NAV Docker Container Image

Contents

[What are Containers? What is Docker? 4](#_Toc492285036)

[Get started – prepare your environment 4](#_Toc492285037)

[Windows Server 2016 with Containers on Azure 4](#_Toc492285038)

[Windows Server 2016 4](#_Toc492285039)

[Windows 10 5](#_Toc492285040)

[Get started – run your first NAV docker container 5](#_Toc492285041)

[NAV Docker image tags 5](#_Toc492285042)

[dynamics-nav-generic 6](#_Toc492285043)

[dynamics-nav 6](#_Toc492285044)

[Scenarios 7](#_Toc492285045)

[Skip self-signed certificates for local docker containers 7](#_Toc492285046)

[Specify username and password for your NAV SUPER user 7](#_Toc492285047)

[Use Windows Authentication for NAV 7](#_Toc492285048)

[With the current Windows AD user on the host computer 8](#_Toc492285049)

[Setup gMSA with the Domain of the host computer 8](#_Toc492285050)

[Publishing ports on the host and specifying a hostname using NAT network settings 8](#_Toc492285051)

[Adding ClickOnce deployment of the Windows Client 9](#_Toc492285052)

[Use a certificate, issued by a trusted authority 9](#_Toc492285053)

[Avoid that the container exits on error during startup 10](#_Toc492285054)

[Specify your own license file to use in the container 10](#_Toc492285055)

[Specify a database which already contains a license file 11](#_Toc492285056)

[Specify a secure Url to a license file 11](#_Toc492285057)

[Specify the path of a license file 11](#_Toc492285058)

[Override the SetupLicense script 11](#_Toc492285059)

[Suppress deployment of the WebClient and/or Http site when running a container 12](#_Toc492285060)

[Publish an extension to NAV when the container is started 12](#_Toc492285061)

[Import a .fob file when a container is started 13](#_Toc492285062)

[Import and compile a .txt file with objects when a container is started 14](#_Toc492285063)

[Specify your own Database backup file to use with a NAV Container 15](#_Toc492285064)

[Place the Database file in a file share on the host computer 15](#_Toc492285065)

[Connect a NAV Container to another Database server 16](#_Toc492285066)

[Connect a NAV Container to a SQL Server using Database authentication or an existing Azure SQL database 16](#_Toc492285067)

[Connect to the NAV Container and develop using the classic development environment (C/SIDE) 18](#_Toc492285068)

[Connect to the NAV Container and develop using Visual Studio Code 19](#_Toc492285069)

[Create your own Image based on a specific NAV image 19](#_Toc492285070)

[Create your own Image based on the generic NAV image 20](#_Toc492285071)

[Appendix 1 - Some good practices I learned when using Docker 22](#_Toc492285072)

[When developing and testing scripts 22](#_Toc492285073)

[Appendix 2 – Scripts 23](#_Toc492285074)

[SetupVariables.ps1 24](#_Toc492285075)

[SetupDatabase.ps1 24](#_Toc492285076)

[SetupCertificate.ps1 25](#_Toc492285077)

[SetupConfiguration.ps1 25](#_Toc492285078)

[SetupAddIns.ps1 26](#_Toc492285079)

[SetupLicense.ps1 27](#_Toc492285080)

[SetupWebConfiguration.ps1 27](#_Toc492285081)

[SetupClickOnce.ps1 27](#_Toc492285082)

[SetupClickOnceDirectory.ps1 28](#_Toc492285083)

[SetupFileShare.ps1 28](#_Toc492285084)

[SetupSqlUsers.ps1 29](#_Toc492285085)

[SetupNavUsers.ps1 29](#_Toc492285086)

[AdditionalSetup.ps1 30](#_Toc492285087)

[AdditionalOutput.ps1 30](#_Toc492285088)

[MainLoop.ps1 31](#_Toc492285089)

# What are Containers? What is Docker?

If you are new to Docker and Containers, please read this document:

<https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/>

it describes very well what Containers are and what Docker is.

If you want more info, there are a lot of Channel9 videos on Containers as well.

<https://channel9.msdn.com/Search?term=containers#ch9Search&lang-en=en&pubDate=year>

If you have problems with Docker (not NAV related), the Windows Containers Docker forum is the place you can ask questions (read the readme first):

<https://social.msdn.microsoft.com/Forums/en-US/home?forum=windowscontainers>

# Get started – prepare your environment

Docker only runs on Windows Server 2016 (or later) or Windows 10.

When using Windows 10, Docker always uses Hyper-V isolation with a very thin layer. When using Windows Server 2016, you can choose between Hyper-V isolation or process isolation. Read more about this here: <https://docs.microsoft.com/en-us/virtualization/windowscontainers/about/> (same link as above)

I will describe 3 ways to get started with Containers. If you have a laptop/machine running Windows Server 2016 or Windows 10 – you can use this one. If not, you can deploy a Windows Server 2016 with Containers on Azure, which will give you everything to get started.

## Windows Server 2016 with Containers on Azure

In the Azure Gallery, you will find an image with Windows Server 2016 and Docker installed and pre-configured. You can deploy this image by clicking this link:

<https://portal.azure.com/#create/Microsoft.Template/uri/https%3A%2F%2Fraw.githubusercontent.com%2FMicrosoft%2FVirtualization-Documentation%2Flive%2Fwindows-server-container-tools%2Fcontainers-azure-template%2Fazuredeploy.json>

**Note**, do not select Standard\_D1 (simply not powerful enough) – use Standard\_D2 or Standard\_D3.

In this VM, you can now run all the docker commands, described in this document.

## Windows Server 2016

Follow the steps in this document to install Docker on a machine with Windows Server 2016:

<https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-server>

## Windows 10

Follow the steps in this document to install Docker on Windows 10:

<https://docs.microsoft.com/en-us/virtualization/windowscontainers/quick-start/quick-start-windows-10>

# Get started – run your first NAV docker container

On your machine with Docker, open a command prompt and type this command (please obtain username and password from Microsoft):

docker login navdocker.azurecr.io -u <username> -p <password>

This will ensure that you have access to a private docker registry called **navdocker.azurecr.io**, and can pull images from this registry.

Now run this command:

docker run -e ACCEPT\_EULA=Y navdocker.azurecr.io/dynamics-nav:2017

**Note**, if you are running **Windows 10**, you will have to add **-m 3G** as an extra parameter to the docker run command above (and in all docker run commands in this doc.)

**Windows 10:**

docker run -m 3G -e ACCEPT\_EULA=Y navdocker.azurecr.io/dynamics-nav:2017

You will see that docker downloads a number of layers and once the download and extraction process is complete, the NAV Container will start.

