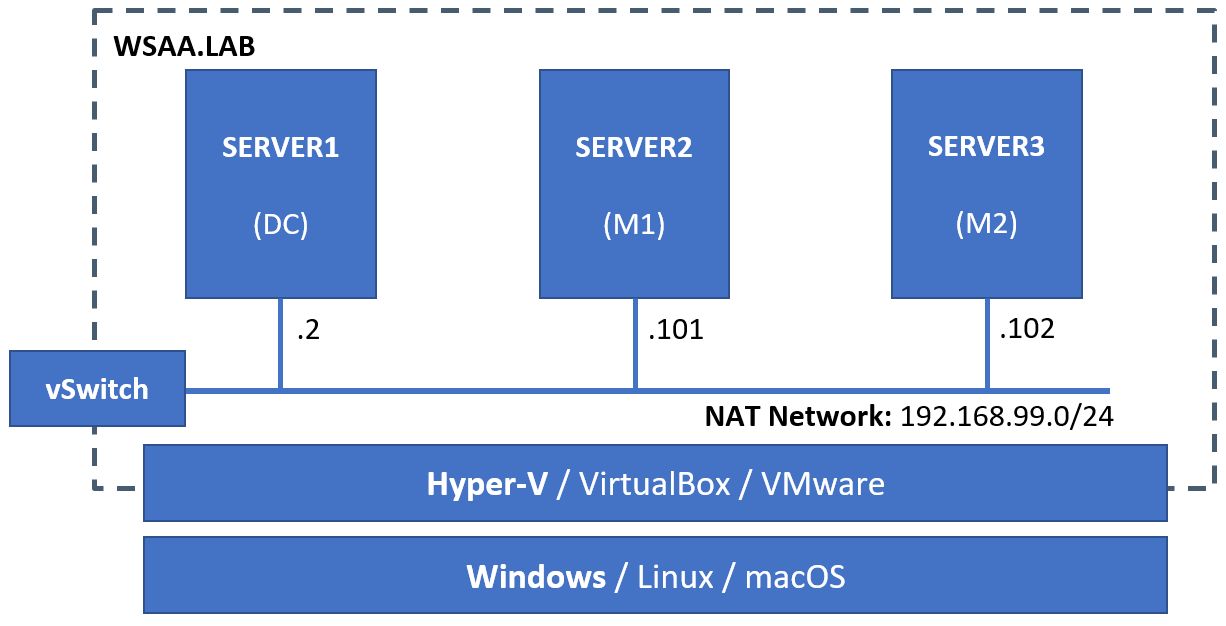
# Practice M3: Containerization

For the purpose of this lab and the course, we will consider that we are working in a pure Windows environment either on-premise or in the cloud and using **Hyper-V** as a virtualization solution. All tasks can be achieved under different configuration (another host OS or virtualization solution) with the appropriate adjustments

For this practice, we will need a small infrastructure with three VMs. One will be the domain controller and two member servers. All could be with **Desktop Experience** or **Core**. **Standard** edition is enough



## Part 1: Containerization 101

### Install Components for Windows Containers

Connect to one of the member servers, for example **SERVER2**

Open a **PowerShell** session with **Run as Administrator**

Install the prerequisites

**Install-WindowsFeature Containers -Restart**

Install the appropriate provider module

**Install-Module -Name DockerMsftProvider -Repository PSGallery -Force**

Then install the **Docker** package with

**Install-Package -Name Docker -ProviderName DockerMsftProvider**

You can restart the computer (**Docker** service will start automatically) or just start the **Docker** service

**Start-Service docker**

### Install Components for Hyper-V Containers

Connect to one of the member servers, for example **SERVER3**

Open a **PowerShell** session with **Run as Administrator**

Install the prerequisites

**Install-WindowsFeature Containers, Hyper-V -IncludeManagementTools -Restart**

Install the appropriate provider module

**Install-Module -Name DockerMsftProvider -Repository PSGallery -Force**

Then install the **Docker** package with

**Install-Package -Name Docker -ProviderName DockerMsftProvider**

You can restart the computer (**Docker** service will start automatically) or just start the **Docker** service

