NAV Docker Container Image

# 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>

# 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.

**After you have created a Docker environment**, you can install the Docker Powershell CmdLets, which are on GitHub here: <https://github.com/Microsoft/Docker-PowerShell>

Run:

Register-PSRepository -Name DockerPS-Dev -SourceLocation <https://ci.appveyor.com/nuget/docker-powershell-dev>  
Install-Module -Name Docker -Repository DockerPS-Dev -Scope CurrentUser

In Appendix 2 you will see some samples on how to use these CmdLets.

## 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:

docker login navdocker.azurecr.io -u 7cc3c660-fc3d-41c6-b7dd-dd260148fff7 -p G/7gwmfohn5bacdf4ooPUjpDOwHIxXspLIFrUsGN+sU=

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

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 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 CU8 all languages
* NAV 2017 CU7 W1
* NAV 2017 CU7 DK
* NAV 2016 CU20 W1
* devpreview july update

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
* 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.

## Connect a NAV Container to another Database server

TODO

## Connect a NAV Container to an Azure SQL database

TODO

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

TODO

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

TODO

## Change the Startup Parameters of the Windows Client running ClickOnce

TODO

## Build your own Docker Image based on the Generic image

TODO

## Build your own Docker Image based on a Specific image

TODO

## More scenarios?

TODO

# Appendix 1 – 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 an SQL Azure Database

This would probably require overriding both the SetupDatabase script and the SetupConfiguration script.

## 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.

## 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.

### 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😊

# Appendix 2 – Example of usage of the Docker CmdLets

If you need to automate the creation of Docker environments, there is nothing like PowerShell.

For this, you can install the Docker Powershell CmdLets, which are on GitHub here: <https://github.com/Microsoft/Docker-PowerShell>

Run:

Register-PSRepository -Name DockerPS-Dev -SourceLocation <https://ci.appveyor.com/nuget/docker-powershell-dev>  
Install-Module -Name Docker -Repository DockerPS-Dev -Scope CurrentUser

Below are some of the scenarios I found myself using PowerShell for over and over again:

## List all docker containers

To get a list of all docker containers use:

Get-Container

This will show all containers (running and exited).

## Remove all docker containers

To remove all containers, you can pipe the result of Get-Container into Remove-Container.

Get-Container | Remove-Container -Force

It is more likely that you will have a where clause and only remove containers, which are not running:

Get-Container | Where-Object { $\_.State -ne 'running' } | Remove-Container -Force

## Pull a new or updated Docker image from the registry

If you are using a private Docker registry (like navdocker.azurecr.io), the Docker CmdLets do not reuse the Docker Login credentials, and you will have to provide those as a parameter.

$authconfig = New-Object Docker.DotNet.Models.AuthConfig

$authconfig.Username = "7cc3c660-fc3d-41c6-b7dd-dd260148fff7"

$authconfig.Password = "G/7gwmfohn5bacdf4ooPUjpDOwHIxXspLIFrUsGN+sU="

Pull-ContainerImage -Repository "navdocker.azurecr.io/dynamics-nav" -Tag "2017" -Authorization $authconfig

If you are pulling from the Docker hub you can avoid the authorization parameter.