**Note**, the download and extraction process might take some time depending on your bandwidth and the performance of the docker host computer.

# NAV Docker image tags

The NAV Docker images currently resides in a private registry called navdocker.azurecr.io. This is a temporary registry and the images will eventually be on the docker hub under microsoft (like all other microsoft docker images) if we decide to publish NAV Docker images.

In this registry you will find 2 categories of images:

* dynamics-nav-generic – generic image without any NAV build, but can be used together with any NAV DVD (NAV 2016 and up) to launch a docker container with that version of NAV.
* dynamics-nav – specific images with a version of NAV pre-installed and pre-configured, ready to configure and run.

The generic image is used as a base for all specific images.

The way, the image is architected, you should not need to build your own image, you can use and run the images as they are. If you for some reason need to, you can also build your own images based on the generic or specific images.

## dynamics-nav-generic

All generic images have one tag, which consists of the date and time when the image was built. Furthermore, the latest generic image has the latest tag stamped on it. You should always use the latest generic image.

## dynamics-nav

All specific images are tagged with the version number of NAV, which is installed. The following list of examples explains the tagging strategy:

* **navdocker.azurecr.io/dynamics-nav:2017** will give you the latest NAV 2017 W1 version.
* **navdocker.azurecr.io/dynamics-nav:2017-cu8** will give you NAV 2017 CU8 W1 version.
* **navdocker.azurecr.io/dynamics-nav:2017-dk** will give you the latest NAV 2017 DK version.
* **navdocker.azurecr.io/dynamics-nav:2017-cu8-dk** will give you NAV 2017 CU8 DK version.
* **navdocker.azurecr.io/dynamics-nav:10.0.17501.0** will give you a specific build of NAV (in this case, NAV 2017 CU8 W1).
* **navdocker.azurecr.io/dynamics-nav:10.0.17501.0-dk** will give you a specific DK build of NAV (in this case, NAV 2017 CU8 DK).

There is no such thing as **dynamics-nav:latest** at this time, instead you can get the latest NAV 2016, the latest NAV 2017 etc.

For this test period, the **navdocker.azurecr.io** registry contains the following images:

* NAV 2017 all CU’s, all languages
* NAV 2016 all CU’s, all languages
* devpreview july update (us, gb)
* devpreview august update (us, ca, gb)

If you are wondering about the tagging of devpreview, it really follows the tagging examples above:

* navdocker.azurecr.io/dynamics-nav:devpreview to get the latest devpreview version with W1
* navdocker.azurecr.io/dynamics-nav:devpreview-us to get the latest devpreview version with US database
* navdocker.azurecr.io/dynamics-nav:devpreview-july to get the july update

Note that image names and tags are case sensitive – everything must be specified in lower case.

# Scenarios

In the following, I will go through a number of scenarios, you might find useful when launching a docker container. Most of the scenarios can be combined, but in some cases, it doesn’t make sense to combine them.

## Skip self-signed certificates for local docker containers

The parameter you need to specify to setup the NAV Container without SSL is:

-e UseSSL=N

The default for **UseSSL** is **Y** when using NavUserPassword authentication and **N** when using Windows authentication.

Example:

docker run -e ACCEPT\_EULA=Y -e UseSSL=N navdocker.azurecr.io/dynamics-nav:2017

**Note**, if you are planning to expose your container outside the boundaries of your own machine, you should not run without SSL.

## Specify username and password for your NAV SUPER user

The parameters needed to specify username and password for your NAV SUPER user are:

-e username=username -e password=password

Example:

docker run -e ACCEPT\_EULA=Y -e username=admin -e password=P@ssword1 navdocker.azurecr.io/dynamics-nav:2017

If you do **NOT** specify a username and a password, the NAV Docker Image will create a user called **admin** with a **random password**. This password is shown in the output of the Docker Container:

NAV Admin Username: admin  
NAV Admin Password: Fewe8407

Please remember to write it down.

## Use Windows Authentication for NAV

The parameters used to specify that you want to use Windows Authentication are:

-e auth=Windows -e username=username -e password=password

A container doesn’t have its own Active Directory, but you can still setup Windows Authentication.

### With the current Windows AD user on the host computer

This is done by specifying the credentials of your Windows AD user (without the domain name) and our Windows AD password.

**Note**, that in this mode, you will be able to locate your Windows AD password in clear text inside the Container and in the caption of the Container Window (if you don’t close it), so this should only be used when running a container on your local computer for development or demo purposes.

**Note also**, if your docker image is publicly available for docker inspect, then you will also see you Windows AD credentials right there… - please use with caution…

Example:

docker run -e ACCEPT\_EULA=Y -e auth=Windows -e username=freddyk -e password=P@ssword1 navdocker.azurecr.io/dynamics-nav:2017

### Setup gMSA with the Domain of the host computer

This is done by setting up group managed service accounts in your AD and then specifying a domain user (with the domain name). In this mode you do not specify the password of the domain user.

**Note**, you have to be a domain admin to setup gMSA.

Example:

docker run -e ACCEPT\_EULA=Y -e auth=Windows -e username=europe\freddyk navdocker.azurecr.io/dynamics-nav:2017

We strongly recommend to use gMSA if you are using Windows Authentication.

## Publishing ports on the host and specifying a hostname using NAT network settings

**Note** that network settings on Docker can be setup in a lot of different ways. Please consult the Docker documentation or this blog post:

<https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-networking>

to learn more about container networking. When installing Docker, by default it creates a NAT network. This scenario explains how to publish ports using NAT network settings only. Publishing ports enables you to access the Container from outside the host computer. The parameters used for publishing ports on the host and specifying a hostname are not specific to the NAV container image, but are generic Docker parameters:

-p <PortOnHost>:<PortInDocker> -h hostname

In order for a port to be published on the host, the port needs to be exposed in the container. By default, the NAV container image exposes the following ports:

8080 file share

80 http

443 https

1433 sql

7045 management

7046 client

7047 soap

7048 odata

7049 development

If you want to publish all exposed ports on the host, you can use: --publish-all or -P (capital P).

**Note**, publishing port 1433 on an internet host might cause your computer to be vulnerable for attacks.

Example:

docker run -h dockertest.navdemo.net -e ACCEPT\_EULA=Y -p 8080:8080 -p 80:80 -p 443:443 -p 7045-7049:7045-7049 navdocker.azurecr.io/dynamics-nav:2017

In this example, dockertest.navdemo.net is a DNS name, which points to the IP address of the host computer (A or CNAME record) and the ports 8080, 80, 443, 7045, 7046, 7047, 7048 and 7049 are all bound to the host computer, meaning that I can navigate to <http://dockertest.navdemo.net:8080> to download files from the NAV container file share.

