Docker for windows side-by-side with Hyper-V Vms

In this step-by-step guide we will install Hyper-V alongside the Windows Subsystem for Linux, and install docker, both in WSL and Windows, but without installing Docker Desktop for Windows.

Intro

Because more and more Kofax Products can be run as Docker containers, we would like to be able to run those containers on our local laptop for demonstration purposes and proof of concepts. However, we cannot (yet) do without regular Virtual Machines, because not all Kofax software can be run as containers.

We also want to be able to run Windows based containers side by side with Linux based containers, because some products, notably RPA, consist of components which cannot all be run on Linux based containers, for example, even though RPA's management console, roboserver and several other components can be run as Linux based containers, the Document Transformation Service cannot, because it is a Internet Information Service application.

I use VirtualBox for my virtual machines and can host a docker service in an Ubuntu VM to run Linux based containers. However, to run windows based containers, we need to run docker for windows, which implies that Hyper-V should be activated on the Laptop. Hyper-V is able run alongside VirtualBox, but Docker Desktop for Windows (the standard solution pushed by Docker to run containers on Windows 10) doesn't play nice with VirtualBox and isn't free. Also, VirtualBox is not free for commercial use.

So I wanted to get rid of VirtualBox, use Hyper-V instead and at the same time have an alternative to Docker Desktop for Windows.

This is in fact possible, and although this means we will not have a nice graphical user interface*, we can build images and run containers from a command line without issue, while at the same time run virtual machines if we need them.

!!! note * I am aware that there are GUI alternatives, for example Rancher, but these are still in heavy development and I have not tried them and probaby won't as a CLI is fine for me.

Goals

We want to be able to run side-by-side on a single laptop

- Virtual machines
- Linux based containers
- Windows based containers

We do not want to use anything for which we would need a commercial license

- no VirtualBox
- no VMWare
- no Docker Desktop for Windows

External references

Although it has been quite a bit of work to compile all of this, none of this can be called orginal research and relies heavily on research and documentation provided by others:

- https://lippertmarkus.com/2021/09/04/containers-without-docker-desktop/
- https://learn.microsoft.com/en-us/windows/wsl/install
- https://learn.microsoft.com/en-us/windows/wsl/basic-commands

The basic docker for windows is still provided free of charge, unlike Docker Desktop for Windows, but the installers are a bit hidden on the Docker site. Check the link below to see what the latest version is. In the steps below we used `docker-20.10.21.zip`, but a newer version might be available by the time you read this.

https://download.docker.com/win/static/stable/x86_64/

Prerequisites

- W10 fully updated as of 17/11/2022 (Only Pro and Enterprise, Home edition will not work)
- Machine connected to the internet (To download all kinds of packages etc.)

Step by step installation

- enable Hyper-V and Containers

  We need the first to run Virtual Machines and the Windows Subsystem for Linux, and the second for Docker.

- convert any existing VirtualBox VM we want to keep

  This might be optional for you, but I want to have a backup I can go back to even if something doesn't work out.

- configure Windows to run docker for Windows

Not difficult but Docker keeps this a bit hidden and pushed Docker Desktop everywhere.

- install the Windows Subsystem for Linux

  Not strictly necessary as we can run Docker for Linux inside a Virtual Machine on Hyper-V, but nice to have as this is light weight and well integrated in the rest of Windows.

