Hands-On Lab: Easy deployment of NAV using Docker

# Who should complete this HOL?

This Hands-On Lab is designed to help you understand what Docker is and what NAV on Docker can do for you. After completing the HOL, you should be able to determine if Docker and especially NAV on Docker is useful in your organization. The HOL will use the Workshop VMs as a foundation for the HOL to have a uniform platform for all.

When you have completed this HOL, you can find more info on the nav-docker project on github: <http://www.github.com/microsoft/nav-docker>. This is also the place you should be filing issues and comments.

# What is Docker?

If you are new to Docker and Containers, you might want to scan through this document before heading into the workshop:

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

This should give you a better understanding of what Docker is.

For the remaining of the workshop, you will be going through some scenarios, using Docker, on how to Deploy NAV.

When you connect to your learning environment, you are presented with a website, which looks like the image on the right side. What you might not be aware is, that when you are viewing this, you are already using Docker. This website is hosted inside a Container, which is running inside an Azure Virtual Machine, called the Docker host and in this case, your Workshop VM.

Clicking the Connect link will download the .rdp file, which launches Remote Desktop to the Workshop VM.

Note that all other workshops at Directions are using the same Workshop VM, but they just don’t care that NAV is running in a Container.

Note also that the Workshop VMs are very much like the NAV Developer Preview VMs with the September Update you can get from <http://aka.ms/navdeveloperpreview>.

Note also that you will not and cannot connect to a remote desktop in the Container. The Container is based on WindowsServerCore, which has no UI, no desktop.

Connecting to the Workshop VM (the Docker host) will allow you to interact with the Docker Containers that are available on that machine by using various commands.

First thing we will do is to have a look at the Workshop VM desktop and what we can do with that.

|  |  |
| --- | --- |
| Open your workshop landing page in a browser and press the Connect button to connect to the remote desktop of your workshop environment. Use the provided credentials to login.  Close Server manager and other windows that pop up and you should have a desktop, with a few shortcuts. |  |
| **Command Prompt**  The Command Prompt is the standard CMD.EXE running as Administrator.  This prompt is primarily there for running Docker commands or other executables.  We will be using the Command Prompt throughout this Hands On Lab. |  |
| **Container Command Prompt**  The Container Command Prompt is the standard CMD.EXE running inside the container.  When you run dir inside the Container Command Prompt you will see the Container file system. Folders that are shared from the host to the container are shown as symbolic directory links (SYMLINKD).  The file system inside the NAV Docker Image consists of a few special folders/files:  **c:\run** the run folder is the folder containing all the scripts, which are used to set up NAV in the container.  **c:\run\my** is the location, where you can place scripts which can override functionality of the run folder. Typical scenario is to share a folder from the host to the c:\run\my folder, containing various scripts that you want executed during start.  **c:\run\start.ps1** is the entry point for the container.  **c:\run\navstart.ps1** is the main script for setting up NAV and launching other setup scripts. |  |
| **Container PowerShell Prompt**  The Container PowerShell Prompt is a PowerShell prompt running inside the container.  All NAV cmdlets are loaded inside the PowerShell prompt, ready to use.  Example:  **Get-NavServerUser NAV**  Will list all users in the NAV server instance (which is the default server instance in the container).  Note that not all commands will work inside the container. You cannot create a new server instance, for example – that is done by spinning up another container (the Docker way😊) |  |
| **FinSql**  C/SIDE a.k.a. the Classic Development Environment.  Note that C/SIDE is not there to support all classic development scenarios.  The primary reason for C/SIDE to be available is for the VS Code developer to be able to see and browse through the source of the base application.  Having said that, you can actually do the majority of classic development scenarios in C/SIDE.  Note that when you start C/SIDE you will be running Database Authentication and you have to login as **SA** and use the Workshop VM password.  Server name is **navserver\SQLEXPRESS** and the database name depends on which localization you are running. |  |
| **Web Client**  Opens a browser with the Web client. The Web client is installed inside the container on IIS and the ports are exposed on the container and published to the host.  Login user name is **student** and the Workshop VM password is your password. |  |
| **Windows Client**  Opens the Windows client.  The Windows client is not installed on the Docker host even though it looks like it.  The Docker host shares a folder to the container called C:\Program Files (x86)\Microsoft Dynamics NAV – and the container then copies the files from that folder to the host.  This gives the best compatibility and allows the folder to be overridden if deploying a new container.  Login user name is **student** and the Workshop VM password is your password.  You can also install the Windows client using ClickOnce. There is a section about this later in the HOL. |  |
| **Landing Page**  The landing page was the starting point of your journey. You will find all info and links here necessary to connect and use the Workshop VM. | |
| **PowerShell ISE**  PowerShell ISE running on the Docker host. This is every IT infrastructure gurus favorite tool and we will be using ISE throughout this Hands On Lab. | |
| **Visual Studio Code**  Visual Studio Code is used for AL development and is not used in this Hands On Lab.  When launching the Workshop VM, the AL Language extension from the landing page is preinstalled. If you deploy a new NAV Container, you will have to uninstall and install a new AL Language extension. | |