## Adding ClickOnce deployment of the Windows Client

The parameter needed to specify that you want to have use the RTC Client via ClickOnce is:

-e ClickOnce=Y

Example:

docker run -e ACCEPT\_EULA=Y -e ClickOnce=Y navdocker.azurecr.io/dynamics-nav:2017

In the output of the docker command, you will find a line, specifying the URL for downloading the ClickOnce manifest, like:

ClickOnce Manifest: http://dockertest.navdemo.net:8080/NAV

Launch this URL in a browser, download and start the Windows Client.

## Use a certificate, issued by a trusted authority

There are no parameters in which you can specify a certificate directly. Instead, you will have to override the SetupCertificate script in the Docker image. Overriding scripts is done by placing a script in a folder on the host computer and sharing this folder to the NAV Container as a folder called c:\run\my. The parameter used to achieve this is:

-v c:\myfolder:c:\run\my

When the NAV Container starts, it will look for scripts in the c:\run\my folder to override scripts, which are placed in c:\run.

You should place your certificate pfx file in c:\myfolder together with this script:

$certificatePfxFile = Join-Path $PSScriptRoot "<Certificate Pfx Filename>"

$certificatePfxPassword = "<Certificate Pfx Password>"

$dnsidentity = "<Dns Identity>"

$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2($certificatePfxFile, $certificatePfxPassword)

$certificateThumbprint = $cert.Thumbprint

Write-Host "Certificate File Thumbprint $certificateThumbprint"

if (!(Get-Item Cert:\LocalMachine\my\$certificateThumbprint -ErrorAction SilentlyContinue)) {

Write-Host "Import Certificate to LocalMachine\my"

Import-PfxCertificate -FilePath $certificatePfxFile -CertStoreLocation cert:\localMachine\my -Password (ConvertTo-SecureString -String $certificatePfxPassword -AsPlainText -Force) | Out-Null

}

If the certificate you use isn’t issued by an authority, which is in the Trusted Root Certification Authorities, then you will have to import the pfx file to LocalMachine\root as well as LocalMachine\my, using this line:

Import-PfxCertificate -FilePath $certificatePfxFile -CertStoreLocation cert:\localMachine\root -Password (ConvertTo-SecureString -String $certificatePfxPassword -AsPlainText -Force) | Out-Null

And then use Docker run with the -v parameter explained above.

Example:

docker run -v c:\myfolder:c:\run\my -h dockertest.navdemo.net -e ACCEPT\_EULA=Y -p 8080:8080 -p 80:80 -p 443:443 -p 7045-7049:7045-7049 navdocker.azurecr.io/dynamics-nav:2017

**Note**, for this to work, dockertest.navdemo.net needs to point to the host computer, and the certificate needs to be a \*.navdemo.net certificate or a dockertest.navdemo.net certificate.

## Avoid that the container exits on error during startup

If you are working on extending the NAV on Docker scripts, you might see that the Container exists immediately if an error occurs during the startup scripts. This is because the Container never reaches the main loop. If you specify the parameter

-e ExitOnError=N

The startup scripts will run and print out the error as normally, but it will still enter the main loop and allow you to connect to the Container to do troubleshooting.

## Specify your own license file to use in the container

You can specify your own license file to use in the Container in a few different ways.

### Specify a database which already contains a license file

If you are connecting to a foreign database server or you have specified a database backup which already contains a license file, then there is no need to specify a lciens

### Specify a secure Url to a license file

If you have a secure Url from which you can download your license file, you can specify this Url as a parameter

-e LicenseFile=<secure license file Url>

The secure license file Url needs to start with http or https, in which case, the script will proceed to download the license file and import it into the NAV Database.

### Specify the path of a license file

If the license file is accessible from the container through a path, you can specify this path as a parameter

-e LicenseFile=<path to license file>

Example:

Place you license file in a folder called c:\myfolder on the Docker host and use this parameter:

-v c:\myfolder:c:\run\my -e LicenseFile=c:\run\my\mylicense.flf

The -v parameter shares the myfolder from the host to the container as c:\run\my and then mylicense.flf is accessible from the container.

### Override the SetupLicense script

You can also choose to override the script, which imports the license file and take all in your own hands.

The SetupLicense script is executed as the first thing after the Service Tier has started and it is invoked always. This means that you need to determine whether or not you need to install a license file. If you place a file called SetupLicense.ps1 in a folder, which gets shared into c:\run\my, then that script will be executed instead of the default script.

Please inspect the file c:\run\SetupLicense.ps1 in the container for the default behavior and see Appendix 1 – Scripts for information about how to override scripts.

Example:

if ($restartingInstance) {

# Nothing to do

} else {

Import-NAVServerLicense -LicenseFile ”c:\run\my\cronus.flf" -ServerInstance 'NAV' -Database NavDatabase -WarningAction SilentlyContinue

}

## Suppress deployment of the WebClient and/or Http site when running a container

If you need to run a container in order to perform a task (like running tests, reports or other things), there might not be any reason to deploy the Web Client (takes ~30 seconds). To suppress deployment of the Web Client, specify the parameter:

-e WebClient=N

If you suppress WebClient, you might also want to suppress the deployment of the http site for file downloads. This can be done with the parameter:

-e httpsite=N

Suppressing both these sites, will cause the container to never start the IIS (no reason to do so).

Example:

docker run -e WebClient=N -e httpsite=N -e ACCEPT\_EULA=Y navdocker.azurecr.io/dynamics-nav:2017

## Publish an extension to NAV when the container is started

There are no parameters for which you can specify extensions you want to publish, instead you will need to override a script to do this. The AdditionalSetup script is designed for a scenario like this. When the AdditionalSetup script is executed, all configurations are done, the NAV Service Tier has been started and you can do what you need.

Example:

Create a folder on the Docker host called c:\myfolder and copy the extension to the folder.

Create a script called AdditionalSetup.ps1 with this content:

Import-Module 'C:\Program Files\Microsoft Dynamics NAV\\*\Service\Microsoft.Dynamics.Nav.Apps.Management.psd1'

$extension = "myextension.navx"

Write-Host "Publishing $extension"

Publish-NAVApp -ServerInstance NAV -Path (Join-Path $PSScriptRoot $extension) -SkipVerification

Write-Host "Installing $extension"

Install-NAVApp -ServerInstance NAV -Path (Join-Path $PSScriptRoot $extension)

Where myextension.navx is the filename of your extension and run your NAV on Docker using:

docker run -v c:\myfolder:c:\run\my -e ACCEPT\_EULA=Y -e UseSSL=N navdocker.azurecr.io/dynamics-nav:2017

This would start up the latest NAV 2017 W1, publish and install the myextension.navx. The output from the Container shows this as well:

Start NAV Service Tier

Using existing license file

Create NAV Web Server Instance

Create http download site

Publishing myextension.navx

Installing myextension.navx

Container IP Address: 172.25.19.201

Container Hostname : navserver

Web Client : http://navserver/NAV/WebClient/

Files:

Ready for connections!

