Contents

Setting up Docker

Docker is a software technology which provides a feature called *containers* which is kind of a light form of virtual machines. It's used a lot in DevOps to setup development environments along with a variety of other uses.

Exercise 3.1 – Installing Docker Desktop

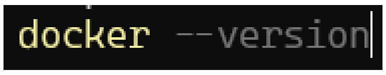
Docker can be installed on Windows, Mac and a variety of Linux operating systems (including Ubuntu, Debian, CentOS, Fedora). In this course, we're going to be using Docker in a variety of ways from acting as a simple Linux machine, to containerizing applications.

Go to the following docker web site (https://docs.docker.com/desktop/install/windows-install/), (or https://docs.docker.com/desktop/install/mac-install/ for the Mac people) download and follow the instruction to install Docker Desktop. Please be sure you read the installation instructions closely to ensure your system supports running Docker and has the needed BIOS features enabled.

Exercise 3.2 – Verifying your installation

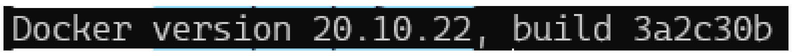
After you have it installed, run the following in a shell to check that Docker is installed and is working correctly:

1) Start a PowerShell (or Linux/Bash shell)
2) In the shell, type: `docker --version`



```
docker --version
```

If the installation went okay, you should see a response similar to the following:



```
Docker version 20.10.22, build 3a2c30b
```

N/B: The version might be different from the above

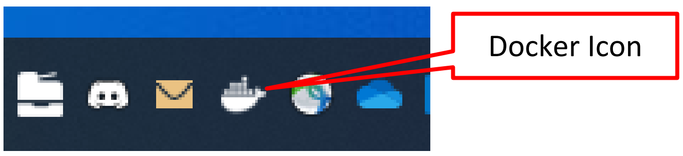
Exercise 3.3 – Using Docker with WSL 2

We should now make sure that the WSL backend is enabled in Docker. With this approach, Docker Desktop runs under WSL2 rather than a traditional VM.

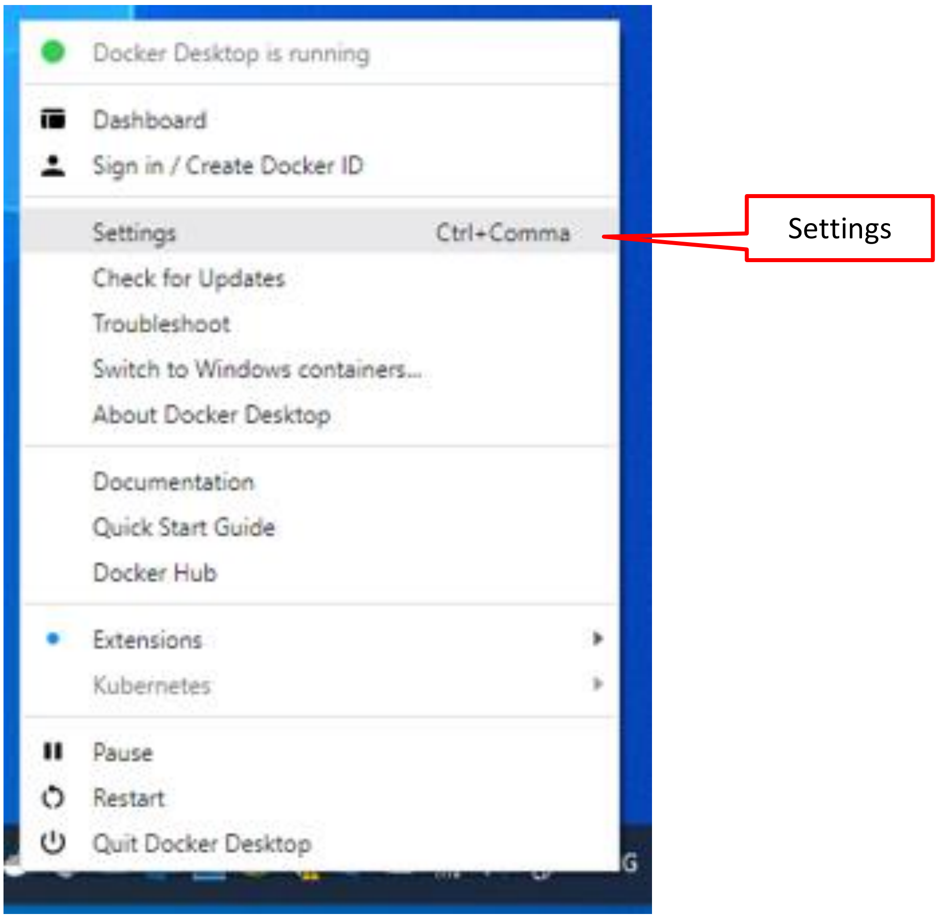
3.3.1 Use the WSL 2 based engine

It provides better performance than Hyper-V backend.