- configure WSL to run docker for Linux

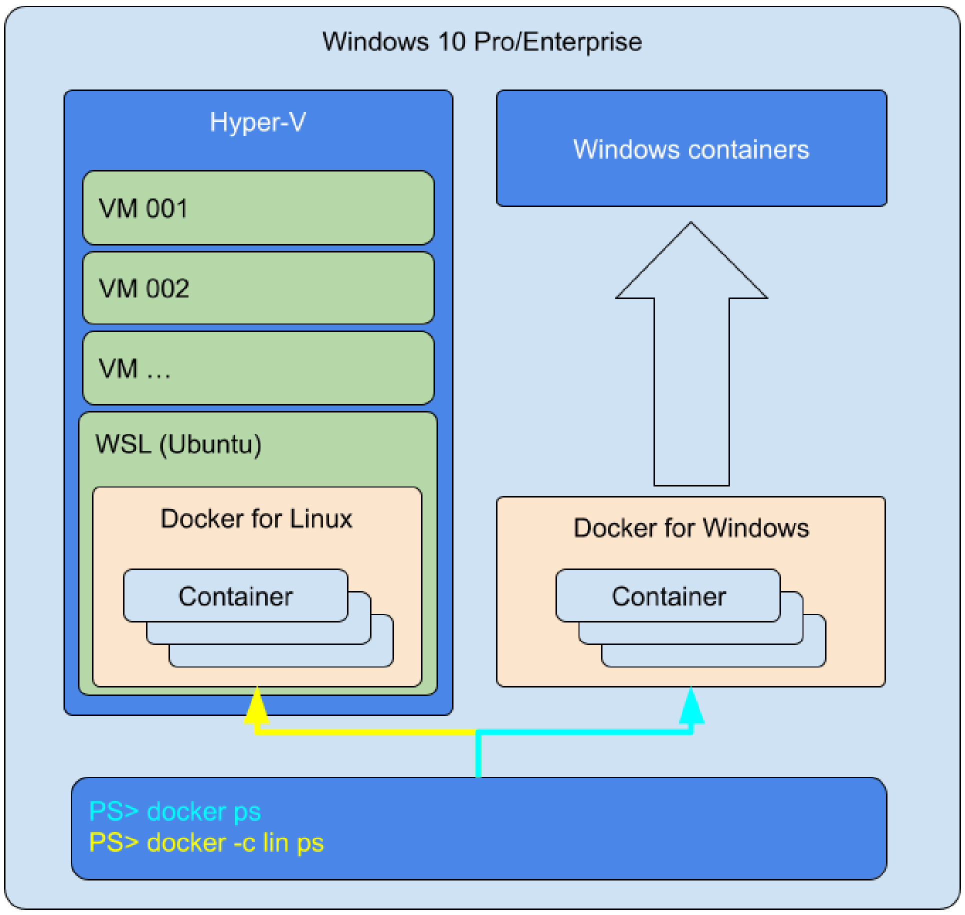
  Basically installing a few packages and you should be good to go, but Ubuntu in WSL lacks systemd so we need to make sure that is all up and running as well.

- expose the linux dockerd as a context in windows

  So we can build images and run containers both in Linux and Windows from the same Powershell.

Architecture

Because it can be quite confusing to grasp the relations between all the components, this diagram might offer some help:



Enable Hyper-V and Containers

In an elevated Powershell:

```
Enable-WindowsOptionalFeature -Online -FeatureName containers -All
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

Then reboot.

After the reboot, the Hyper-V manager will be available in the Windows Start menu in the group Windows Administrative tools.

Convert any existing VirtualBox VM we want to keep

Existing VirtualBox VMs cannot be run directly under Hyper-V, and as far as I know it is not possible to import Virtual Appliances (.ova) into Hyper-V, but we can convert the virtual disk to a format that can be used by Hyper-V.

With Hyper-V installed we can convert .vhd drives to the .vhdx format required for Hyper-V VMs.

If you VirtualBox VM is not yet a single .vhd (because it has snapshots and/or is in .vdi format) Use the VirtualBox manager to clone you VM first and specifiy that you only keep the last state and convert to .vhd.

In VirtualBox Manager:

- Select your VM & Clone (make sure to select 'Full clone' and 'Current machine state')

Then convert the .vdi to .vhd

- File --> Virtual media manager
- Locate and select the .vdi you created in the previous step
- Click Copy (select 'VHD' and 'Dynamically allocated')

Then convert that .vhd to a .vhdx in an elevated powershell (an example is shown below)