## Import a .fob file when a container is started

There are no parameters for which you can specify a .fob file you want to import, instead you will need to override a script to do this. The AdditionalSetup script is designed for a scenario like this. When the AdditionalSetup script is executed, all configurations are done, the NAV Service Tier has been started and you can do what you need.

Example:

Create a folder on the Docker host called c:\myfolder and copy the .fob file to the folder.

Create a script called AdditionalSetup.ps1 with this content:

Import-Module "C:\Program Files (x86)\Microsoft Dynamics NAV\\*\RoleTailored Client\Microsoft.Dynamics.Nav.Ide.psm1" -wa SilentlyContinue

$objects = "myobjects.fob"

Write-Host "Import $objects"

Import-NAVApplicationObject -DatabaseServer "localhost" `

-DatabaseName $databaseName `

-Path (Join-Path $PSScriptRoot $objects) `

-ImportAction Overwrite `

-SynchronizeSchemaChanges Force `

-NavServerName "localhost" `

-NavServerInstance "NAV" `

-confirm:$false

Where myobjects.fob is the filename of your objects and run your NAV on Docker using:

docker run -v c:\myfolder:c:\run\my -e ACCEPT\_EULA=Y -e UseSSL=N navdocker.azurecr.io/dynamics-nav:2017

This would start up the latest NAV 2017 W1, publish and import objects from myobjects.fob. The output from the Container shows this as well:

Start NAV Service Tier

Use existing license file

Create NAV Web Server Instance

Create http download site

Import myobjects.fob

Container IP Address: 172.25.17.97

Container Hostname : navserver

Web Client : http://navserver/NAV/WebClient/

Files:

Ready for connections!

## Import and compile a .txt file with objects when a container is started

There are no parameters for which you can specify a .txt file you want to import, instead you will need to override a script to do this. The AdditionalSetup script is designed for a scenario like this. When the AdditionalSetup script is executed, all configurations are done, the NAV Service Tier has been started and you can do what you need.

Note that you will have to specify a developer license to the docker image in order to import and compile .txt files. In this example, I will place the license file in the shared folder and specify the name of the license file to the licensefile parameter.

Example:

Create a folder on the Docker host called c:\myfolder and copy the .txt file and your developer license file to the folder.

Create a script called AdditionalSetup.ps1 with this content:

Import-Module "C:\Program Files (x86)\Microsoft Dynamics NAV\\*\RoleTailored Client\Microsoft.Dynamics.Nav.Ide.psm1" -wa SilentlyContinue

$objects = "myobjects.txt"

Write-Host "Import $objects"

Import-NAVApplicationObject -DatabaseServer "localhost" `

-DatabaseName $databaseName `

-Path (Join-Path $PSScriptRoot $objects) `

-ImportAction Overwrite `

-SynchronizeSchemaChanges Force `

-NavServerName "localhost" `

-NavServerInstance "NAV" `

-confirm:$false

Write-Host "Compile uncompiled objects"

Compile-NAVApplicationObject -DatabaseServer "localhost" `

-DatabaseName "CRONUS" `

-Filter "Compiled=No" `

-SynchronizeSchemaChanges Force `

-NavServerName "localhost" `

-NavServerInstance "NAV"

Where myobjects.txt is the filename of your objects and run your NAV on Docker using:

docker run -v c:\myfolder:c:\run\my -e ACCEPT\_EULA=Y -e UseSSL=N -e licensefile=”c:\run\my\mylicense.flf” navdocker.azurecr.io/dynamics-nav:2017

Where mylicense.flf is the filename of your developer license file.

This would start up the latest NAV 2017 W1, publish and import objects from myobjects.txt and compile them. The output from the Container shows this as well:

Start NAV Service Tier

Using license file 'c:\run\my\mylicense.flf'

Import NAV License

Create NAV Web Server Instance

Create http download site

Import myobjects.txt

Compile uncompiled objects

Container IP Address: 172.25.29.41

Container Hostname : navserver

Web Client : http://navserver/NAV/WebClient/

Files:

Ready for connections!

## Specify your own Database backup file to use with a NAV Container

If you have a database backup file (.bak), you can specify that TODO

## Place the Database file in a file share on the host computer

The database files are placed inside the container by default. If you want to copy the database to a share on the Docker host, you can override the SetupDatabase.ps1 script by creating a file called SetupDatabase.ps1 in c:\myfolder with this content:

if (!$restartingInstance) {

$mdfName = Join-Path $PSScriptRoot "$DatabaseName.mdf"

$ldfName = Join-Path $PSScriptRoot "$DatabaseName.ldf"

$filesExists = (Test-Path $mdfName -PathType Leaf) -and (Test-Path $ldfName -PathType Leaf);

Write-Host "Take database [$DatabaseName] offline"

Invoke-SqlCmd -Query "ALTER DATABASE [$DatabaseName] SET OFFLINE WITH ROLLBACK IMMEDIATE"

if ($filesExists) {

Write-Host "Database files for [$DatabaseName] already exists"

} else {

Write-Host "Move database files for [$DatabaseName]"

(Invoke-SqlCmd -Query "SELECT Physical\_Name as filename FROM sys.master\_files WHERE DB\_NAME(database\_id) = '$DatabaseName'").filename | ForEach-Object {

$FileInfo = Get-Item -Path $\_

$DestinationFile = "{0}\{1}{2}" -f $PSScriptRoot, $DatabaseName, $FileInfo.Extension

if (($DestinationFile -ne $mdfName) -and ($destinationFile -ne $ldfName)) { throw "Unexpected filename: $DestinationFile" }

Copy-Item -Path $FileInfo.FullName -Destination $DestinationFile -Force

}

}

Write-Host "Drop database [$DatabaseName]"

Invoke-SqlCmd -Query "DROP DATABASE [$DatabaseName]"

$Files = "(FILENAME = N'$mdfName'), (FILENAME = N'$ldfName')"

Write-Host "Attach files as new Database [$DatabaseName]"

Invoke-SqlCmd -Query "CREATE DATABASE [$DatabaseName] ON (FILENAME = N'$mdfName'), (FILENAME = N'$ldfName') FOR ATTACH"

}

and then start a container with a command like this:

docker run -v c:\myfolder:c:\run\my -e ACCEPT\_EULA=Y -e UseSSL=N navdocker.azurecr.io/dynamics-nav:2017

Then you will see that database files will be copied to c:\myfolder unless they already exist. If the files already exists, they will be attached to the SQL Server in the Container. You can of course modify the script to fit your needs.