**Start-Service docker**

### Alternative Option

*If everything worked as expected, please skip this section*

If due to some reason the **PSGallery** is not available, there is an alternative way

Download the **Docker** archive

**Invoke-WebRequest -UseBasicParsing -OutFile docker-20.10.7.zip https://download.docker.com/win/static/stable/x86\_64/docker-20.10.7.zip**

Extract the archive

**Expand-Archive docker-20.10.7.zip -DestinationPath $Env:ProgramFiles -Force**

Remove the archive (if you want)

**Remove-Item -Force docker-20.10.7.zip**

Add **Docker** to the path for the current session

**$env:path += ";$env:ProgramFiles\docker"**

Or, optionally, modify **PATH** to persist across sessions

$**newPath = "$env:ProgramFiles\docker;" + [Environment]::GetEnvironmentVariable("PATH", [EnvironmentVariableTarget]::Machine)**

**[Environment]::SetEnvironmentVariable("PATH", $newPath, [EnvironmentVariableTarget]::Machine)**

Register the **Docker** daemon as a service

**dockerd --register-service**

Start the **Docker** service

**Start-Service docker**

### Pre-flight check

*Execute the following steps on both servers*

Check that **Docker CLI** is accessible and the **Docker** daemon reachable

Get some information about both the client and the daemon

**docker version**

Ask for system-wide information with

**docker info**

In the output of the second command we can see what is the default isolation mode. On **Windows Server** it is **process**, while on **Windows 10** it is **hyper-v**

If both return information and no errors, we can move on

### Hello container world

*Execute the following steps on both servers*

Now, that we have **Docker** up and running, let’s execute a simple hello world container

**docker run hello-world:nanoserver**

We should see a message that our installation is correctly set up

If you decide to follow the advice given by the hello world application, you will end up with an error saying that there is no tag latest. Microsoft has different strategy on tags, we will see it in a minute

### Searching for images

*Execute the following steps on one of the servers*

We cannot know all images by name, so we can do a search by a keyword

On the terminal this done like

**docker search windows**

Unfortunately, the output is not very informative

Alternatively, you can visit the famous **Docker Hub** at <https://hub.docker.com/search>

Let’s go there and ask again for **windows**

Now, visit one of the official repositories of **Microsoft**, scroll down a bit, and voila, there is no **latest** tag

Okay and now which version and tag to use?

For the versions – it depends on what we want to do

For the tags – first, we must check our exact version. This can be done with the **winver** command

This is especially true when working with **process isolation** mode

When using **Hyper-V** isolation, we can void this rule

### Download (install) an image

*Execute the following steps on both servers*

So far, we executed one **run** command on a brand-new installation without any images

What happed behind the scenes? **Docker**, knowing that we don’t have the image locally, went out, found one for us, downloaded it and then ran a container out of it

There is another approach. Knowing what we need, we can issue a special command to download the image locally

**docker pull mcr.microsoft.com/windows/servercore:1809**

Once the image has been downloaded (it can take some time), we can check the list of our local images with

**docker image ls**

### Create a container

*Execute the following steps on the server with Hyper-V installed*

Let’s use a little bit modified version of the hello world **run** command

**docker run -it mcr.microsoft.com/windows/servercore:1809 powershell**

This will start a container and will create an interactive terminal session with **PowerShell** to it

It looks and feels exactly like a real thing

Try a few commands (for example: **hostname**, **dir**, **ping**, **tracert**, etc.)

Once you are done execute the **exit** command

You will be returned in your host session

Now, let’s check what containers we have running with

**docker container ls**

It appears that there aren’t any. It is true, we just terminated the only one. When we executed **exit**, we closed the **PowerShell** session which was the only running process (in a way) in the container

If we modify the command a little bit to

**docker container ls -a**

We will see all containers running and stopped

Okay, so how we exit from container without terminating it? There is a key combination **Ctrl+P** and then **Ctrl+Q**. Try it and see yourself. First create a container with

**docker run -it --name C1 mcr.microsoft.com/windows/servercore:1809 powershell**

And now, how to return to the container? We can attach back to it. First, we need to know its ID or name. This can be seen from the output of