# Basic Docker commands

Let’s drill into some of the basic Docker commands to get a better understanding of what Docker is and how it works.

You can run these commands in PowerShell, but Docker is a simple Windows Executable and will run in a command prompt as well.

For simplicity reasons, we will use the Command Prompt.

|  |  |
| --- | --- |
| Open the Command Prompt and write:  **docker ps**  This gives you a list of all the running Docker Containers on your machine.  Take some time to Inspect the info:   * The container name is navserver. * The container ID starts with 755757f82076. * The container is based on the navdocker.azurecr.io/dynamics-nav:devpreview-finus image. * Ports 443 and 7046-7049 are all exposed on the Docker host, meaning you can access them from the outside. * Port 8080 from the Docker container is published as port 80 on the Docker host. * Ports 80, 1433 and 7045 are open for the host. |  |
| You might wonder why the previous section says: *“The container ID starts with…”*. The reason for this is, that the ID really is a 64 digit globally unique hex identifier, but most time you can refer to the ID by specifying the first digits until your specification isn’t ambiguous.  You will get the full ID by typing:  **docker ps --no-trunc**  but if you only have one image you can identify it by writing **7** |  |
| The next command to try is:  **docker images**  This gives you a list of all images available for you to run.  In this picture there are the 2 Microsoft base images: Windows Server Core and Nano Server. Beside them, the latest devpreview image with Financials US localization.  A Docker image is really a set of services installed in a box (container) ready to run on demand.  A specific version of the NAV Docker image is a specific version (incl. localization) of NAV installed in a Container ready to run (ex. NAV 2017 CU7 DK).  The NAV Docker images are highly configurable and customizable. |  |
| Now, try to run another instance of the dynamics-nav image you have available:  **docker run navdocker.azurecr.io/dynamics-nav:devpreview-finus**  As the error indicates, you will have to accept the End User License Agreement before this container can start.  Use:  **docker inspect --format='{{.Config.Labels.legal}}' navdocker.azurecr.io/dynamics-nav:devpreview-finus**  to view the legal documents for the preview. |  |
| Let’s run another instance of the image and accept the EULA:  **docker run -e accept\_eula=Y navdocker.azurecr.io/dynamics-nav:devpreview-finus**  Press Ctrl+C in the command prompt to exit the container and leave it running in the background.  Now, run:  **docker ps**  The command will show you two containers running. Inspect the difference in names, ports etc.  Note that Docker automatically assigns a readable name to the container if you don’t assign a name in the Docker run statement.  The original container will have ports exposed on the host, the new container will only have ports exposed on the container. |  |
| Open the Web client in a browser. Ignore the certificate warnings for the self-signed certificate. |  |
| Now try to run  **docker ps -a**  which will show you all containers – running ones and exited ones. If you did try to run a container earlier without specifying the accept\_eula=Y then you will have an exited container in the list.  Remove the dead container using  **docker rm <containerid>**  If you want to remove a running container you either need to stop it first or use the -f parameter:  **docker rm <contianerid> -f** |  |
| Use  **docker inspect navserver**  to inspect settings, status, labels etc. on a container or an image.  You will also find network settings etc. if you look through the emitted JSON. |  |
| Use  **docker stats**  to get statistics from the currently running containers. |  |
| If you dislike the format of Docker stats (if you would like the container name included) you can modify the output by specifying a statsFormat property in the c:\users\student\.docker\config.json file.  In VSCode, create a new file with this content:  {  "statsFormat": "table {{.Name}}\t{{.CPUPerc}}\t{{.MemUsage}}\t{{.NetIO}}\t{{.BlockIO}}"  }  And save it in c:\users\student\.docker\config.json.  Now re-run  **docker stats**  and you will see the info requested.  You can also add a section for psFormat etc. |  |