## Connect a NAV Container to another Database server

If you want to Connect NAV to another Database, which resides on a local SQL Server, you can do so by specifying DatabaseServer, DatabaseInstance and DatabaseName as parameters to Docker Run. This requires Windows Authentication (gMSA) to the SQL Server and needs to have setup SPN’s etc.

Example:

docker run --name navserver -h navserver -e accept\_eula=Y -e usessl=N -e DatabaseServer=sqlserver -e DatabaseName=CRONUS navdocker.azurecr.io/dynamics-nav:2017

Note, that NAV doesn’t have to be setup for Windows Authentication in order for the Docker Container to be able to connect to a SQL Server using Windows Authentication. You will however have to setup gMSA, which in effect will map the user running the Service Tier to a user in the AD hosting the container.

## Connect a NAV Container to a SQL Server using Database authentication or an existing Azure SQL database

If you want to Connect NAV to another Database, you can do so by specifying DatabaseServer, DatabaseInstance and DatabaseName as parameters to Docker Run. This requires Windows Authentication to the SQL Server and needs to have setup SPN’s etc.

If however you want to connect to another Database, which uses Database Authentication (like Azure SQL), then you need to override the SetupDatabase script with a script like this:

if (!$RestartingInstance) {

Write-Host "Change Database Connection"

$DatabaseServer = "<databaseserver>"

$DatabaseInstance = "<databaseinstance>"

$DatabaseName = "<databasename>"

$DatabaseUserName = "<databaseusername>"

$DatabasePassword = "<databasepassword>"

$EncryptionPassword = "<encryptionpassword>"

$TrustSQLServerCertificate = $false

$DatabaseSecurePassword = ConvertTo-SecureString -String $DatabasePassword -AsPlainText -Force

$DatabaseCredentials = New-Object PSCredential -ArgumentList $DatabaseUserName, $DatabaseSecurePassword

$EncryptionKeyPath = Join-Path $ServiceTierFolder 'DynamicsNAV.key'

$EncryptionSecurePassword = ConvertTo-SecureString -String $EncryptionPassword -AsPlainText -Force

New-NAVEncryptionKey -KeyPath $EncryptionKeyPath -Password $EncryptionSecurePassword -Force | Out-Null

Write-Host "Import Encryption Key"

Import-NAVEncryptionKey -ServerInstance NAV `

-ApplicationDatabaseServer $DatabaseServer `

-ApplicationDatabaseCredentials $DatabaseCredentials `

-ApplicationDatabaseName $DatabaseName `

-KeyPath $EncryptionKeyPath `

-Password $EncryptionSecurePassword `

-WarningAction SilentlyContinue `

-Force

Set-NAVServerConfiguration -ServerInstance "NAV" -KeyName "EnableSqlConnectionEncryption" -KeyValue "true" -WarningAction SilentlyContinue

Set-NAVServerConfiguration -ServerInstance "NAV" -KeyName "TrustSQLServerCertificate" -KeyValue $TrustSQLServerCertificate.Tostring().ToLowerInvariant() -WarningAction SilentlyContinue

Set-NavServerConfiguration -serverinstance "NAV" -databaseCredentials $DatabaseCredentials

}

Where *DatabaseServer*, *DatabaseInstance*, *DatabaseName*, *DatabaseUsername* and *DatabasePassword* are kind of self-explanatory.

*EncryptionPassword* is the password for the encryption key. This needs to be the same for all NAV Service Tiers connecting to the same database.

*TrustSQLServerCertificate* specifies whether you automatically trust the Certificate from the SQL Server, even if it isn’t trusted by a trusted authority.

Example:

In one command prompt run this command:

docker run --name navserver -h navserver -e accept\_eula=Y -e usessl=N -e username="admin" -e password="P@ssword1" navdocker.azurecr.io/dynamics-nav:2017

Place the above script in a folder called c:\myfolder, remove the databaseServer, databaseInstance and databaseName variables from the script and modify the remaining database settings in the script to:

$DatabaseUserName = "sa"

$DatabasePassword = "P@ssword1"

$EncryptionPassword = "1234abcd!1234abcd"

$trustSQLServerCertificate = $true

Then open a 2nd command prompt and run this command:

docker run --name navserver1 -h navserver1 -e accept\_eula=Y -e usessl=N -v C:\myfolder:c:\run\my -e username="admin" -e password="P@ssword2" -e DatabaseServer=navserver -e DatabaseName=CRONUS navdocker.azurecr.io/dynamics-nav:2017

Now, if you connect to http://navserver1/NAV/WebClient , you will have to login with admin and P@ssword1, because the 2nd docker container connects and uses the database server in the 1st container.

As the matter of fact, you will see that the 2nd container never starts the SQL Server instance because the DatabaseServer parameter is not localhost.

This functionality might be moved to the generic docker image later.

## Connect to the NAV Container and develop using the classic development environment (C/SIDE)

It is recommended that you use Windows Authentication when running the classic development environment. Most things will work with Username/Password authentication, but some things will not, like changing tables and synchronizing schema changes.

If you do not use Windows Authentication, you will have to connect to the database using Database authentication and use the username SA and the password specified or created to the default NAV user.

There are a few ways to run the classic Development Environment, I will explain two:

#### Use Click-Once

When starting the NAV Container, you can specify parameter

-e ClickOnce=Y

Which will cause the container output to include a line called ClickOnce Manifest URL.

Container Hostname : navserver

Web Client : http://navserver/NAV/WebClient/

Dev. Server : http://navserver

Dev. ServerInstance : NAV

ClickOnce Manifest : http://navserver:8080/NAV

Files:

http://navserver:8080/al-0.8.12437.vsix

Ready for connections!

Opening the ClickOnce Manifest URL in a browser allows you to install pre-requisites and both the Windows Client and C/SIDE development environment.

#### Copy necessary files to a Shared Folder

Override the AdditionalSetup.ps1 script by sharing a folder to the Container in the C:\Run\my folder. Add this code to the AdditionalSetup.ps1:

if (!(Test-Path "$myPath\RoleTailored Client" -PathType Container)) {

Write-Host "Copy RoleTailoted Client files"

Copy-Item -path $roleTailoredClientFolder -destination $myPath -force -Recurse -ErrorAction Ignore

}

Then you can run C/SIDE directly from *c:\myfolder\RoleTailored Client\finsql.exe*.