1) Right-click on the Docker icon in your system tray

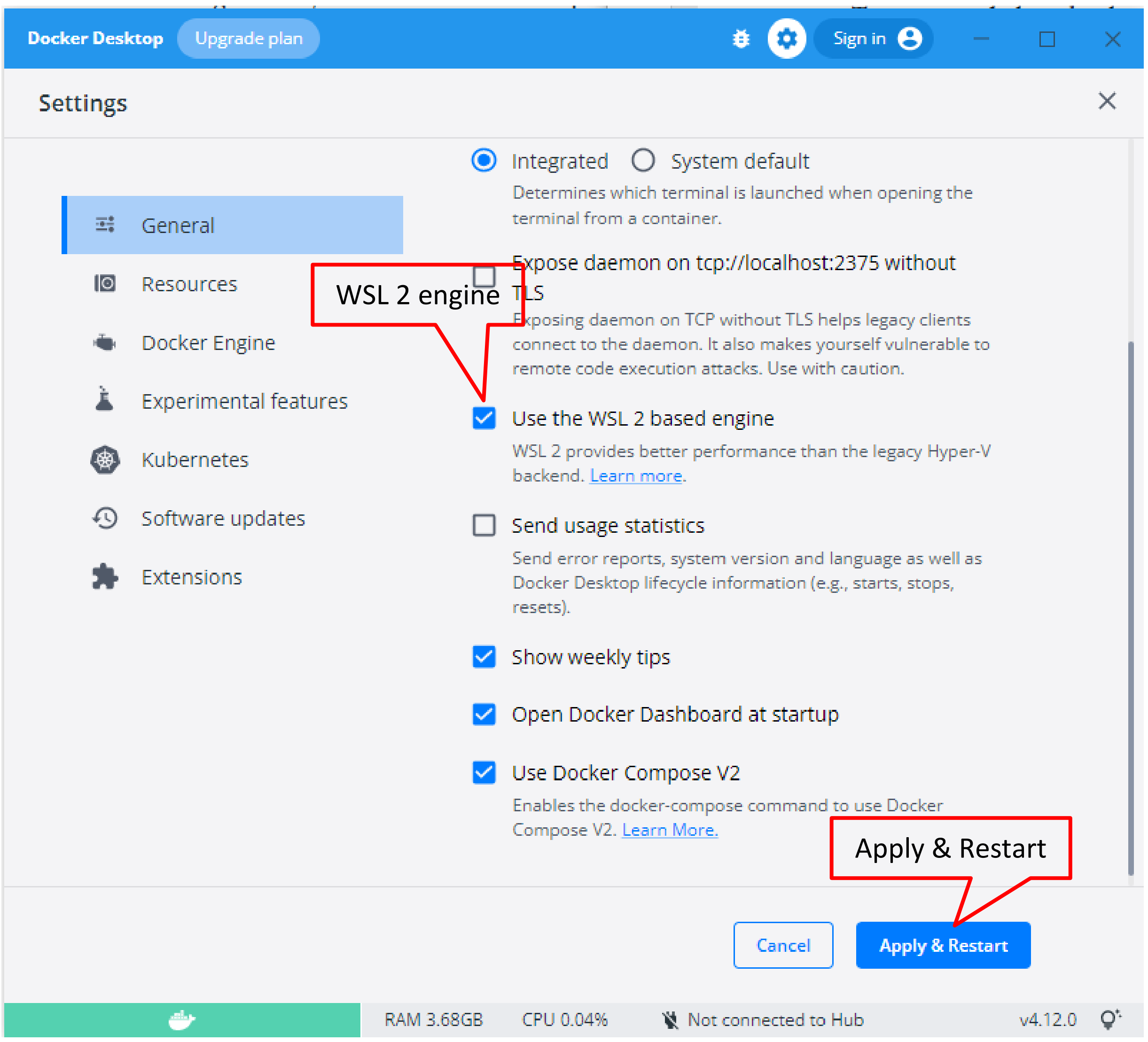


2) Choose Settings



3) Use the WSL 2 based engine

Ensure this option is selected in order to configure Docker Desktop to run under WSL 2. If not, set the tick mark and click on Apply & Restart:



[✓ Integrated] System default
Determines which terminal is launched when opening the terminal from a container.

☐ Expose daemon on tcp://localhost:2375 without TLS
Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.

☑ Use the WSL 2 based engine
WSL 2 provides better performance than the legacy Hyper-V backend. Learn more.