# Use PowerShell ISE to modify files in the container

|  |  |
| --- | --- |
| If you are new to Docker you might not yet be annoyed over how cumbersome it is to modify files in the container.  You can connect using the PowerShell prompt or the command prompt, but since the file system is remote and you don’t have a UI, you cannot edit files using Notepad.  But…  You can use ISE – it just requires a small trick.  Open ISE and run  Enter-PSSession -ContainerId (docker ps --no-trunc -qf "name=navserver")  Now you will enter a remote session in PowerShell (much like the Container PowerShell Prompt) and inside of this you can use psEdit to edit files remotely without having to share folders and copy back and forth.  Note that psEdit is an ISE specific function and does NOT work inside the Container PowerShell Prompt. |  |

# Advanced parameters

When using Docker run with the NAV image, there are a lot of different parameters you can use. All NAV image specific parameters are specified as environment variables (-e or –env).

|  |  |
| --- | --- |
| There are a number of different parameters you can set when running the NAV Container. This command uses some of them:  **docker run -e accept\_eula=Y -e usessl=N -e auth=Windows -e username=student -e password=<password> --name test navdocker.azurecr.io/dynamics-nav:devpreview-finus**  If you specify the password of your student user, then this command will start NAV in a Container without SSL and using Windows Authentication.  Note that this is a known hack, that you can use Windows Authentication between two machines if they share the same username and password.  **Note** that the Web client is now without SSL and if you open it in a browser, you will find that you are logged directly into NAV. Use:  **docker rm test -f**  to remove the container named test. |  |
| In the above example, test is the container name.  Most Docker commands take container ID or container name as parameter.  The container name however is not added to the DNS resolver and you cannot ping the container name.  In order to access the container using TCP or HTTP you need to use the hostname.  The default hostname is the first 10 characters of the container ID.  You can specify your own hostname using:  **--hostname test**  Docker will automatically maintain the IP address in the DNS resolution for the hostname, locally on the host.  You can also specify a public DNS name, which is the CNAME record, which points to your host if you are exposing the container to the world using a trusted certificate. PublicDnsName will default to the hostname.  **-e publicDnsName=s11.navdemo.net**  If you do not use SSL, the publicDnsName is only used for calculating properties like PublicWebBaseUrl, PublicSoapBaseUrl etc. in the config file. |  |

# Pulling new NAV on Docker Images

Up until now, you have only used the images that already exist on the VM. If you want to use non-existing images or if you want to get an update to an existing image, you will have to pull these images. If you pull dynamics-nav:2017 without any specific cu you will always get the latest cu. Pulling dynamics-nav:2017-cu8 will only be updated if there are updates to the Windows Server Core or the NAV Generic Image.

|  |  |
| --- | --- |
| Currently all NAV images resides in a private registry and you will need to login to get access. Use this command to get access to all images on navdocker.azurecr.io:  **docker login navdocker.azurecr.io -u 7cc3c660-fc3d-41c6-b7dd-dd260148fff7 -p G/7gwmfohn5bacdf4ooPUjpDOwHIxXspLIFrUsGN+sU=** |  |
| After logging in to the private registry, you can try to pull an image, which is not already on the VM.  Docker images are constructed in layers. That means a Docker pull will only need to download those layers that are different from already downloaded layers. Try:  **docker pull navdocker.azurecr.io/dynamics-nav:devpreview-findk**  You will see, that the first ~18 layers already exists and only a few layers will need to be downloaded.  The ~5 layers, which are downloaded is the difference between the US localization and the DK localization, so only this difference will have to be downloaded.  The common layers are:   * WindowsServerCore * Dynamics NAV Generic Image * Devpreview W1 Image |  |
| As you might have guessed by now – if US and DK includes W1 (are built on top of W1), pulling W1 should not cause any downloads. Try:  **docker pull navdocker.azurecr.io/dynamics-nav:devpreview**  Indeed – nothing to download, all layers already exist. |  |