If you also want to run the Windows Client directly from the copied folder, you will need to add a ClientUserSettings.config file. This can be done by adding these lines to the AdditionalSetup.ps1 script:

$ClientUserSettingsFileName = "$runPath\ClientUserSettings.config"

[xml]$ClientUserSettings = Get-Content $clientUserSettingsFileName

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='Server']").value = "$hostname"

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='ServerInstance']").value="NAV"

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='ServicesCertificateValidationEnabled']").value="false"

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='ClientServicesPort']").value="$publicWinClientPort"

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='ACSUri']").value = ""

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='DnsIdentity']").value = "$dnsIdentity"

$clientUserSettings.SelectSingleNode("//configuration/appSettings/add[@key='ClientServicesCredentialType']").value = "$Auth"

$clientUserSettings.Save("$destFolder\RoleTailored Client\ClientUserSettings.config")

After this you can run the Windows Client directly from *c:\myfolder\RoleTailored Client\Microsoft.Dynamics.Nav.Client.exe*.

## Connect to the NAV Container and develop using Visual Studio Code

Visual Studio Code can NOT connect to NAV 2017 or earlier versions – you need to use a NAV Container post NAV 2017, f.ex. the NAV Developer Preview.

When you deploy a version of NAV, which can be used with Visual Studio Code, you will see two lines with Dev. Server and the AL Language extension for Visual Studio (a .vsix file) in the Files area:

Container Hostname : navserver

Web Client : http://navserver/NAV/WebClient/

Dev. Server : http://navserver

Dev. ServerInstance : NAV

Files:

http://navserver:8080/al-0.8.12437.vsix

Ready for connections!

The name of .vsix file reveals the version of the AL Language extension needed by this server. If you do not have this version already installed in Visual Studio Code, then copy the URL for the .vsix file and download the file. Start Visual Studio Code, uninstall prior versions of the AL Language extension (.vsix) and install the newly downloaded version.

Press Ctrl+Shift+P, select AL: Go!

Select Local Server and modify the Server and the ServerInstance settings in launch.json to match the Dev. Server and Dev. ServerInstance info from the container.

In launch.json, also change the authentication setting to the setting used by the Container: Windows or UserPassword.

In app.json modify the application locale if necessary and you should now be able to download symbols and press F5 to deploy the test app.

## Create your own Image based on a specific NAV image

Typically, you shouldn’t need to build your own image based on a specific NAV image. Most things can be done by overriding scripts and running the specific Docker image directly. The disadvantage of building your own images is, that you will have to build a new image every time there is a new version of the specific image.

If you despite this, want to build an image based on a specific image, you should follow the same process, as we do, when we build the country versions of NAV.

* **navdocker.azurecr.io/dynamics-nav:2017-cu9-dk** is a layer built on top of
* **navdocker.azurecr.io /dynamics-nav:2017-cu9** which is a layer built on top of
* **navdocker.azurecr.io /dynamics-nav-generic** which is a layer built on top of
* **microsoft/sql-server-developer** which is a layer built on top of
* **microsoft/windowsservercore**

When building a country version of NAV we use a DOCKERFILE that looks like this:

FROM navdocker.azurecr.io/dynamics-nav:2017-cu9

ENV COUNTRYURL ”<URL to COUNTRY specific zipfile>”

COPY ./importCountry.ps1 C:/RUN/importCountry.ps1

ENV DatabaseName CronusDK

RUN PowerShell .\Run\importCountry.ps1

The ImportCountry script will download the database backup for DK and the local installers folder. Then it will restore the Danish database, modify customsettings.config, run the installers, remove the W1 CRONUS database and cleanup to avoid that the downloaded artefacts takes up space in the image.

Note, that navstart.ps1 is NOT run during this, it has already been run as part of building the dynamics-nav:2017-cu9 (with buildingimage set to true), NAV is installed and ready to run, so adding another layer should only replace/install the things that are needed.

You should do the same if you are basing your image on a specific image, make the layer as slim as possible.

## Create your own Image based on the generic NAV image

Typically, you shouldn’t need to build your own image at all, but if you want to build an Image based on the Generic NAV Docker Image, you should follow the same process as we do when we create specific W1 images.

When building a specific W1 image we use a DOCKERFILE that looks like this:

FROM navdocker.azurecr.io/dynamics-nav-generic

ENV VSIXURL “<URL to Vsix file>”

ENV NAVDVDURL ”<URL to NAVDVD zipfile>”

RUN PowerShell .\Run\buildimage.ps1

ENV DatabaseServer localhost

ENV DatabaseName CRONUS

The Buildimage.ps1 script will download the artefacts, set $buildingImage to true and run c:\run\navstart.ps1.

The PowerShell code for downloading looks like this:

if ("$env:NAVDVDURL" -ne "") {

(New-Object System.Net.WebClient).DownloadFile("$env:NAVDVDURL", "C:\NAVDVD.zip")

[Reflection.Assembly]::LoadWithPartialName("System.IO.Compression.Filesystem") | Out-Null

[System.IO.Compression.ZipFile]::ExtractToDirectory("C:\NAVDVD.zip","C:\NAVDVD\")

Remove-Item -Path "C:\NAVDVD.zip" -Force

}

And after installation the NAVDVD folder is deleted as well, as it is not needed during docker container execution.

# Appendix 1 - Some good practices I learned when using Docker

## When developing and testing scripts

Create a PowerShell script like this:

$imageNameTag = "navdocker.azurecr.io/dynamics-nav:2017"

docker rm navserver -f

docker pull $imageNameTag

docker run --name navserver `

--hostname navserver `

--volume c:\myfolder:c:\run\my `

--env accept\_eula=Y `

--env usessl=N `

--env username="vmadmin" `

--env password="P@ssword1" `

--env ExitOnerror=N `

$imageNameTag

Modify your scripts in the shared folder (c:\myfolder) and run the above script to test

This enables you to have shortcuts to access WebClient, PowerShell prompts and other things as the URLs most likely will be the same.

It also allows you to quickly try your script with other version of NAV if that should be relevant.

And it is easy to add a line like

--env ClickOnce=Y `

To enable ClickOnce. Or

--env licensefile="c:\run\my\mylicense.flf" `

To use your own license file. Or

--env auth=Windows `

To use Windows authentication, etc. etc.

# Appendix 2 – Scripts

When building, running or restarting the NAV Docker image, the c:\run\navstart.ps1 script is being run. This script will launch a number of other scripts (listed below in the order in which they are called from navstart.ps1). Each of these scripts exists in the c:\run folder. If a folder called c:\run\my exists and a script with the same name is found in that folder, then **that** script will be executed **instead** of the script in c:\run (called overriding scripts).

