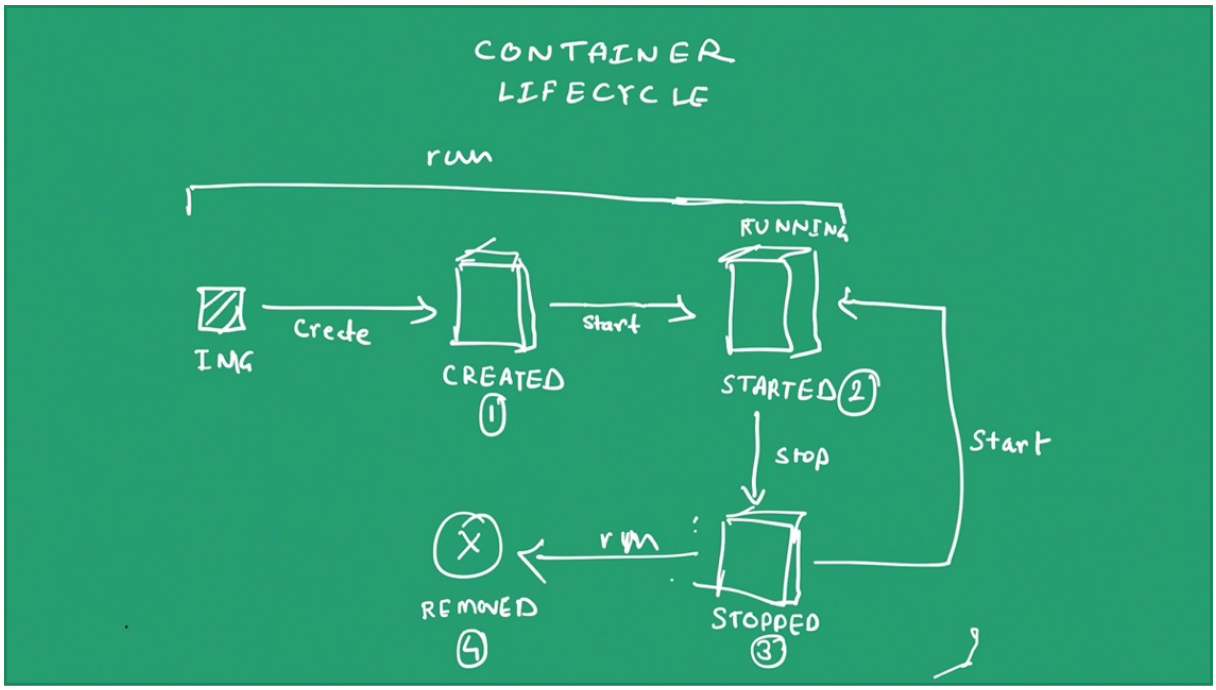
After w3svc is started, wait for a few minutes and try accessing the 80 port (as mapped while launching the container) on the docker host. You should see IIS web page available. From the above console, typing `^c` will stop the process. You could then exit the container to get back to the original command prompt/powershell console you started from.

You have learnt the basic troubleshooting of containers with logs and exec.

Managing Container’s Lifecycle

A container goes through the following four lifecycle phases,

1. CREATED
2. STARTED/RUNNING
3. STOPPED
4. REMOVED



We have already learnt how to launch container and put it in the STARTED/RUNNING state using docker run command. You could also individually create and then start containers using the following sequence,

e.g.

```
docker create -it --name twostep mcr.microsoft.com/windows/nanoserver:1809 cmd

docker start twostep
```

Note, you can not use -d option with create command as detaching a container while creating it is invalid. When you start the container in this case, it automatically started with detached mode.