# Using Docker commands from PowerShell

|  |  |
| --- | --- |
| Even though Docker is a command line executable, you can use it in PowerShell like other executables and it does have some advantages.  When working on a project, I typically create a PowerShell script to refresh my NAV Container, install licenses etc.  Create a folder in your VM called c:\test.  Start PowerShell ISE in your VM and copy the script here to a new window in PowerShell ISE and save the script in c:\test\start.ps1.  Run the script using F5. | $imageNameTag = "navdocker.azurecr.io/dynamics-nav:devpreview-finus"  $username = "student"  $password = "P@ssword1"  $locale = "en-US"  $licensefile = "license.flf"  $name = "myserver"  Copy-Item -Path "c:\demo\$licensefile" -Destination (Join-Path $PSScriptRoot $licensefile) -Force  New-Item -Path (Join-Path $PSScriptRoot "my") -ItemType Directory -Force -ErrorAction Ignore | Out-Null  docker ps -a --filter name=$name -q | % {  Write-Host "Remove container $name"  docker rm $\_ -f | Out-Null  }  Write-Host "Pull new container image for $imageNameTag"  docker pull $imageNameTag  Write-Host "Run container"  $id = docker run `  --name $name `  --hostname $name `  --env accept\_eula=Y `  --env useSSL=N `  --env auth=NavUserPassword `  --env username=$username `  --env password=$password `  --env ExitOnError=N `  --env licenseFile="C:\TEST\$licensefile" `  --detach `  --volume ${PSScriptRoot}:c:\TEST `  --volume ${PSScriptRoot}\my:c:\run\my `  --env locale=$locale `  $imageNameTag  if ($LastExitCode -ne 0) { throw "Docker run error" }  Write-Host "Waiting for container to be ready, this shouldn't take more than a few minutes"  Write-Host "Time: ½ 1 ½ 2"  $cnt = 150  $log = ""  do {  Write-Host -NoNewline "."  Start-Sleep -Seconds 2  $logs = docker logs $name  if ($logs) { $log = [string]::Join(" ",$logs) }  if ($log.Contains("<ScriptBlock>")) { $cnt = 0 }  } while ($cnt-- -gt 0 -and !($log.Contains("Ready for connections!")))  if ($cnt -gt 0) {  Write-Host "Ready"    Write-Host "Copy new .vsix file to $PSScriptRoot"  remove-item "$PSScriptRoot\\*.vsix" -force  docker exec -it $name powershell "Copy-item -Path 'C:\Run\\*.vsix' -destination 'C:\TEST' -force"    Write-Host "Start Web Client"  Start-Process -FilePath "http://$name/NAV/WebClient"    Write-Host "Start NAV Container prompt"  Start-Process "cmd.exe" @("/C";"docker exec -it $name powershell -noexit C:\Run\prompt.ps1")  }  Write-Host "Container Output:"  docker logs $name | % { Write-Host $\_ } |
| You should see an output, which is like the output on the right here.  You can press F5 again and again and the script will automatically remove the old container and start a fresh.  It becomes very easy to try out other parameters and re-deploy the NAV Container.  Please inspect the script and understand:   * How the license file is shared to the container? * How the .vsix file is made accessible from the host? * How containers are removed? * How images are pulled? * How the NAV Container prompt has all NAV cmdlets available?   Feel free to modify the script a few times to better understand how these things works or navigate around inside the container. BTW - you might want to type c:\run\prompt.ps1 inside the container shell 😊. | PS C:\demo> C:\TEST\start.ps1  Remove container myserver  Pull new container image for navdocker.azurecr.io/dynamics-nav:11.0.18426.0-finus  11.0.18426.0-finus: Pulling from dynamics-nav  Digest: sha256:e54d0440f1f3e4cdc012282fc5efe211e3b32d0e6451456a67bc3cfe011e124d  Status: Image is up to date for navdocker.azurecr.io/dynamics-nav:11.0.18426.0-finus  Run container  Waiting for container to be ready, this shouldn't take more than a few minutes  Time: ½ 1 ½ 2  ................................Ready  Copy new .vsix file to C:\TEST  Start Web Client  Start NAV Container prompt  Container Output:  Initializing...  Hostname is myserver  PublicDnsName is myserver  Running Specific Image  Starting Local SQL Server  Starting Internet Information Server  Using NavUserPassword Authentication  Using Database Connection localhost/SQLEXPRESS [FinancialsUS]  Modifying NAV Service Tier Config File for Docker  Creating Self Signed Certificate  Self Signed Certificate Thumbprint F24E70A9A8FB205EB973EAA5606115D592B1F1CA  Modifying NAV Service Tier Config File with Instance Specific Settings  Start NAV Service Tier  Using license file 'C:\TEST\license.flf'  Import NAV License  Creating DotNetCore NAV Web Server Instance  Creating http download site  Creating Windows user  Enabling SA  Creating NAV user  Container IP Address: 172.19.36.42  Container Hostname : myserver  Container Dns Name : myserver  Web Client : http://myserver/NAV/WebClient/  Dev. Server : http://myserver  Dev. ServerInstance : NAV  Files:  http://myserver:8080/al-0.10.13063.vsix  Ready for connections! |
| Try to modify the password to match the password of your Workshop VM and modify the authentication to Windows Authentication – and then re-run the script.  The Web client now should use Windows Authentication and automagically log you in. | $password = "<your VM password>"  --env auth=Windows ` |