Overriding scripts is done by creating the script, placing it in a folder (like **c:\myfolder**) on the host, and sharing this folder to the Docker container in **c:\run\my**. You can try to create a script called AdditionalOutput.ps1 in c:\myfolder with this line:

Write-Host "This is a message from AdditionalOutput"

and run NAV on Docker with **-v c:\myfolder:c:\run\my**. You should see something like this in the output:

...   
Container IP Address: 172.25.25.115  
Container Hostname : ec54b7a5756a  
Web Client : <http://ec54b7a5756a>  
This is a message from AdditionalOutput  
  
Ready for connections!

Below you will find a list of the scripts, a description of their responsibility and in which scenario you typically would override the script.

When overriding the scripts, there are a number of variables you can/should use. Of the following 4 variables, only one is true at a time and will indicate why the navstart scripts is running.

* **$buildingImage** – this should only be true when you are building a specific image based on the generic image.
* **$restartingInstance** – this variable is true when the script is being run as a result of a restart of the docker instance.
* **$runningGenericImage** – this variable is true when you are running the generic image with a shared NAVDVD.
* **$runningSpecificImage** – this variable is true when you are running a specific image.

The following variables are used to indicate locations of stuff in the image:

* **$runPath** – this variable points to the location of the run folder (C:\RUN)
* **$myPath** – this variable points to the location of my scripts (C:\RUN\MY)
* **$NavDvdPath** – this variable points to the location of the NAV DVD (C:\NAVDVD)

The following variables are parameters, which are defined when running the image:

* **$Auth** – this variable is set to the NAV authentication mechanism based on the environment variable of the same name. Supported values at this time is Windows and NavUserPassword.
* **$serviceTierFolder** – this variable is set to the folder in which the Service Tier is installed.
* **$WebClientFolder** – this variable is set to the folder in which the Web Client binaries are present.
* **$roleTailoredClientFolder** – this variable is set to the folder in which the RoleTailored Client files are present.

Please go through the navstart.ps1 script to understand how this works and how the overridable scripts are launched.

## SetupVariables.ps1

#### Responsibility

When running the NAV Docker Image, most parameters are specified by using -e parameter=value. This will actually set the environment variable parameter to value and in the SetupVariables script, these environment variables are transferred to PowerShell variables.

#### Default behavior

The script will transfer all known parameters from environment variables to PowerShell variables, and make sure that default values are correct.

#### Override

This script will be executed as the very first thing in navstart.ps1 and you should always call the default SetupVariables script if you decide to override this script.

# do stuff

# Invoke default behavior

. (Join-Path $runPath $MyInvocation.MyCommand.Name)

# do additional stuff

#### Reasons to override

##### Hardcode variables.

Call the default SetupVariables.ps1 and then set the PowerShell variables you need afterwards (authentication, default usernames, passwords, database servers etc.)

## SetupDatabase.ps1

#### Responsibility

The responsibility of SetupDatabase is to make sure that a database is ready for the NAV Service Tier to open. The script will not be executed if a $databaseServer and $databaseName parameter is specified as environment variables.

#### Default behavior

The script will be executed when running the generic or a specific image, and it will be executed when the container is being restarted. The default implementation of the script will perform these checks:

1. If the container is being restarted, do nothing.
2. If an environment variable called bakfile is specified (either path+filename or http/https) that bakfile will be restored and used as the NAV Database.
3. If no bakfile parameter is specified and you are running the generic image, the script will restore the database from the DVD and use that as the NAV Database.
4. If no bakfile parameter is specified and you are running a specific image, the pre-installed database will be used as the NAV Database.

#### Override

If you override the SetupDatabase script, you typically would not call the default behavior.

#### Reasons to override

##### Place your database file on a file share on the Docker host

Sharing a folder from the host to the Docker instance allows you to maintain the database files outside the docker file system (See scenarios)

##### Connect to a SQL Azure Database or another SQL Server

This would probably require overriding both the SetupDatabase script and potentially other scripts as well.

## SetupCertificate.ps1

The responsibility of the SetupCertificate script is to make sure that a certificate for secure communication is in place. The certificate will be used for the communication between Client and Server (if necessary) and for securing communication to the Web Client and to Web Services (unless UseSSL has been set to N).

The script will only be executed during run (not build or restart) and the script will not be executed if you run Windows Authentication unless you set UseSSL to Y and you would typically not need to call the default SetupCertificate.ps1 script from your script.

The script will need to set 3 variables, which are used by navstart.ps1 afterwards.

# OUTPUT

# $certificateCerFile (if self signed)

# $certificateThumbprint

# $dnsIdentity

#### Reasons to override

##### Use a certificate signed by a trusted authority

If you are setting up NAV for production in a hosted environment, you probably don’t want to use a self signed certificate. A sample of how to override the SetupCertificate can be found in the scenarios section under [Use a certificate, issued by a trusted authority](#_Use_a_certificate,).

## SetupConfiguration.ps1

#### Responsibility

The responsibility of the SetupConfiguration script is to setup the NAV Service Tier configuration file. The script also needs to add port reservations if the configuration is setup for SSL.

#### Default behavior

The default behavior configures the NAV Service Tier with all instance specific settings. Hostname, Authentication, Database, SSL Certificate and other things, which changes per instance of the NAV Docker container.

#### Override

If you override the SetupDatabase script, you typically would not call the default behavior.

#### Reasons to override

##### Changes needed to the settings for the NAV Service Tier

If you need to change MaxConcurrentCalls, ClientServicesReconnectPeriod, ServicesDefaultTimeZone or other settings in the config file, which are not covered by the parameters implemented for the NAV Docker Container, then override this file, call the default behavior and make your changes.

Example:

# Invoke default behavior

. (Join-Path $runPath $MyInvocation.MyCommand.Name)

$CustomConfigFile = Join-Path $ServiceTierFolder "CustomSettings.config"

$CustomConfig = [xml](Get-Content $CustomConfigFile)

$customConfig.SelectSingleNode("//appSettings/add[@key='MaxConcurrentCalls']").Value = “10”

$CustomConfig.Save($CustomConfigFile)

## SetupAddIns.ps1

#### Responsibility

The responsibility of this script is, to make sure that custom add-ins are available to the Service Tier and in the RoleTailored Client folder.

#### Default Behavior

Copy the content of the C:\Run\Add-ins folder (if it exists) to the Add-ins folder under the Service Tier and the RoleTailored Client folder.