**docker container ls**

But wait, we know the name because we set it to custom value

Now, to return to the second (still running) container, we must execute

**docker container attach <container-id-or-name>**

Exit the session again but do not terminate the container (use the key combination)

### Explore the different isolation levels

*Execute the following steps on the server with Hyper-V installed*

Having still one container with **process** isolation level running, let’s ask for the processes running in it

**docker container top C1**

Pay attention to the PIDs. Get the one of **powershell.exe** and see if you can find it as process on the host

Let’s create another container with different isolation level

**docker run -it --name C2 --isolation hyperv mcr.microsoft.com/windows/servercore:1809 powershell**

Close the session but do not terminate the container (use the key combination)

Ask for the processes running on the **C2** container

**docker container top C2**

They should be with smaller IDs and you should not be able to find the corresponding processes on the host

### Isolation levels #2

Return to C1

**docker container attach C1**

Start the following ping command

**ping -t localhost**

Close the session to the container by pressing the key combination

Repeat the procedure for **C2**

Now, being on the container host, ask for the processes of **C1**

**docker container top C1**

And then for the ones of **C2**

**docker container top C2**

Now, try to find those two **PING** processes on the container host. Did you succeed?

Finally, stop both containers with

**docker container stop C1 C2**

## Part 2: Containerization 102

### Exchange data between the container host and a container

Let’s create a folder on the container host. For example, **C:\DATA**

Create a simple **readme.txt** file there with some content

Now, let’s create a container, but a modified **run** command

**docker run -it --name C20 --volume C:\DATA:C:\DATA mcr.microsoft.com/windows/servercore:1809 powershell**

*Note, that some of the options have long (--volume) and short (-v) form, while others (--name) don’t*

Being in the container, navigate to the file and check its contents

**cat .\data\readme.txt**

Add something at the end of the file or add a new file

**"C20 was here" | Out-File C:\data\readme.txt -Append**

Check again the content of the file. If you see some extra spaces, it is due to encoding mismatch. Save the file in Unicode and try again

Once done, exit (and terminate) the container

On the container host, go and see if the changes persisted. They should be there

### Exchange data between containers

Let’s create a volume

**docker volume create myvol**

Start a new container and map the volume to it

**docker run -it -v myvol:c:\temp mcr.microsoft.com/windows/servercore:1809 powershell**

Check the files and folders in **C:\**

Create a **readme.txt** file in the **C:\temp** folder

**"This file was created by a container" | Out-File C:\temp\readme.txt**

Check that the content is there

**cat c:\temp\readme.txt**

Terminate the container by closing the session with

**exit**

Open File Explorer on the container host and navigate to

**C:\ProgramData\docker\volumes\myvol\\_data**

Add some text and close the file

Start one new container with the volume mounted to **c:\data** folder

**docker run -it --name C21 -v myvol:c:\data mcr.microsoft.com/windows/servercore:1809 powershell**

Check the file content

Close the session without terminating the container

Start another container with the volume mounted to **c:\files** folder

**docker run -it --name C22 -v myvol:c:\files mcr.microsoft.com/windows/servercore:1809 powershell**

Check the file content

If you wish, add something

Close the session to the container

Stop both containers

**docker container stop C21 C22**

If you wish, try with another container, but this time with isolation mode set to **Hyper-V**

### Container with web server

Let’s check this repository

<https://hub.docker.com/_/microsoft-windows-servercore-iis>

It appears, that we can run a web server (IIS) in a container, so let’s do it

Execute the following command

**docker run -d -p 8080:80 --name web1 mcr.microsoft.com/windows/servercore/iis**

Check the list of running containers

**docker container ls**

Open a browser and navigate to <http://localhost:8080>

You should see the default welcome page of **IIS**

Alternatively, get the IP address of the container with

**docker inspect -f "{{ .NetworkSettings.Networks.nat.IPAddress }}" web1**

And use it in a browser tab. For example, <http://172.21.21.82>

Remove the container by executing

**docker container rm --force web1**

### Set a custom page

Now, let’s put the mount binging approach in use again

Create a folder on the container host. For example, **C:\web**

Create a custom **index.html** in it with a simple content. For example

**"My Custom Web Page" | Out-File C:\web\index.html**

Now, start an IIS based container but with different command

**docker run -d -p 8080:80 --name web1 -v c:\web:c:\inetpub\wwwroot\ mcr.microsoft.com/windows/servercore/iis**

Open a browser and navigate to <http://localhost:8080>

You should see your custom index page

If not, some troubleshooting may be needed

Open a session to the container with

**docker exec -it web1 powershell**

And explore what is wrong and correct it

Please note, that closing the session with **exit** won’t terminate the container