☐ Send usage statistics
Send error reports, system version and language as well as Docker Desktop lifecycle information (e.g., starts, stops, resets).

☑ Show weekly tips

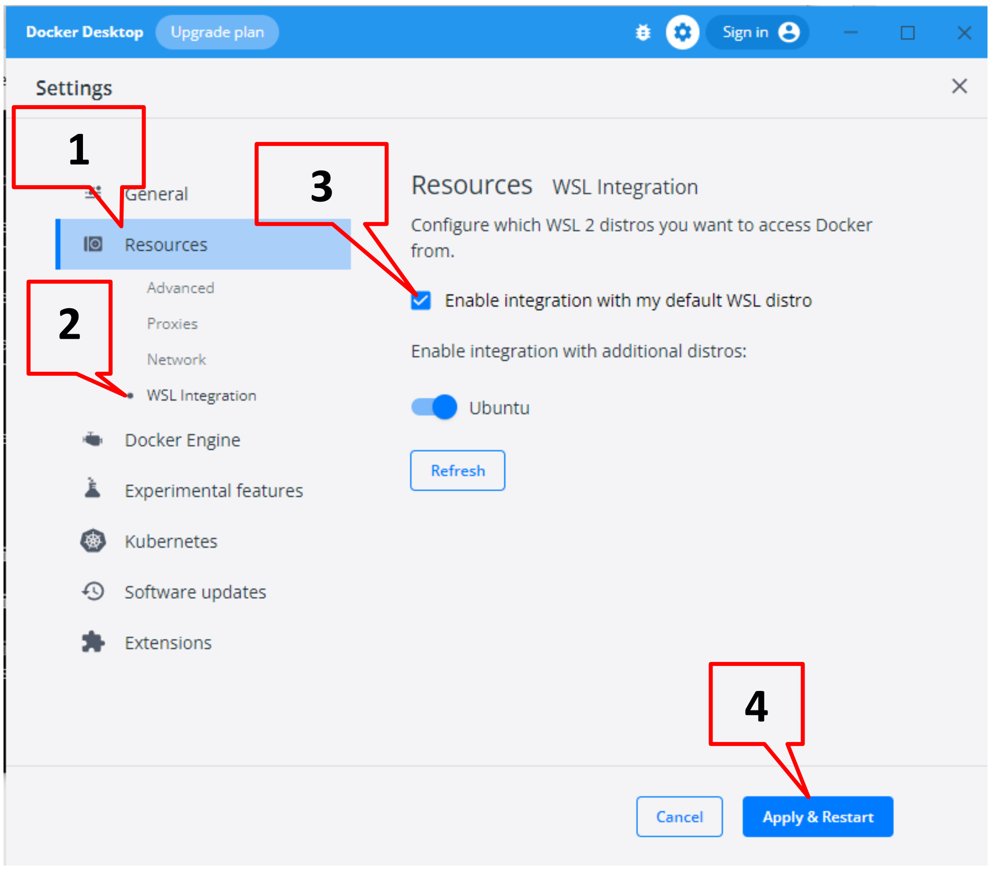
☑ Open Docker Dashboard at startup

☑ Use Docker Compose V2
Enables the docker-compose command to use Docker Compose V2. Learn More.

3.3.2 Enable Integration with your default WSL distro

The last step in the setup is about configuring which WSL 2 distributions you want to access Docker from. I suggest that you enable integration with your default WSL distro as shown below:

1) Click on Resources
2) Click on WSL Integration
3) Select Enable integration with my default WSL distro
   (N/B: You may also Enable integration with additional distros else leave it disabled)
4) Click on Apply & Restart



Exercise 3.4 – Pull and Run Your first image

We can already check Docker with existing images by pulling and running a sample from the Docker registry.

1) Start Docker Desktop if you have not already done so.
2) Start a PowerShell
3) In the shell, type: `docker run --rm hello-world`

```
docker run --rm hello-world
```