#### Override

If you override this script, you should execute the default behavior before doing what you need to do. In your script you should use the $serviceTierFolder and $roleTailoredClientFolder variables to determine the location of the folders.

**Note** that you can also share a folder with Add-Ins directly to the ServiceTier Add-Ins folder and avoid copying stuff around altogether.

#### Reasons to override

##### Copy Add-Ins from a different location

If your add-ins are available on a network location instead of a sharable folder, then this is where you would copy the files to the Add-ins folder of the Service Tier and the RoleTailored Client.

## SetupLicense.ps1

#### Responsibility

The responsibility of the SetupLicense script is to ensure that a license is available for the NAV Service Tier.

#### Default Behavior

The default behavior of the setupLicense script does nothing during restart of the Docker instance. Else, the default behavior will check whether the LicenseFile parameter is set (either to a path on a share or a http download location). If the licenseFile parameter is specified, this license will be used. If no licenseFile is specified, then the CRONUS Demo license is used. If you are running a specific image, the license is already imported. If you are running the generic image, the license will be imported.

#### Override

When overriding this script you are likely not to invoke the default behavior.

#### Reasons to override

##### If you have moved the database or you are using a different database

You might need to modify the way a license is imported.

##### If you want to import the license to a different location

If you need the license to not be in the NavDatabase for some reason.

## SetupWebConfiguration.ps1

#### Responsibility

The responsibility of the SetupWebConfiguration is to do final configuration changes to Web config.

#### Default Behavior

The default script is left empty, base Web Configuration is done in navstart.ps1 and cannot be overridden.

#### Override

Even though the default script is left empty, you should still invoke the default behavior if you override SetupWebConfiguration. Some of the specific images will actually add a SetupWebConfiguration script.

#### Reasons to override

Change things in the Web configuration, shich isn’t supported by parameters already.

## SetupClickOnce.ps1

#### Responsibility

The responsibility of the SetupClickOnce script is to setup a ClickOnce manifest in the download area.

#### Default Behavior

Create a ClickOnce manifest of the Windows Client

#### Override

If you override this function you should take over the full process of creating a ClickOnce manifest and you should not invoke the default behavior.

#### Reasons to override

This script is rarely overridden, but If you want to create an additional ClickOnce manifest, this is where you would do it.

## SetupClickOnceDirectory.ps1

#### Responsibility

The responsibility of the SetupClickOnceDirectory script is to copy the files needed for the ClickOnce manifest from the RoleTailored Client directory to the ClickOnce ApplicationFiles directory.

#### Default Behavior

Copy all files needed for a standard installation, including the Add-ins folder.

#### Override

If you override this script, you would probably always call the default behavior and then perform whatever changes you need to do afterwards. The location of the Windows Client binaries is given by *$roleTailoredClientFolder* and the location to which you need to copy the files is *$ClickOnceApplicationFilesDirectory*.

#### Reasons to override

##### Changes to ClientUserSettings.config

If you need to change settings in ClientUserSettings.config for the ClickOnceManifest, then invoke the default behavior and change the file in the location given by *$ClickOnceApplicationFilesDirectory*.

##### Copy additional files

If you need to copy additional files, invoke the default behavior and perform copy-item cmdlets like:

Copy-Item "$roleTailoredClientFolder\Newtonsoft.Json.dll" -Destination "$ClickOnceApplicationFilesDirectory"

## SetupFileShare.ps1

#### Responsibility

The responsibility of the SetupFileShare script is to copy files, which you want to be available to the user to the file share folder.

#### Default Behavior

Copy .vsix file (NAV new Development Environment add-in) if it exists to file share folder.

Copy self-signed certificate (if you are using SSL) to file share folder.

#### Override

You should always invoke the default behavior if you override this script (unless the intention is to not have the file share).

#### Reasons to override

##### Add additional files to the file share

Copy files need to *$httpPath*

## SetupSqlUsers.ps1

#### Responsibility

Responsibility of the SetupSqlUsers script is to make sure that the necessary users are created in the SQL Server.

#### Default Behavior

If the databaseServer is not localhost, then the default behavior does nothing, else…

If a password is specified, then set the SA password and enable the SA user for classic development access.

If you are using windows authentication and gMSA, then add the user to the SQL Database.

#### Override

If you override this script, you might or might not need to invoke the default behavior.

#### Reasons to override

##### Change configurations to SQL Server

If you need to do any configuration changes to SQL Server – this is the place to do it.

## SetupNavUsers.ps1

#### Responsibility

The responsibility of the SetupNavUsers script is to setup users in NAV.

#### Default Behavior

If the container is running *Windows Authentication*, then this script will create the current Windows User as a SUPER user in NAV. This script will also create the LocalUser if necessary you have specified username and password (i.e. if you are NOT using gMSA). If the user already exists in the database, no action is taken.

If the container is running *NavUserPassword authentication*, then this script will create a new SUPER user in NAV. If Username and Password are specified, then they are used, else a user named **admin** with a random password is created. If the user already exists in the database, no action is taken.

#### Override

If you override this script, you might or might not need to invoke the default behavior.

#### Reasons to override

##### If you are connecting to a NAV Database on another SQL Server

When connecting to a database on another server, then users probably have been created already. You can override this script with an empty script.

##### If you want to create multiple users in NAV for demo purposes

If you are using gMSA you could enumerate the users in your AD and add them to NAV as demo users.

## AdditionalSetup.ps1

#### Responsibility

This script is added to allow you to add additional setup to your Docker container, which gets run after everything else is setup. You will see, that in the scenarios, the AdditoinalSetup script is frequently overridden to achieve things.

#### Default Behavior

The default script is empty and does nothing.

#### Override

If you override this script there is no need to call the default behavior.

#### Reasons to override

##### If you need to perform additional setup when running the docker container

This script is the last scrips, which gets executed before the output section and the main loop.

## AdditionalOutput.ps1

#### Responsibility

This script is added to allow you to add additional output to your Docker container.

#### Default Behavior

The default script is empty and does nothing.

#### Override

If you override this script there is no need to call the default behavior.

#### Reasons to override

If you need to output information to the user running the Docker Container, you can write stuff to the host in this script and it will be visible to the user running the container.

## MainLoop.ps1

#### Responsibility

The responsibility of the MainLoop script is to make sure that the container doesn’t exit. If no “message” loop is running, the container will stop running and be marked as Exited.

#### Default Behavior

Default behavior of the MainLoop is, to display Application event log entries concerning Dynamics products.

#### Override

If you override the MainLoop, you would rarely invoke the default behavior.

#### Reasons to override

##### Avoid printing out event log entries

Override the MainLoop and sleep for a 100 years😊