# Using a different database server

Up until now, we have been using NAV in a Container with the database living inside the same container. That is convenient when doing demos, but frequently you probably want to run the database on a different SQL Server or maybe even on Azure SQL. It is very easy to point out a different database server, instance, and name on the command line and if your containers are set up with gMSA (Group Managed Service Accounts) and Windows Authentication this should be sufficient to connect.

**BUT…**

The Workshop VMs do NOT use gMSA (as this would require the setup of AD servers), meaning that connecting to a different database server requires a little more work. What you will do here is to override the SetupDatabase script. Overriding scripts is an important feature of the NAV on Docker Images, and it really provides the flexibility needed for setting up a system as complex as NAV.

In the end we’ll have our new container ”myserver” running NAV and we’ll just reuse the first container ”navserver” as a database server. We could also use a new container containing only SQL Server, but we’ll be faster this way.

|  |  |
| --- | --- |
| In the start.ps1 script, change the auth parameter to NavUserPassword and add the three database parameters as specified here. | --env auth=NavUserPassword `  --env databaseServer=navserver `  --env databaseInstance=SQLEXPRESS `  --env databaseName=FinancialsUS ` |
| Copy this script, paste it into PowerShell ISE, modify the “<your password>” with the password for your Workshop VM, and save the script as c:\test\my\SetupDatabase.ps1.  Inside the container, the primary setup runner (**c:\run\navstart.ps1**), will call several scripts in the c:\run folder to setup variables, database, license etc.  For each of these scripts, it will check whether there is a script in **c:\run\my** which overrides the behavior. The overriding script can determine if it is necessary to call the default behavior, but it needs to take over the responsibility of the script.  The NAV on Docker documentation talks about all the different scripts you can override and typical scenarios for why you would want to do it. | if (!$RestartingInstance) {    Write-Host "Change Database Connection"  $DatabaseUserName = "sa"  $DatabasePassword = "<your password>"  $EncryptionPassword = "1234abcd!1234abcd"  $TrustSQLServerCertificate = $true  $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  } |
| Switch back to the start script and press F5.  If you inspect the log, you will see that the myserver container never starts the local SQL Server.  Try to connect to the original Web client (on the Desktop) and open <http://myserver/nav> at the same time.  The highlighted section on the right are all outputs, which is a result of the script override. | Initializing...  Hostname is myserver  PublicDnsName is myserver  Running Specific Image  Starting Internet Information Server  Using NavUserPassword Authentication  Change Database Connection  Import Encryption Key  WARNING: You should encrypt the connection to the database to help protect  against network attacks, such as man-in-the-middle attacks.  To encrypt the connection, select the Enable Encryption on SQL Server  Connections setting (EnableSqlConnectionEncryption = True) and clear the Enable  Trust of SQL Server Certificate setting (TrustSQLServerCertificate = False).  WARNING: The new settings value will not take effect until you stop and restart  the service.  Modifying NAV Service Tier Config File for Docker  Creating Self Signed Certificate  Self Signed Certificate Thumbprint 4FC627D76406C87FBD909DF7BBE254DB190C4C3E  Modifying NAV Service Tier Config File with Instance Specific Settings  Start NAV Service Tier  Using license file 'C:\TEST\license.flf'  Import NAV License  Create DotNetCore NAV Web Server Instance  Creating http download site  Creating Windows user  Container IP Address: 172.19.35.115  Container Hostname : myserver  Container Dns Name : myserver  Web Client : http://myserver/NAV/WebClient/  Dev. Server : http://myserver  Dev. ServerInstance : NAV  Files:  http://myserver:8080/al-0.9.12794.vsix  Ready for connections! |
| Running **docker stats** now reveals two containers and the one running SQL Server and NAV uses more memory than the one running NAV only. |  |
| Modify the **start.ps1** script and add the following parameters to Docker run. Telling the container to NOT set up the Web client and NOT deploy an HTTP download site causes the container to use ~300MB memory less and startup is reduced with ~20 seconds (IIS is never started). | --env WebClient=N `  --env httpSite=N ` |
| Running Docker stats indicates that |  |