Should you want to stop and remove the container, execute

**docker container rm --force web1**

### Modify Docker configuration

Currently we are running with the default configuration

Let’s modify it in order to turn on **experimental** features

Open the file **C:\ProgramData\docker\config\daemon.json** if it doesn’t exist, create new

If you are editing an existing file, then add the following line between the curly ({}) braces

**"experimental": true,**

If you are creating a new file, then type the following

**{**

**"experimental": true**

**}**

Save and close the file and then restart the daemon

**Restart-Service docker**

Then check if the setting has been applied by executing

**docker info**

And look for **Experimental: true**

One of the benefits of this setting is that it will allow us to run **Linux** containers

In the same manner, we can modify other aspects of **Docker**’s behavior

List of configuration settings (not all are applicable to **Windows**) can be found here:

<https://docs.docker.com/engine/reference/commandline/dockerd/>

## Part 3: Containerization 103

### Create own simple image

For this exercise, we will start with a base image that gives us most of the features we want and then we will add just a few bits

Create a folder on the container host

**mkdir C:\image1**

Navigate to the folder and create a subfolder **web**

Create a custom **index.html** file in the web subfolder with the following content

**<h1>My First Image</h1>**

Return to the **C:\image1** folder and create a **Dockerfile** file with the following content

**FROM mcr.microsoft.com/windows/servercore/iis**

**RUN powershell -NoProfile -Command Remove-Item -Recurse C:\inetpub\wwwroot\\***

**WORKDIR /inetpub/wwwroot**

**COPY web/ .**

Save and close the file

Being in the **C:\image1** folder execute the following command to create the custom image

**docker build -t custom-iis:v1 .**

Once done, check the images list

**docker image ls**

And test the new image by running a container out of it

**docker run -d -p 8080:80 --name web1 custom-iis:v1**

Open a browser tab on the container host and navigate to <http://localhost:8080/> to see the result

Remove the running container with

**docker container rm --force web1**

### Create own image

Now, we will start with a more generic image and will add not just our custom files, but also Windows features

Create a folder on the container host

**mkdir C:\image2**

Navigate to the folder and create a subfolder **web**

Create a custom **index.html** file in the web subfolder with the following content

**<h1>My Second Image</h1>**

Return to the **C:\image2** folder and create a **Dockerfile** file with the following content

**# escape=`**

**FROM mcr.microsoft.com/windows/servercore:1809**

**LABEL maintainer="student@softuni.bg"**

**RUN powershell -Command `**

**Add-WindowsFeature Web-Server; `**

**Invoke-WebRequest -UseBasicParsing -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.10/ServiceMonitor.exe' -OutFile 'C:\ServiceMonitor.exe'**

**RUN sc config w3svc start=demand**

**WORKDIR /inetpub/wwwroot**

**COPY web/ .**

**EXPOSE 80**

**ENTRYPOINT ["C:\\ServiceMonitor.exe", "w3svc"]**

Save and close the file

Being in the **C:\image2** folder execute the following command to create the custom image

**docker build -t custom-iis:v2 .**

Once done, check the images list

**docker image ls**

And test the new image by running a container out of it

**docker run -d -p 8080:80 --name web1 custom-iis:v2**

Open a browser tab on the container host and navigate to <http://localhost:8080/> to see the result

Remove the running container with

**docker container rm --force web1**

*Of course, the recommended way of creating own images based on IIS is to use one of the officially provided images*

## Clean up containers and images

We can stop and then remove running containers or directly force their removal no matter if they are running or not

**docker container rm <container-id-or-name> --force**

After we remove the containers, we can remove the images as well

**docker image rm <image-id>:<tag>**

There is a **prune** sub-command, available in **container**, **image**, and **volume** categories

When used, it removes all unused and unnecessary artefacts of the corresponding type