To stop a running container use the following command,

```
docker stop twostep
```

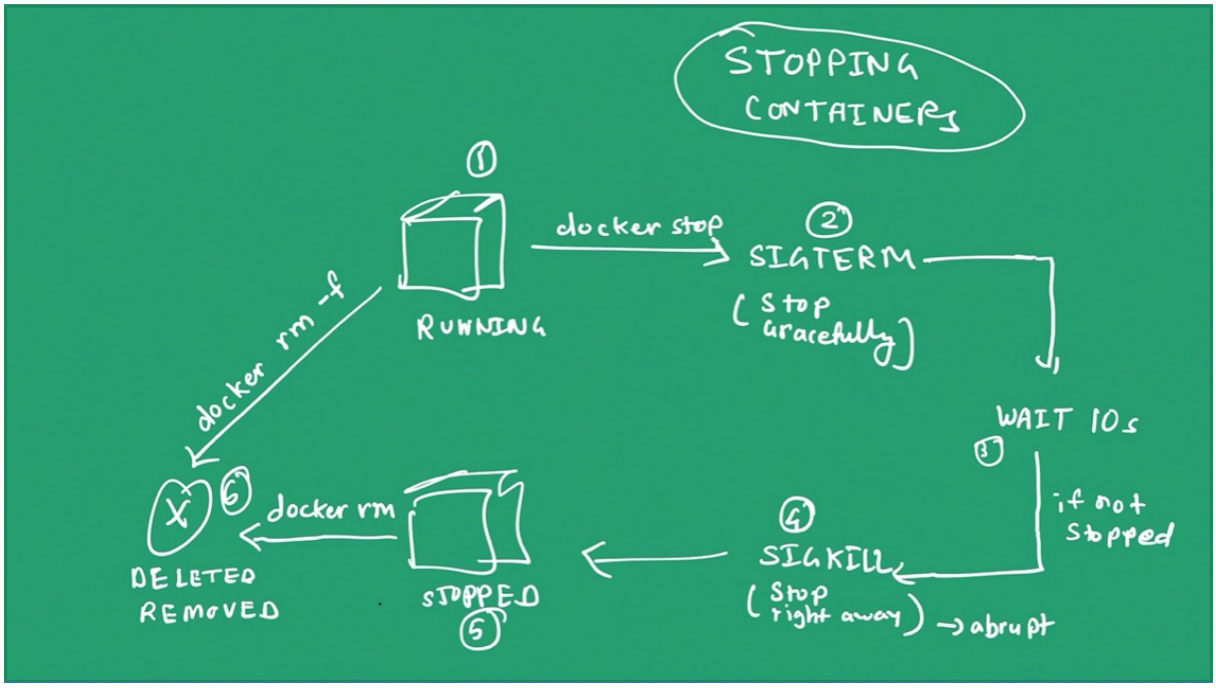
where twostep is the name of the container. You could also stop multiple containers by providing a space separated list as,

```
docker stop container_id  container_id  container_id
```

A stopped container can be started back with,

```
docker start twostep
```

> **Note** : Stopping a container does not delete the data/changes you have made to the container. When you talk about immutable deployments (e.g. kubernetes) , the containers are deleted and replaced for every update, so retaining the data is not possible in such cases. However in local environments, it can be stopped. Which means it retains all changes until its removed and can be used just like a VM.



A stopped container can be removed using `docker rm` commands as follows

```
docker rm twostep
```

A running container needs a -f option for it to be stopped and removed e.g.

```
docker rm -f  webapp
```

where `webapp` is the name of the running container.

You could also delete all the stopped containers in one go using the following command,

```
docker container prune
```

> Do note `docker container prune` will delete all the stopped containers. This may not be what you want as you have have created a few containers to use for your continuous development work, which would also be delete if in stopped state. So use this command very carefully.

Summary

In this chapter, you learnt how to launch a container, provide it with the most common options, what happens when you run a container, how to access it by using port mapping, common troubleshooting tasks with logs and exec and how to stop and remove the containers. These are essential skills for anyone who would want to get started working with containers or even use a container orchestrate engine.

#docker/windows/labs