If the installation went okay, you should see a response similar to the following:

```
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Exercise 3.4 – Pull and Run a Ubuntu image

1) Start a Linux (WSL) shell
2) Start Docker Desktop (DD) if you have not already done so.
3) Open DD and take a look at the images you have currently available locally.
4) In the shell type: `docker pull ubuntu`
5) Look at the DD images again – you should now see that you have an Ubuntu image.
6) In the shell type: `docker run --name scripttest -it ubuntu`
   This accomplishes:
   a. It creates a new container called scripttest (`--name scripttest`) based on the Ubuntu image.
   b. It runs the default executable in the Ubuntu image (which is a Bash shell) (`run`)
   c. It runs the default executable (i.e. the Bash shell inside the container) interactively (`-it`). This means that we can now type commands into the Bash shell that runs inside the container, and whatever output the shell generates inside the container is directed to the terminal where we started the container.
7) Type `ls -al`
8) Type `pwd`

The stuff you see in step 7-8 should look different than it normally does when you start a Linux shell. To verify this:

9) Start another (new) Linux shell and type the same two commands into that:
10) Type `ls -al`
11) Type `pwd`

It's important to understand that the reason for the difference is that although we do all the work in step 1-8 and 9-10 in a Linux shell, and we run a Bash shell in both cases, the stuff we see in step 7-8 comes from *a Bash shell that runs inside a container*. So we have in effect 3 different Bash shells running at this point:

- Bash shell #1 running in our "normal" Linux/WSL environment, started in step 1
- Bash shell #2, running inside a new container, and Bash shell #2 was started via a command to Bash shell #1 in step 6.
- Bash shell #3, running in our "normal" Linux/WSL environment, started in step 9.

12) Finally, stop Bash shell #2 in the container by typing `exit`
13) Stop shell#1 and shell#2 by typing `exit`

Exercise 3.5 – Starting and stopping an existing container

Once we have created a container from some image, we can start and stop it as many times as we want. Sometimes we create containers we'll be using for a long time, and at other times we create containers that's only used once, perhaps even for a few seconds, and then discarded.

In exercise 3.4 we created a container based on a Ubuntu image, and we called that container `scripttest`. If we know the name of a container (you can always find an overview of the containers you have created by looking in Docker Desktop), it is very easy to restart it with the docker start command.

1) Start the scripttest container again using this command in a Linux shell:
   `docker start -i scripttest`

2) NOTE: The container will start and because of the -i option it will run interactively. That means that all the commands we type in the following steps of this exercise will go to and from the shell that runs inside the container.

3) Change to the /tmp folder by typing `cd /tmp`
4) Create a small file by typing: `echo "howdy partner" > myfile.txt`
5) Verify that the file exists: `ls -al`
6) Verify the contents of the file: `cat myfile.txt`
7) Now stop the shell/container again, but this time, instead of typing exit, try to press Ctrl-D.

Ctrl-D has the same effect as typing exit because Ctrl-D in Linux is used as an End-of-File marker (EOF). And when a shell sees EOF in its input (i.e. the stuff it sees from the keyboard) it terminates.

8) Now restart the container like you did in step 1.
9) Check that the file you created in step is still there and holds the same content.
10) Stop and restart the container a few more times.

Exercise 3.6 – Updating a container

A container is kind of a small virtual Linux computer. So, we can update the contents (programs) inside the container as we have seen it done in the "normal" Linux/WSL shell, i.e. by using the apt-get command inside the container.

Update your `scripttest` container by following these steps to install the micro editor:

1) Restart the container.
2) Type: `apt-get update`
3) Type: `apt-get install micro` and answer y to install it.

So now you have micro installed inside your container. Try running it like this to read and update the contents of the file we created just before.

4) `cd /tmp`
5) `micro myfile.txt`
6) Add some extra text to the file and save the result. Remember that in micro you can always get a quick command overview by pressing ALT-g.
7) Stop the container again (using exit or Ctrl-D as you please).
8) Restart the container
9) Add some more text to the file /tmp/myfile.txt using micro.
10) Stop the container.

Exercise 3.7 – Working inside a container (using scripts)

Now we will create and run a script inside our `scripttest` container.

1) Start the container. The shell that gets started we will call shell#1
2) `cd /tmp`
3) `micro script1_input.sh`
   The file `script1_input.sh` does not exist but micro will create it for us if we write something and saves the file.
4) Open another Linux/WSL shell which we will call shell#2
5) Go to the folder where you put the scriptdemo-stuff.
6) Show the contents of the file script1_input.sh- by typing `cat script1_input.sh`
7) Now do a copy-paste of the file from the shell#2 window into the shell#1 window where micro is running.
8) Make micro save the file and exit micro.
9) Make the file `script1_input.sh` executable: `chmod +x script1_input`.sh
10) Run the script: `./script1_input.sh`

11) Make some changes to the script
12) Save the script
13) Run the script
14) Repeat step 11-14 a few times, making changes to the file and see if it still runs

15) Repeat steps 3-10, only this time copy the file `script2_loop.sh` from shell#2 to a new file (also called `script2_loop.sh`) in shell#1. Use the micro + copy/paste trick again.
16) Use micro to create a file with this content:

```
1356
1597
1671
1691
1365
1772
1112
1747
1337
1272
1060
1417
1055
1422
1728
1772
1010
```

17) Now, inside your nice container, use the `script script2_loop.sh` to sort and list the contents of the file created in step 16.