# Running any version of NAV 2016 or NAV 2017

|  |  |
| --- | --- |
| Create a folder called c:\2017 and create a PowerShell script called start.ps1 in this folder, with this content, where “<your password>” is replaced with the password for your Workshop VM.  This will start the latest NAV 2017 W1 on your machine. | $imageNameTag = "navdocker.azurecr.io/dynamics-nav:2017"  $username = "student"  $password = "<your password> "  $name = "NAV2017"  docker ps -a --filter name=$name -q | % {  Write-Host "Remove container $name"  docker rm $\_ -f | Out-Null  }  Write-Host "Pull new container image for $imageNameTag"  docker pull $imageNameTag  Write-Host "Run container"  $id = docker run `  --name $name `  --hostname $name `  --env accept\_eula=Y `  --env useSSL=N `  --env auth=Windows `  --env username=$username `  --env password=$password `  --detach `  --env ExitOnError=N `  --volume ${PSScriptRoot}:c:\TEST `  $imageNameTag  if ($LastExitCode -ne 0) { throw "Docker run error" }  Write-Host "Waiting for container to be ready, this shouldn't take more than a few minutes"  Write-Host "Time: ½ 1 ½ 2"  $cnt = 150  $log = ""  do {  Write-Host -NoNewline "."  Start-Sleep -Seconds 2  $logs = docker logs $name  if ($logs) { $log = [string]::Join(" ",$logs) }  if ($log.Contains("<ScriptBlock>")) { $cnt = 0 }  } while ($cnt-- -gt 0 -and !($log.Contains("Ready for connections!")))  if ($cnt -gt 0) {  Write-Host "Ready"    Write-Host "Start Web Client"  Start-Process -FilePath "http://$name/NAV/WebClient"    Write-Host "Start NAV Container prompt"  Start-Process "cmd.exe" @("/C";"docker exec -it $name powershell -noexit C:\Run\prompt.ps1")  }  Write-Host "Container Output:"  docker logs $name | % { Write-Host $\_ } |
| Add a cumulative update number to the imageNameTag and re-run in order to run a specific CU on your machine. | $imageNameTag = "navdocker.azurecr.io/dynamics-nav:2017-cu7" |
| Add a country version to the imageNameTag and re-run and you will be running the local version on your machine.  Omitting the -cu7 tag will give you the latest NAV 2017 DK. | $imageNameTag = "navdocker.azurecr.io/dynamics-nav:2017-cu7-dk" |

# Adding C/SIDE and the Windows Client through ClickOnce

|  |  |
| --- | --- |
| The NAV Docker image also supports deploying the Windows client using ClickOnce and it is as easy as adding a parameter to the Docker run statement. | --env ClickOnce=Y ` |
| In the output section of the container you will find a URL to the ClickOnce manifest, which you can open in a web browser, download the prerequisites and run the Windows client. | ...  Creating NAV user  Creating ClickOnce Manifest  Container IP Address: 172.19.47.219  Container Hostname : NAV2017  Container Dns Name : NAV2017  Web Client : http://NAV2017/NAV/WebClient/  ClickOnce Manifest : http://NAV2017:8080/NAV  ... |

# Portainer.io

Portainer is a free GUI for maintaining your Docker environment.

|  |  |
| --- | --- |
| Portainer doesn’t work with IE and Edge doesn’t run on Windows Server 2016, so we need to download and install Chrome on the Workshop VM from: | <https://www.google.com/intl/en/chrome/browser/> |
| Copy the PowerShell script, paste it into PowerShell ISE and run it. The script will:  1. Reconfigure Docker deamon  2. Open port 2375 in the firewall  3. Create a Portainer directory  4. Get the IP address  5. Download and run the Portainer Docker image | '{  "hosts": ["tcp://0.0.0.0:2375", "npipe://"]  }' | Set-Content "C:\ProgramData\docker\config\daemon.json"  restart-service docker  netsh advfirewall firewall add rule name="Docker" dir=in action=allow protocol=TCP localport=2375  new-item -Path "C:\Portainer" -ItemType Directory  $ipAddress = (get-netadapter | Select-Object -First 1 | get-netipaddress | ? addressfamily -eq 'IPv4').ipaddress  docker run -d -v C:\Portainer:C:\data --name portainer --hostname portainer portainer/portainer -H tcp://${ipAddress}:2375 |
| Open Google Chrome and navigate to  <http://portainer:9000>  On your first connection, you will have to create an admin password for Portainer.  After that…  **Welcome to a free tool for maintaining your Docker environment** |  |