```
cd "C:\Users\michel.anders\VirtualBox VMs\Docker Clone"
convert-vhd  -Path '.\Docker Clone_copy.vhd' -DestinationPath '.\Docker
Clone_copy.vhdx'
```

You can test any converted VMs before proceeding by creating a new VM in the Hyper-V manager that refers to the .vhdx.

Configure Windows to run docker for Windows

We need to download and install docker for windows (which is not the same as Docker Desktop for Windows!) first, and the run it as a service. That's just a few commands and we test whether docker can run a container as the final step. This will run a windows based container, as you will be able to verify by looking at the output.

In Powershell w. elevated privileges

```
cd \Temp
curl.exe -o docker.zip -L -O
```

```
https://download.docker.com/win/static/stable/x86_64/docker-20.10.21.zip
Expand-Archive docker.zip -DestinationPath C:\
[Environment]::SetEnvironmentVariable("Path", "$($env:path);C:\docker",
[System.EnvironmentVariableTarget]::Machine)
$env:Path = [System.Environment]::GetEnvironmentVariable("Path","Machine")
dockerd --register-service
Start-Service docker
docker run hello-world
```

We also want to install docker-compose. Unlike for Linux, this is only available as a standalone, not as a plugin. (see: https://docs.docker.com/compose/install/other/)

```
[Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
Start-BitsTransfer -Source
"https://github.com/docker/compose/releases/download/v2.12.2/docker-compose-
Windows-x86_64.exe" -Destination $Env:ProgramFiles\Docker\docker-compose.exe
docker-compose version
```

If you want to store your images, containers, etc. ... in a different location than the default (`C:\ProgramData\docker`) create the following file: `C:\ProgramData\docker\config\daemon.json` with this content

```
{
    "data-root": "d:\\dockerWindowsData"
}
```

The directory is what I use, it can be anything of course.

Reboot to have this take effect.

Install the Windows Subsystem for Linux

Installing the Windows Subsystem for Linux (version 2) used to be a hassle in older revisions of Windows 10, but nowadays it is a single command.

In Powershell w. elevated privileges

```
wsl --install
```

Then reboot.

This will make Ubuntu available directly in the Start menu. It is possible to install a different Linux distro, but here we used Ubuntu and the following steps rely on this.

Configure WSL to run docker for Linux

First enable systemd/distrod inside WSL2. This provides easy management and configuration of Linux daemons, something we need to make sure docker starts when we start windows.

see: https://github.com/nullpo-head/wsl-distrod

Inside your WSL (Ubuntu) terminal:

```
sudo bash
```

```
curl -L -O "https://raw.githubusercontent.com/nullpo-head/wsl-
distrod/main/install.sh"
chmod +x install.sh
./install.sh install
/opt/distrod/bin/distrod enable --start-on-windows-boot
```

!!! note be warned that if you copy this last set of commands from a markdown page as shown on GitHub for example, the last command is in fact not executed if you paste the whole lot into your terminal. Make sure to to cut/paste this last one seperately.

That last command will inform you that you'll need to provide your Windows credentials in order to install this service.

In Powershell w. elevated privileges

```
wsl --terminate Ubuntu
```

If you open your WSL terminal again, your Ubuntu now has systemd.

Then we install the docker service, start it and test it.

```
sudo bash
```

```
apt-get install -y apt-transport-https ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
apt-get update
apt-get install -y docker-ce docker-ce-cli containerd.io docker-compose-plugin
docker run hello-world
```

This hello-world container will be based on a linux image.

You may want to change the location of the image and container data. The default is in /var/lib/docker and the root file system of the WSL system actually takes up space on C: and that space is limited. However, I had all kinds of errors relating to the filesystem when pulling some images, so in the end I unregistered Ubuntu and installed it again and left things to their default.

(see: https://linuxconfig.org/how-to-move-docker-s-default-var-lib-docker-to-another-directory-on-ubuntu-debian-linux)

```
sudo bash
```

```
systemctl stop docker.service
systemctl stop docker.socket
# in an editor, change the ExecStart line to
# ExecStart=/usr/bin/dockerd -g /mnt/d/docker -H fd:// --containerd=/run/containerd/containerd.sock
joe /lib/systemd/system/docker.service
mkdir -p /mnt/d/docker
cp -rp /var/lib/docker/ /mnt/d/docker   # will complain about a block device, can be ignored
systemctl daemon-reload
systemctl start docker.service
docker run hello-world
```

Expose the linux dockerd as a context in windows

Now we have two docker installations, one inside WSL and one directly in Windows. But we want to be able to control both of them from a single Powershell. This is possible is we expose the port that dockerd inside the WSL is listening on, and configure the WSL docker as a context to be used in our windows environment.

Inside your WSL (Ubuntu) terminal:

```
sudo bash
```

```
cp /lib/systemd/system/docker.service /etc/systemd/system/
sed -i 's/\ -H\ fd:\/\/\/\ -H\ fd:\/\/\ -H\ tcp:\/\/127.0.0.1:2375/g' /etc/systemd/system/docker.service
systemctl daemon-reload
systemctl restart docker.service
```

In Powershell w. elevated privileges

```
docker context create lin --docker host=tcp://127.0.0.1:2375
```

Now we can interact with both dockerd deamons from within a Powershell w. elevated privileges:

```
# the windows dockerd
docker ps
# the WSL dockerd
docker -c lin ps
```

Tests

Building and deploying RPA images in docker in WSL

The proof of the pudding for Docker inside WSL is checking if we can build an run images.

This would mean

- unpacking the Linux RPA package (I used a RPA 11.4 prerelease, `KofaxRPA-11.4.0.tar.gz`)
- copying the file `docker-compose-kapplets-rfs.yml` to the extracted folder
- copying secrets etc.
- and then verifying the yaml, building the images and deploying the services

!!! note secrets in docker-compose on a remote context do not work. So unless we go the docker swarm way or start using Kubernetes we cannot use secrets this way.

In an elevated Powershell

```
docker context use lin
cd D:\KofaxRPA-11.4.0.0.184-Linux\
# edit the docker compose file with license info (optional, you can also enter
these in the MC's first run)
# then verify your config
docker-compose -f .\docker-compose-kapplets-rfs.yml --project-directory . config
# bring up one of the postgres databases, just to verify we can
docker-compose -f .\docker-compose-kapplets-rfs.yml --project-directory . up
postgres-service -d
# build the managementconsole-service
docker-compose -f .\docker-compose-kapplets-rfs.yml build managementconsole-
service
# build everything (mc, roboserver, kapplets, rfs)
docker-compose -f .\docker-compose-kapplets-rfs.yml build
# list them
docker images
# deploy the whole setup
docker-compose -f .\docker-compose-kapplets-rfs.yml up -d
# containers will be up quickly, but the MC might take quite a while to be fully
up and running (tomcat has to start etc)
# after a minute or two, we should be able to access te management console
# In chrome:  http://localhost  (login with admin/admin or whatever you configured
in the docker-compose, and enter
# your license info if prompted)
# likewise, the kapplets server should be on http://localhost:84
# (but make sure to set relevant entries in your hosts file)
```

Entries for your hosts file (I don't know yet how to have containers accessible on the external interface)

```
127.0.0.1 robotfilesystem-service
127.0.0.1 postgres-service-data
127.0.0.1 managementconsole-service
```

Building and deploying RPA images in docker for Windows

TODO

Questions to solve

- can we use docker compose to start containers in different context from a single docker-compose file? unknown; you can deploy all services to a single context with --context , but unclear if a docker-compose file can be configured to deploy services to different contexts
- can you expose usb devices inside containers? unknown
- can we bridge WSL? should be possible, see: https://github.com/luxzg/WSL2-fixes, but I cannot get to to work yet. Perhaps because I don't run Windows 11 and/or don't have the prerelease of the WSL (my wsl command does not recognize the --version option)
- security inside WSL we execute docker commands as root. It would be better to allow just some unprivileged users to run docker